Идеи, определившие облик информатики

Классические статьи по компьютерным наукам

Под редакцией Гарри Р. Льюиса

ДМК ПРЕСС издательство

«Я могу предсказать твое будущее. Нет ничего проще… Но кто может знать твое прошлое?.. Что оно значило? Что оно пыталось сказать тебе?»

Торнтон Уайльдер, «На волосок от гибели»

В книге собрано 46 классических статей по информатике, которые прочертили карту развития этой науки.

Охвачены все аспекты компьютерных наук; особое внимание уделяется периоду с 1936 по 1980 год, но рассматриваются и важные ранние работы.

Знакомя читателей с работами мыслителей от Аристотеля и Лейбница до Алана Тьюринга и Норберта Винера, книга документирует открытия и изобретения, приведшие к созданию современного цифрового мира. Каждую статью сопровождает краткий очерк, где представлен исторический и интеллектуальный контекст, за авторством Гарри Льюиса, редактора издания.

Гарри Рой Льюис *– американский ученый, математик, профессор Гарвардского университета, известный своими исследованиями в области вычислительной логики и теоретической информатики. Удостоен награды за выдающийся вклад в обучение студентов. Среди учеников Гарри Льюиса Билл Гейтс и Марк Цукерберг, а также многочисленные будущие преподаватели Гарварда и других образовательных учреждений. Веб-сайт Six Degrees to Harry Lewis, созданный Цукербергом во время учебы в Гарварде, предвосхитил появление сети Facebook.*

The MIT Press

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
books@alians-kniga.ru

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО
www.дмк.рф

ISBN 978-5-93700-208-2
9 785937 002082 >

Идеи, определившие облик информатики

Классические статьи по компьютерным наукам

Под редакцией Гарри Р. Льюиса

ДМК Пресс
издательство
Москва, 2023

УДК 004.04
ББК 32.372
И29

И29 Идеи, определившие облик информатики / под ред. Гарри Р. Льюиса; пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2023. – 616 с.: ил.

ISBN 978-5-93700-208-2

В книге собрано 46 классических статей по информатике, которые прочертили карту развития этой науки. Охвачены все аспекты компьютерных наук; особое внимание уделяется периоду с 1936 по 1980 год, но рассматриваются и важные ранние работы. Знакомя читателей с работами мыслителей от Аристотеля и Лейбница до Алана Тьюринга и Норберта Винера, книга документирует открытия и изобретения, приведшие к созданию современных цифровых технологий. Каждую статью сопровождает краткий очерк, где представлен исторический и интеллектуальный контекст, за авторством Гарри Льюиса, редактора издания.

Книга будет полезна всем, кто интересуется информатикой и истоками этого увлекательного мира.

УДК 004.04
ББК 32.372

The MIT Press Cambridge, Massachusetts London, England.
The rights to the Russian-language edition obtained through Igor Korzhenevskiy of Alexander Korzhenevski Agency (Moscow).

ISBN 978-0-26204-530-8 (англ.)
ISBN 978-5-93700-208-2 (рус.)

Посвящается
тем экстравагантным и скептически
настроенным учителям, которые показали мне новые места:
Филу Бриджесу, Джиму Магуайру, Вэну Эллиоту,
Десмонду О'Грейди, Шейле Грейбах, Тому Читэму,
Айвену Сазэрленду и Бэрту Дребену

Я могу предсказать твое будущее. Нет ничего проще. …
Но кто может знать твое прошлое? … Что оно значило?
Что оно пыталось сказать тебе?

– Торнтон Уайльдер «The Skin of Our Teeth»

Содержание

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Предисловие

Эта книга – рассказ об информатике словами тех, кто ее создавал. Появилась она по двум причинам. Во-первых, чтобы избавить читателей, живущих в XXI веке, от иллюзии, будто сложившиеся в этой области соглашения были дарованы современной культуре в готовом виде. Информатика имеет богатую семейную историю, которую следует знать студентам и всем пашущим на этой ниве. А во-вторых, чтобы помочь читателям разглядеть, как появлялись на свет важнейшие идеи, как робкие, неуклюжие шажки постепенно превращались в твердую и уверенную поступь – а иногда в течение многих лет никуда не приводили, чтобы возобновить движение после длительного перерыва. Информатика – все еще юная и динамичная наука; всякий осматривающийся в ней сегодня может разглядеть какое-то новшество, которое завтра станет каноном, но пока находится в зачаточном состоянии и едва различимо.

Чтобы рассказать вам эту историю, я отобрал, трепеща в душе, 46 статей, начиная с античных времен и заканчивая 1980 годом, и предпослал каждой краткий очерк с описанием контекста. Каждая из этих статей внесла запоминающийся вклад в свою область. Можно было бы отобрать много других работ в дополнение к этим или вместо них, да и дата отсечения, 1980 год, выбрана достаточно произвольно, хотя она все же представляет тот момент, после которого информатика стала настолько разветвленной, что попытка составить такой небольшой сборник заведомо потерпела бы неудачу.

Эта книга написана только в образовательных целях, чтобы документировать истоки научной области; она не является ни критическим переизданием статей, ни историей вопроса. Во введении статьи перечисляются в историческом контексте, но для тех читателей, кто жаждет более подробного освещения истории событий со всеми нюансами, рекомендую работу Priestley (2011), автору которой удалось удачно избежать излишнего доверия к своекорыстным воспоминаниям действующих лиц и заблуждений типа «после того значит вследствие того». За сведениями о ранней истории вычислительных машин отсылаю читателя к работам Pratt (1987) и Jones (2016).

Эту книгу можно использовать в качестве основы односеместрового курса для студентов старших курсов и аспирантов, и в таком качестве она действительно использовалась в Гарварде и МТИ. Она также может послужить путеводителем для любопытствующих профессионалов. При отборе и редактировании статей я руководствовался, в частности, следующими принципами.

1. Многие статьи значительно сокращены, чтобы привлечь внимание к основным идеям, опуская технические детали, которые сегодня уже не представляют большого интереса. Например, я опустил далекие от элегантности детали кода универсальной машины Тьюринга. Опущенный текст обозначается многоточием <...>. Нет нужды говорить, что читатели, жадные до деталей, могут обратиться к полному тексту статей. Каждая глава начинается библиографической отсылкой к статье, хотя указанный в ней год может не совпадать с годом, указанным в заголовке самой статьи, потому что некоторые работы сначала были представлены устно или были переработаны после первой публикации. Кроме того, изобретение или открытие, описываемое в статье, могло иметь место раньше даты публикации.
2. Я отдавал предпочтение коротким и удобным для восприятия статьям перед длинными и трудными, пусть даже более важными.
3. Я включал только статьи, но не отрывки из книг (за исключением «Законов мышления» Буля и очерка, давшего название сборнику Брукса «Мифический человеко-месяц»).
4. Я не старался дать выжимку из технических отчетов, в которых определены такие важнейшие языки программирования, как FORTRAN, COBOL или ALGOL, как бы важны они ни были для развития отрасли.
5. Также опущены ранние усилия по систематизации материала, в частности доклад *Curriculum 68* (Atchison et al., 1968) и история языков программирования в работе Jean Sammet (1972).
6. Количество страниц, включенных в книгу, не может считаться мерой значимости автора. При таком критерии Дональд Кнут был бы недооценен, а Эдсгер Дейкстра – переоценен. С другой стороны, такие великие имена, как Бэкус, Хомский, Чёрч, Флойд, Грэй, Клини, Ньюэлл, Лэмпорт, Лэмпсон, Рабин, Скотт и Тарьян, вообще не упомянуты. Приношу извинения тем, кто не нашел здесь своей любимой статьи или автора.

И еще несколько чисто редакторских замечаний.

- Все ссылки собраны в общей библиографии в конце книги, при их оформлении используется Чикагское руководство по стилю. Документы, которые цитируются в оригинальных статьях, но не в той части текста, которая включена в книгу, в библиографию не включены.
- Опечатки, встречающиеся в оригинальных статьях, исправлены, пунктуация приведена к единому стандарту.
- Нумерация разделов и рисунков единая во всей книге: § 33.7 – это раздел 7 (или, возможно, седьмой ненумерованный раздел) оригинальной статьи, которая здесь напечатана в главе 33.
- Оригинальная нумерация теорем и лемм сохранена.
- Сноски опущены или включены прямо в текст в скобках.
- В нескольких местах я добавил примечания редактора, оформленные в виде «[Примечание редактора: комментарий]».

Я выражаю благодарность слушателям курса CS191, прочитанного в 2019 году в Гарвардском университете, и курса 6.S897, прочитанного в 2020 году

в Массачусетском технологическом институте (МТИ), за тщательную корректуру. Особенно остроглазыми проявили себя Брайан Сапожников и Адхем Мегид. Спасибо также Питеру Дэннингу, Биллу Газарху, Уоррену Гольдфарбу, Мэттью Лена, Марианте Маллиарис, Таше Шенстейн, Ллойду Стриклэнду, Шерри Тэркл и Джоэлю Вахману за полезные замечания и поправки. Разумеется, за все оставшиеся ошибки ответственность несу только я.

Гарри Льюис
Июль 2020

Введение: корни и рост информатики

Все начиналось с чисел; инструментами первых бухгалтеров были кучки *calculi* (камушков). Более поздние нотационные и механические изобретения были придуманы астрономами, а также военными инженерами и мореплавателями. Нужда в вычислениях возрастала по мере того, как люди учились понимать физический мир и управлять им, особенно в эпоху Просвещения и во время последующих войн.

Но интеллектуальные истоки информатики не сводятся только к бухгалтерии, астрономии и баллистике. Информатика – дитя логики, математики и человеческого воображения. В силу столь разнородной интеллектуальной родословной и лишь косвенной связи с миром природы эта область знаний с трудом отвоевывала свое право на существование на протяжении большей части XX столетия. К тому времени компьютерные вычисления проникли во все отрасли науки, техники и экономики, а равно в математику, и стихли семантические войны на тему того, правомерно ли называть наукой «компьютерную науку» или какую-то другую «науку об искусственном» (Simon, 1996). То, что эта область знаний была признана наукой, – заслуга первопроходцев XX века, разработавших первые курсы и учебные программы для колледжей и университетов, часто несмотря на сопротивление со стороны математических и инженерных факультетов, от которых отпочковались. Мы не можем рассказать здесь их историю, но будет справедливо отметить, что это изложение становления информатики могло бы выглядеть совершенно иначе, если бы основоположники системы образования организовали предмет по-другому.

Логика таилась в ветвях генеалогического древа еще со времен античности, вступая лишь в неловкие эпизодические контакты с вычислением. Так продолжалось до тех пор, пока в середине XIX века она не стала инструментом метаматематики. Алгоритмы превратились из абстрактного в конкретное в первые десятилетия XX века, когда были включены в программу метаматематики, имеющую целью установить, какая математика может быть признана истинной. А научные и коммерческие расчеты, долгое время бывшие движущей силой ряда последовательных улучшений механического вычислительного устройства (калькулятора), получили мощный толчок со стороны физики и техники во время Второй мировой войны.

На протяжении всего пути в серьезной работе нет-нет да и мелькала нескромная идея: а не может ли человек вдохнуть жизнь в создаваемые им машины? По мере становления информатики это мифическое видение закручивалось тугим водоворотом вокруг математики вычислений, предвещая безудержность технологической сингулярности. Качество машинного зрения, синтетической речи и ловкости роботов постоянно улучшается, порождая дебаты о том, какие последствия это сулит отдельным людям и обществу в целом.

Честно говоря, дата рождения информатики выбрана произвольно. Мы относим к доисторическим временам такие работы, как «Арифметика» Диофанта, написанная в III веке (Diophantus, 1910) и «Алгебра» аль-Хорезми, датируемая IX веком (al Khwarizmi, 1915), какими бы выдающимися интеллектуальными качествами они ни обладали. (Диофант мелькнет в главе 5 этой книги, поскольку изучал задачи теории чисел, которые после обобщения оказываются рекурсивно неразрешимыми.)

Но начать нужно с Аристотеля, заложившего понятие о высказываниях, которые могут включать переменные, представляющие свойства (глава 1). Такое высказывание может быть истинным при любых значениях переменных, может быть истинным только иногда или вообще никогда. Этот логический анализ имеет прямое отношение к информатике. Что делает цифровой компьютер «универсальным»? Тот факт, что одни и те же двоичные логические элементы могут означать разные вещи в разные моменты времени. Регистр памяти может содержать время суток, когда используется в одной программе, или адрес при использовании в другой программе. Аристотелю мы обязаны пониманием того, что одни и те же правила логики могут применяться к разным явлениям, – того, что логика дает общую основу для рассуждений.

В начале XVII века Кеплер сформулировал математические законы движения планет, Декарт свел геометрию к алгебре, а Паскаль математически охарактеризовал жидкости. Поскольку наблюдаемые физические явления описывались математическими формулами, известные с античных времен задачи вычисления площадей и объемов криволинейных фигур и тел приобрели важное практическое значение – тем более важное, что с развитием оптики измерения небесных тел становились все более точными. Развитие математики непрерывных величин заложило основу для открытия исчисления бесконечно малых, совершенного почти одновременного Исааком Ньютоном в Англии и Готфридом Лейбницем на Континенте.

Лейбниц был опытным вычислителем. Паскаль изобрел механическую суммирующую машину, наблюдая за работой отца – сборщика налогов. Лейбниц усовершенствовал это устройство, наделив его возможностью выполнять умножение и деление, и тем самым построил одну из первых вычислительных машин с вложенными циклами (см., однако, стр. 95). Лейбниц осознал преимущества двоичной системы счисления и написал об этом задолго до того, как ее полезность оценили другие; даже пионер компьютеростроения Говард Хатауэй Эйкен, который жил и работал на 250 лет позже, с большой неохотой отказался от десятичной арифметики. Но самое главное, что помимо открытия анализа бесконечно малых Лейбниц придумал исчисление идей. В главе 2 описывается лишь одно из многих утопических предложений по

рационализации дел человеческих, и Лейбниц недалеко продвинулся в осуществлении своего плана. Зная, что в XIII веке Раймонд Луллий построил машину для выполнения некоторых силлогистических выводов, Лейбниц замечает, что когда его исчисление примет законченную форму, человечество «будет иметь инструмент, который сослужит возвышенному разуму службу не хуже той, что Телескоп сослужил совершенному зрению». Логические атомы, которыми оперирует его исчисление, вновь возникают двумя столетиями позже в работе Джорджа Буля (см. ниже) и становятся базовыми фактами в таких языках логического программирования, как Prolog и Datalog.

Конструкция Аналитической машины Чарльза Бэббиджа (глава 3) могла бы ознаменовать наступление века компьютеров, поскольку устройство должно было быть программируемым и адаптируемым. Но Бэббидж не смог ее построить. У него постоянно не хватало денег, несмотря на заявления о том, что машина почти готова, – в 1835 году Бэббидж (Babbage 1989, стр. 245) писал, что «самые серьезные трудности уже преодолены и планы будут исполнены в течение нескольких месяцев», – и ее важность для национальной обороны: «управление с помощью Аналитической машины всеми астрономическими вопросами, от которых так сильно зависит флот, едва ли оставит Ее Величество равнодушной к предмету», писал он принцу Альберту (Babbage, 1843). Тем не менее ученица Бэббиджа Ада Лавлейс, рассуждая об этой так и не заработавшей машине, сумела предвосхитить самые разные концепции современного программирования.

У Джорджа Буля (George Boole, 1854, здесь глава 4) была иная повестка. Он был на 25 лет младше Бэббиджа и не получил университетского образования, но их пути, по крайней мере однажды, пересеклись. Однако у Буля была своя идея: актуализировать Аристотеля, выразить правила человеческого мышления в математической форме. Его работа по логике была слишком новаторской и выбивалась из общего русла. Ее влияние на вычисления было ограниченным – до 1930-х годов, когда Клод Шеннон положил ее в основу проектирования цифровых схем.

К началу XX века логики использовали методы самой математики для математизации идеи доказательства и стремились довести до логического завершения план Лейбница. Величайшая проблема Гильберта, проблема разрешения (Entscheidungsproblem) – определить, возможно ли доказать формализованные математические утверждения, – была строго поставлена лишь спустя несколько лет после его знаменитого обращения к Международному конгрессу математиков 1900 года (мы включили отрывки из него в главу 5). Но «Математические проблемы», с одной стороны, предвосхищают механизированную логику в своем обращении к финитным методам, а с другой – разделяют оптимизм Лейбница в том смысле, что если математики всего мира будут трудиться достаточно усердно, то они приведут свой дом в порядок.

Этого не случилось, по крайней мере не так, как представлял себе Гильберт. Сначала Гёдель, потом Чёрч, а потом еще и Тьюринг (Turing 1936, здесь глава 6) открыли мир метаматематики, который даже вообразить себе было невозможно до XX века. И хотя каждый из них оказал значительное влияние на информатику, вклад Тьюринга оказался наиболее важен, потому что (а) была

убедительно формализована идея вычислительной машины, а стало быть, и идея о том, что можно доказывать утверждения о ней; (б) был убедительно сформулирован принцип вычислительной универсальности и показано, что его можно воплотить в жизнь с помощью устройств, состоящих из простых частей; (в) сочетание (а) и (б) с неопровержимой логикой доказывало, что у вычислимости есть пределы.

Отрицательное решение гильбертовой Entscheidungsproblem стало кульминацией его статьи. В середине доказательства был использован важный технический прием – устройство хранения данных, в котором автомат мог хранить программы для других автоматов. Для завершения доказательства Тьюринг прибег к диагональному методу, предложенному в 1891 году Кантором – для совершенно другой цели, доказательства несчетности множества действительных чисел (Cantor 1996). Спустя десять лет Эккерт и Мочли воспользовались аналогичной идеей хранимой программы при модификации своего компьютера ENIAC, хотя сделали это из чисто практических соображений, а не под влиянием теоретической работы Тьюринга – хранение программы в памяти на электровакуумных лампах ускоряло работу машины и упрощало изменение программы (Priestley 2011, стр. 125). Когда Джон фон Нейман в 1945 году вошел вместе с Эккертом и Мочли в группу по проектированию машины EDVAC, пришедшей на смену ENIAC, он взял на себя труд по составлению пояснительной записки (глава 10), и с тех пор конструкция машины с хранимой программой известна (не вполне заслуженно) под названием «архитектуры фон Неймана». Та же идея примерно в то же время была использована в Манчестерской малой экспериментальной машине (Baby), а чуть позже в проекте ACE Тьюринга. В своем предложении Тьюринг (Turing 1945, стр. 3) цитирует отчет об EDVAC, но идея хранимой программы, по-видимому, из тех, что почти одновременно рождаются в головах разных людей, когда для этого приспело время.

Но вернемся в 1930-е годы. Когда Тьюринг переезжал в Принстон, чтобы продолжить свои исследования по логике, прикладной математик Говард Хатауэй Эйкен (Aiken et al., 1964, здесь глава 7) в Гарварде работал над старой проблемой численных расчетов, в которой не было ощутимого продвижения со времен Лейбница. Эйкен спроектировал громоздкий электромеханический компьютер Mark I для печати таблиц математических функций – некоторые из них, например функции Бесселя, тянулись из математического анализа в традициях XVIII века, тогда как другие, например баллистические траектории, требовались для нужд современного вооружения. Его проект закончился ссорой с компанией IBM, которая финансировала создание машины; вопрос стоял почти так же, как в случае с Бэббиджем: «Принадлежит ли честь и слава конструктору компьютера или людям, которые его построили и запустили»?

Эйкен был не единственным, кто размышлял об автоматических вычислениях в конце 1930-х годов. Конрад Цузе работал над собственными электромеханическими калькуляторами в Берлине, а Джон Винсент Атанасов разрабатывал электронную машину в Айове. Все они трудились независимо, хотя Эккерт и Мочли, работавшие по контракту с армией США в Пенсильванском университете, конечно же, знали о работе Атанасова, что впоследствии привело к ожесточенному патентному спору.

В то время как компьютеры еще только вылуплялись из яйца в нескольких разных местах, работы по телефонии повсюду шли полным ходом. Существовало несколько способов соединить коммутаторы проводами, получив один и тот же функциональный результат, и изворотливые инженеры освоили искусство минимизации необходимого оборудования. Начав работать над этими проблемами в бытность свою аспирантом МТИ, Клод Шеннон (Claude Shannon 1938, здесь глава 8) понял, что законы мышления Буля, о которых он узнал на курсе философии, являются также законами построения электрических схем. Если перевести схему на язык булевой логики, то полученную логическую формулу можно будет упростить, а затем вернуться к языку схем, получив более экономичную конструкцию. Эти методы оказались необычайно важны для проектирования цифровых компьютеров, они используются и по сей день.

У двоичной системы счисления много преимуществ с точки зрения электротехники. Мало того что булеву логику можно использовать для упрощения сложных выражений, так она еще упрощает восстановление истинного значения ослабленного сигнала в случае, когда возможных значений всего два: один уровень напряжения представляет 0, а другой 1. Когда в 1940-х годах двоичная система прочно заняла свое место, механизация логики ускорилась. Группа Эккерта, Мочли и фон Неймана воспользовалась двоичным представлением при проектировании компьютера EDVAC (von Neumann 1993, здесь глава 10), именно ей принадлежит честь раннего анализа алгоритмов двоичной арифметики. Шеннон опубликовал вторую основополагающую работу, увязав технику связи с двоичным представлением данных и попутно определив «бит» (Shannon 1948, здесь глава 12). А Хэмминг (Hamming 1950, здесь глава 13) предложил общий метод включения дополнительных битов в двоичные данные, так чтобы ошибки в процессе передачи данных можно было обнаруживать и при определенных условиях исправлять.

Все эти несомненно важные разработки происходили постепенно. У архитектуры с хранимой программой было много прародителей, и Шеннон отдавал должное Найквисту и другим инженерам-связистам своего времени. Работа нейрофизиолога Уоррен Мак-Каллока и логика-самоучки Уолтера Питтса (McCulloch and Pitts 1943, здесь глава 9) совершенно иного рода. Ничего подобного ранее не публиковалось. Литература Возрождения уподобляла человеческое тело набору рычагов, а в XIX веке эту аналогию расширили, связав потребление телом энергии с паровой машиной. Идея о том, что мышление есть форма вычисления, появилась еще до Лейбница и восходит по крайней мере к Томасу Гоббсу (Hobbes 1655, стр. 2): «под “рассуждением” я понимаю вычисление» (*per ratiocinationem autem intelligo computationem*). Но объяснение самого мозга как особого вида механизма было чем-то совершенно новым, а еще более смело выглядела попытка связать возбуждение нейронов по принципу «все или ничего» с коммутацией электрических схем и тем самым развить логическое исчисление Уайтхеда и Рассела (Whitehead and Russell 1910). Мак-Каллок и Питтс не считали эту аналогию просто игрой ума, они заявили на весь мир, что отныне тайны человеческого разума разрешены – осталось лишь несколько деталей, которые будут уточнены позже. Их работа стала единственной, цитируемой в первой редакции отчета фон Неймана по

конструированию компьютера (глава 10). Эволюционируя, нервные сети превратились в важную модель вычислений, хотя большая часть деталей работы Мак-Каллока и Питтса заменена другими идеями. Принципиально важным переосмыслением нейронной модели с приближением ее к реальности стала статья Фрэнка Розенблатта (Rosenblatt 1958a, здесь глава 18) о перцептронах, хотя нервные сети и по сей день занимают свое место в теоретической информатике, далеко уйдя от своих нейроанатомических истоков.

Двойственная идея о том, что машина может действовать как человек, имеет глубокие корни в мифологии. Гомер описывал механических слуг божественного кузнеца Гефеста:

> Сильно хромая, шатаясь на слабых ногах. Отодвинул
> В сторону, прочь от горнила, меха ...
> ... и с толстою палкой, хромая,
> Двинулся к двери. Навстречу ему золотые служанки
> Вмиг подбежали, подобные девам живым, у которых
> Разум в груди заключен, и голос, и сила, – которых
> Самым различным трудам обучили бессмертные боги.
> Под руки взяли владыку служанки...
>
> *Гомер*, Илиада, песнь 18

Уже в VIII веке до н. э. в этом отрывке описано несколько разделов искусственного интеллекта: общий интеллект, обучение, речь, подвижность. Есть мнение, что Буль обсуждал «думающую машину» Бэббиджа в ходе их единственной известной встречи в 1862 году, хотя леди Лавлейс предостерегала его от надежд на то, что Аналитическая машина будет способна на оригинальные мысли. Но кошку уже выпустили из мешка; вскоре после того Сэмюель Батлер в статье «Дарвин среди машин» предвидел, что машины могут эволюционировать и стать разумными (Butler 1863, оригинальная публикация анонимна).

К концу 1940-х годов Тьюринг уже смог объединить все, что ему было известно о компьютерах (не только теоретически – он конструировал и строил вычислительные машины), с дошедшими из глубины веков рассуждениями о думающих машинах и современной ему британской аналитической философией. В результате на свет появилась статья «Вычислительные машины и разум» (Turing, 1950, здесь глава 14), в которой Тьюринг представил себе – в качестве умозрительной альтернативы метафизическому думающему компьютеру – машину, которая сможет успешно обмануть исследователя, заставив его поверить, что он ведет диалог не с машиной, а с человеком. Бесстрастный разбор контраргументов (в том числе высказанных леди Лавлейс) дал начало оживленным спорам и ответным статьям.

В 1964 году Джозеф Вейценбаум (Weizenbaum 1966, здесь глава 27) написал простенькую программу, которая поддерживала бессодержательный диалог с человеком, просто подставляя употребленные человеком слова в ответы компьютера на синтаксически правильные позиции. Вейценбаум рассматривал эту программу, которую назвал Элиза, как техническую демонстрацию обработки естественного языка, пусть и ограниченную. Она изначально не

была предназначена для ведения осмысленного диалога, но неуемное желание людей общаться с ней так, как они общались бы с людьми, породило важные этические вопросы о взаимодействии человека и машины. Норберт Винер (Wiener 1960, здесь глава 19) уже призывал профессиональных компьютерщиков задуматься об этических последствиях своей работы, когда размышлял о последствиях автоматизации и умениях игровых компьютеров – иначе говоря, о последствиях бездумного препоручения компьютерам решений, которые следовало бы оставить за человеком.

1950-е годы стали чем-то вроде кембрийской эпохи в конструировании компьютеров. Подробная конструкторская документация по EDVAC открыла дорогу всевозможным вариациям и улучшениям, а также экспериментам с новыми элементами памяти и переключателями – как в стенах академических учреждений, так и в коммерческом секторе. Изобретение микрокода Морисом Уилксом в 1952 году (Wilkes 1981, здесь глава 15) включено в качестве примера появления в простой форме идеи, важной для решения конкретной насущной задачи: он просто устал перепаивать компьютерные схемы всякий раз, как в систему команд вносились улучшения. Фантастическое увеличение количества логических компонентов, ставшее возможным в результате перехода на транзисторы, а позже на интегральные схемы, привело в действие закон безумного экспоненциального роста, названный в честь Гордона Мура (Moore 1965, здесь глава 25) и опубликованный в журнальной статье с мультяшным изображением домашнего компьютера – задолго до того, как кто-то мог представить себе, для чего такое устройство могло бы понадобиться.

Грейс Хоппер осознала – еще до того, как это стало очевидно прочим, – что оборудование скоро станет самой дешевой частью вычислительной системы, потому что компьютер покупается один раз, а инвестиции в новое математическое обеспечение могут продолжаться бесконечно (Hopper, 1952, здесь глава 16). Кроме того, она поняла, что языки высокого уровня – не просто удобство, а необходимость, т. к. никакой владелец кодовой базы не может позволить себе переписывать ее с нуля всякий раз при покупке нового компьютера.

Языки Fortran, Algol и Cobol появились в 1950-е годы благодаря Хоппер, которой была ясна важность применения компьютеров в коммерческом секторе. Мы не можем здесь воздать должное ни одному из этих языков программирования и их создателям, за исключением Джона Маккарти, который рискнул адаптировать лямбда-исчисление Алонзо Чёрча, разработанное с целью отправить на покой Entscheidungsproblem, и превратил его в язык функционального программирования (McCarthy 1960, здесь глава 21). По ходу дела он связал логическую традицию непосредственно с искусством программирования: если трактовка вычисления как манипулирования символами – ключ к пониманию пределов вычислимости, то почему бы не сделать манипулирование символами примитивом практического языка программирования? Маккарти также уверенно поставил во главу компьютерного программированиия рекурсивный стиль определения функции, который прочно занял место в метаматематике начала XX века и был систематизирован Розой Петер (Péter 1951).

Влияние Алонзо Чёрча невозможно переоценить. К числу его докторантов в Принстонском университете относятся Тьюринг, Майкл Рабин и Дана Скотт. А Маккарти, который также писал докторскую диссертацию в Принстоне во времена Чёрча, заложил основы искусственного интеллекта, первой системы с разделением времени (Corbató et al. 1962, здесь глава 23), а также символического и функционального программирования.

Конструирование компьютерных систем не только для вычислений, но и в качестве устройств, помогающих человеку в процессах мышления и запоминания, можно возвести к публикации в популярном журнале статьи Ванневара Буша (Bush 1945a, здесь глава 11) «Как мы можем мыслить». Он пытался вообразить, как человеческому мышлению могла бы в будущем помочь технология, но не имел ясного представления о том, что помочь может именно компьютер. В следующем десятилетии начали появляться огромные, громоздкие компьютеры, и провидцы всех мастей стали придумывать, как люди когда-нибудь смогут работать с ними более удобно. Дж. К. Р. Ликлайдер (Licklider 1960, здесь глава 20) представлял себе кооперацию людей и компьютеров, а Дуг Энгельбарт (Engelbart 1962, здесь глава 22) работал над тем, чтобы воплотить эту идею в жизнь. Айвен Сазерленд (Sutherland 1963, здесь глава 24) открыл область машинной графики в своей докторской диссертации в МТИ, в стенах которого Буш грезил о мыслительном помощнике почти двадцатью годами ранее.

Изучение формально-математических абстрактных методов описания и анализа вычислений развивалось во многих направлениях. На протяжении столетий было создано много ручных алгоритмов – одни хуже, другие лучше, – а описания механических программируемых калькуляторов Лавлейс и Эйкена предусматривают некоторые варианты программирования для повышения производительности или точности. Алгоритмы на графах вошли в состав исследования операций в годы Второй мировой войны и выделились в важный раздел информатики. Крускал (Kruskal 1956, здесь глава 17) предложил алгоритм построения минимального остовного дерева, который теперь изучают практически все студенты компьютерных специальностей, а Эдмондс (Edmonds 1965) явно говорил об эффективности алгоритмов при обсуждении максимального паросочетания.

Область изучения алгоритмов и их эффективности естественным образом расширялась, поскольку компьютеры нуждались в точной постановке задачи, а программирование ставило новые вопросы. Если говорить о положительной стороне, то Фолькер Штрассен открыл совершенно новые и неожиданные алгоритмы решения старых задач умножения и обращения матриц (Strassen 1969, здесь глава 30), а Эдсгер Дейкстра (Dijkstra 1965, здесь глава 26) математически строго решил сложную задачу управления конкурентностью. С другой стороны, Стивен Кук (Cook 1971b, здесь глава 34) модифицировал данное Тьюрингом доказательство возможности описать программу логическими формулами и применил его к конкретному случаю классов $\mathcal{NP}$ и логики высказываний, поставив тем самым до сих пор не разрешенную задачу: верно ли, что $\mathcal{P} = \mathcal{NP}$? А Ричард Карп (Karp 1972, здесь глава 36) показал, что самые разные известные задачи, характеризуемые комбинаторным взрывом, являются по существу вариациями одной и той же проблемы. Кнут (Knuth 1976,

здесь глава 43) предложил нотацию, которая теперь применяется практически повсеместно для сравнения вычислительной сложности алгоритмов, а также – в не менее восхитительной статье (Knuth 1974b) – предложил терминологию $\mathcal{P}$ и $\mathcal{NP}$, с которой научное сообщество согласилось.

Вместе с абстрагированием алгоритмов росло стремление рассматривать сами программы более формально и абстрактно, даже если это вело к отказу от части выразительной способности языков программирования. Так, Дейкстра (Dijkstra 1968a, здесь глава 29) предложил полностью избавиться от переходов, но это радикальное предложение не было принято; Хоар (Hoare 1969, здесь глава 31) предложил рассматривать программы как логические формулы, которые можно было подвергнуть формальной верификации; а ДеМилло с соавторами (DeMillo et al. 1979, здесь глава 44) резко возражал против плана верификации. Лисков и Зиллес (Liskov and Zilles 1974, здесь глава 39) предложили применять те же стандарты абстрагирования к данным, и это предложение было воспринято положительно, что впоследствии привело к объектно-ориентированному программированию.

Линия развития ведет от самой первой операционной системы с разделением времени, спроектированной Корбато с сотрудниками (Corbató et al. 1962, здесь глава 23), к системе мультипрограммирования «THE», разработанной Дейкстрой (Dijkstra 1968b, здесь глава 28), и системе Unix (Ritchie and Thompson 1974, здесь глава 37), которая ныне существует во многих вариантах.

Такие большие программные системы становилось все труднее писать и поддерживать в работоспособном состоянии. В 1960-е и 1970-е годы исправление ошибок в нескольких многомиллионных проектах сразу после их запуска обошлось в миллионы долларов, а некоторые пришлось вообще отправить в мусорную корзину. Программная инженерия возникла как искусство практиков, этой теме были посвящены две классические статьи: Ройса (Royce 1970, здесь глава 33) и Брукса (Brooks 1995, впервые опубликована в 1975 году, здесь глава 40). Каждый программист должен прочесть обе.

Для обработки больших наборов данных также понадобились новые методы. Кодд (Codd 1970, здесь глава 32) определил реляционную модель, концептуально элегантную, но потребовавшую немалых усилий для практической реализации. Ныне она является основой индустрии управления данными. Поиск в больших текстовых базах данных теперь считается само собой разумеющимся аспектом поиска в вебе, но на самом деле это старая задача информационного поиска. Карен Спарк Джонс (Jones 1972, здесь глава 35) открыла полезные принципы релевантности терминов, сопоставляя частоту встречаемости в документе (которая предполагает релевантность) с частотой встречаемости во всем корпусе документов (которая предполагает, что слово недостаточно специфично и потому не может являться полезным ключом индексирования).

На 1970-е годы пришелся взрывной рост сетей. Серф и Кан (Cerf and Kahn 1974, здесь глава 38) заложили основы протоколов интернета, которые мало изменились по сравнению с первоначальным описанием, а Меткалф и Боггс (Metcalfe and Boggs 1976, здесь глава 41) описали протоколы локальных сетей

Ethernet. Принятие обоих протоколов в качестве некоммерческих стандартов открыло возможность для взаимодействия любых компьютеров между собой.

В условиях вездесущей связанности потребовалось улучшить защищенность информации. Шифрование издавна применялось для секретного обмена сообщениями, но традиционные методы в интернете имели ограниченную пригодность, потому что ключ шифрования-дешифрирования приходилось передавать по тому же самому незащищенному каналу, что и сообщение. Диффи и Хеллман (Diffie and Hellman 1976a, здесь глава 42) ошеломили сообщество, предложив решение (которое, как оказалось, частично предвидел Ральф Меркл и сотрудники Центра правительственной связи, британской секретной службы). Ривест с сотрудниками (Rivest et al. 1978, здесь глава 45) разработали необходимый математический аппарат; их алгоритм широко используется и сегодня, хотя его безопасность основывается на недоказанных предположениях. Современный всплеск интереса к квантовым вычислениям в немалой степени обусловлен тем, что квантовые компьютеры можно будет использовать для вскрытия кодов RSA (Shor, 1999). Блестящая статья Шамира (Shamir 1979, здесь глава 46), последняя в этой книге, посвящена разделению секретов таким образом, что для восстановления требуется определенный уровень кооперации; эту задачу можно поставить без всякой связи с компьютерами, и для ее решения достаточно школьной математики, но смысл она имеет только в век компьютеров.

1 Первая аналитика (~350 год до н. э.)[1]

Аристотель

Некоторые идеи, встречающиеся в информатике, настолько хорошо знакомы, что даже трудно представить, что когда-то они были новыми. Такова идея логики. Методы счета, дошедшие до нас из тьмы веков, слабо повлияли на конструкцию современных компьютеров, но принципы двузначной логики лежали в основе цифровых вычислительных машин.

Компьютеры универсальны. Большинство компьютеров для игры в шахматы или для бухгалтерских расчетов – это самые обычные компьютеры, которые можно использовать как для игр, так и для деловых целей. Бит, который сегодня означает, что конь на поле f3 является белым или черным, завтра может означать, что книга имеется в наличии. Логические правила, встроенные в оборудование компьютера, можно использовать для манипулирования как той, так и другой информацией. Но абстрактные идеи вещей и свойств и логические правила рассуждения о них существовали не всегда. Это идеи Аристотеля, они стали первыми и непременными шагами в направлении вычислений, касающихся вещей и их свойств.

Аристотель (384–322 до н. э.) был великим систематизатором. Во многих работах, большая часть которых утрачена, он подвергал анализу и классификации все, что можно себе вообразить. В книге «Первая аналитика» миру была явлена первая логическая система. Ее цель – делать выводы из посылок способом, зависящим только от формы аргументации, но не от убедительности оратора или от чего-то, не упомянутого в посылках. Данное Аристотелем объяснение логической дедукции лежит в основе всей современной логики.

---

[1] Текст печатается по изданию: *Аристотель*. Аналитики / пер. Б. А. Фохта. Гос. изд-во полит. литературы, 1952.

Технический лексикон Аристотеля сильно затрудняет чтение. По счастью, архаические детали нам не важны. Он говорит об идее предиката – свойства, которым вещь может обладать или не обладать (в терминологии Аристотеля «приписывается» или «не приписывается»). Это подводит нас к современной идее о вещи как элементе множества всех вещей, обладающих некоторым свойством. Аристотелеву идею «принадлежности» можно интерпретировать как отношение между множеством и подмножеством. Например, сказать, что свойство A не принадлежит ни одной из вещей B, – все равно, что сказать, что ни один элемент множества B не является элементом A, т. е. что B не пересекается с A (или что A и B дизъюнктны). Таким образом, пример «если A приписывается всем B, а B – всем C, то по необходимости A приписывается всем C» – это утверждение о транзитивности отношения надмножества: если $A \supseteq B$ и $B \supseteq C$, то $A \supseteq C$. В распоряжении Аристотеля не было такой нотации, но люди, формализовавшие логику и теорию множеств, стояли на его плечах.

Убедительные математические рассуждения существовали и до Аристотеля, но именно Аристотель первым отделил форму таких рассуждений от их содержания. При этом он показал, как следует рассуждать механически – сопоставляя высказывания с общими шаблонами и делая выводы, которые следовали с железной неотвратимостью. Аристотель не проектировал и не строил логических вычислительных машин, но применяемый им лексикон наводит на мысль, что он описывает процесс вычисления. Слово, которое ниже переведено как «силлогизм», – в оригинале συλλογισμός и означает «подсчет» или «вычисление».

Аристотель также описывает метод, позволяющий показать, что претендующие на истину формы логического вывода не являются универсально верными. Для этого он использует контрпримеры. Он приглашает читателя сделать общий вывод из посылок $A \supseteq B$ и $B \cap C = \emptyset$. На самом деле из этих двух посылок нельзя вывести никакого заключения о соотношении между A и C. Ибо если считать, что A = животные, B = лошади и C = люди, то все посылки удовлетворены (все лошади – животные, но лошадь – не человек) и $A \supseteq C$ (все люди – животные). Однако если A = животные, B = люди и C = камни, то посылки тоже удовлетворены (все люди – животные, но человек – не камень), но A и C не пересекаются (ни один камень не является животным). (В современном рассуждении мы должны были бы добавить, что C не может одновременно быть подмножеством A и не пересекаться с A при условии, что C не пусто.) Этот метод и по сей день используется для опровержения предположений и доказательства независимости гипотез.

Прежде всего следует сказать о предмете исследования и о том, кем оно должно быть выполнено, именно: что исследовать должно доказательство и что это – дело доказывающей науки. Далее необходимо определить, что такое посылка, термин, силлогизм, а также какой силлогизм совершенный и какой – несовершенный; затем – что значит «это целиком содержится или не содержится в этом» и что значит «что-либо приписывается всем или ни одному».

Посылка есть высказывание, утверждающее или отрицающее что-нибудь о чем-нибудь. Высказывание же это бывает или общим, или частным, или неопределенным. Общим я называю суждение, когда (А), например, присуще всем или не присуще ни одному (Б), частным – когда (А) присуще или не присуще некоторым или присуще не всем (Б), неопределенным – когда нечто одно присуще или не присуще другому, без указания на то, присуще ли оно всему или не всему другому (как, например, суждение «противоположности изучаются одной и той же наукой» или «удовольствие не есть благо»); отличается же доказывающее суждение от диалектического, ведь доказывающее суждение есть принятие одного из членов противоречия (ибо тот, кто доказывает, не спрашивает, а утверждает), а диалектическое суждение есть вопрос относительно членов противоречия. При образовании силлогизмов это различие не имеет никакого значения как в том, так и в другом случае. Ибо как тот, кто доказывает, так и тот, кто спрашивает, одинаково строят силлогизм из положений о том, что нечто присуще или не присуще (чему-нибудь другому), так что силлогистическое суждение есть вообще утверждение или отрицание чего-нибудь о чем-нибудь по указанному выше способу; при этом доказывающим оно будет в том случае, если оно истинно и взято из предположений, выдвинутых с самого начала; диалектическим же оно является для вопрошающего как вопрос относительно членов противоречия, а для строящего умозаключения – как принятие того, что кажется, и того, что вероятно, как об этом сказано в *Топике*.

В дальнейшем изложении будет точно сказано о том, что такое суждение и чем отличаются друг от друга суждения силлогистическое, аподиктическое и диалектическое; а пока достаточно и того, что определено сейчас.

Термином я называю то, на что разлагается суждение, то, что приписывается, и то, чему приписывается, независимо от того, присоединяется или отнимается то, что выражается посредством глаголов «быть» и «не быть»; *силлогизм* же есть высказывание, в котором при утверждении чего-либо из него необходимо вытекает нечто отличное от утвержденного и именно в силу того, что это есть. Под словами же «в силу того, что это есть», я разумею, что это отличное вытекает благодаря этому, а под словами «вытекает благодаря этому» – что оно не нуждается ни в каком постороннем термине, чтобы следовать с необходимостью. Совершенным силлогизмом я называю такой, который для выявления необходимости заключения не нуждается ни в чем другом, кроме того что принято. Несовершенным я называю такой, который хотя и является необходимым благодаря положенным в основание данного силлогизма терминам, но нуждается в одном или нескольких суждениях, которых нет в посылках.

Выражения «нечто одно целиком содержится в другом» и «нечто одно приписывается всему другому» означают одно и то же. Говорим же мы «нечто одно приписывается всему другому», когда не может быть указана ни одна часть подлежащего, о которой нечто другое не высказывалось бы. И точно так же, когда мы говорим, что «ничего ничему другому не приписывается».

Всякое суждение есть или суждение о том, что присуще, или о том, что необходимо присуще, или о том, что возможно присуще; и из этих суждений, в зависимости от того, приписывается ли что-либо в них или не приписы-

вается, одни бывают утвердительными, другие – отрицательными; и далее, одни утвердительные и отрицательные бывают общими, другие – частными, третьи – неопределенными.

Суждение о присущем, если оно общеотрицательное, необходимо допускает обращение в отношении своих терминов, например: если никакое удовольствие не есть благо, то и никакое благо не есть удовольствие. Общеутвердительное же суждение тоже необходимо допускает обращение, однако не в общее, а в частное, например: если всякое удовольствие есть благо, то какое-нибудь благо есть удовольствие; из частных суждений утвердительное необходимо допускает обращение его в частное же (ибо если какое-нибудь удовольствие есть благо, то и какое-нибудь благо будет удовольствием); обращение же частноотрицательного суждения не необходимо, ибо если некоторым живым существам не присуще быть людьми, то отсюда не следует, что некоторым людям не присуще быть живыми существами.

Возьмем сперва в качестве общеотрицательного суждения АБ. Если А не присуще ни одному Б, то и Б не будет присуще ни одному А. Ибо если бы оно было присуще чему-нибудь, например В, то было бы неправильно заключить, что А не присуще ни одному Б, так как В есть также часть Б. Если же А присуще всему Б, то Б будет присуще некоторым А, ибо если бы Б не было присуще ни одному А, то и А не было бы присуще ни одному Б; но ведь было предположено, что А присуще всем Б. Точно так же обстоит дело с обращением и в том случае, если суждение частное. Ибо если А присуще некоторым Б, то и Б необходимо будет присуще некоторым А. Если же Б не было бы присуще ни одному А, то и А не было бы присуще ни одному Б. Но если А некоторым Б не присуще, то не необходимо, чтобы и Б не было присуще некоторым А, как, например, в том случае, если Б есть живое существо, а А – человек; ибо не всем живым существам присуще быть людьми, однако всем людям присуще быть живыми существами. <…>

После того как мы дали эти определения, мы укажем теперь, посредством чего, когда и каким образом строится всякий силлогизм; затем придется говорить о доказательстве. О силлогизме мы должны говорить раньше, чем о доказательстве, потому что силлогизм есть нечто более общее: ведь (всякое) доказательство есть некоторого рода силлогизм, но не всякий силлогизм – доказательство.

Итак, если три термина так относятся между собой, что последний целиком содержится в среднем, а средний целиком содержится или не содержится в первом, то необходимо, чтобы для двух крайних терминов образовался совершенный силлогизм. *Средним* термином я называю тот, который сам содержится в одном, в то время как в нем самом содержится другой, и по положению он является средним; *крайними* же я называю и тот, который содержится в другом, и тот, в котором содержится другой. В самом деле, если А приписывается всем Б, а Б – всем В, то А необходимо приписывается всем В. А как следует понимать выражение «приписывается всем», об этом было сказано выше. Точно так же если А не приписывается ни одному Б, а Б приписывается всем В, то А не будет присуще ни одному В. Если же первый термин присущ всему среднему, а средний не присущ ни одному последнему, то для крайних терминов нельзя будет построить никакой силлогизм, ведь из того,

что здесь имеется, ничего не следует с необходимостью, ибо в таком случае первый термин возможно присущ и всем и ни одному последнему, так что ни частное, ни общее заключение не вытекает здесь с необходимостью. Но так как здесь ничего с необходимостью не вытекает, то нельзя построить силлогизм. Пусть терминами для случая, когда первый термин присущ всему последнему, будут: живое существо – человек – лошадь; для случая, когда он ему вовсе не присущ: живое существо – человек – камень. Не получится также никакого силлогизма и тогда, когда ни первый не присущ среднему, ни средний не присущ ни одному последнему. Для случая, когда первый термин присущ всему последнему, терминами пусть будут: наука – линия – врачебное искусство; для случая, когда он ему вовсе не присущ: наука – линия – единица.

Итак, если термины взяты в общих посылках, то ясно, когда по этой фигуре может быть построен силлогизм и когда нет; и если силлогизм получается, то термины необходимо должны находиться друг к другу в указанном выше отношении, и если термины находятся друг к другу в таком отношении, то силлогизм получится.

Если же один из терминов взят в общей посылке, а другой – не в общей и первый является большим крайним, в утвердительной или в отрицательной посылке, второй же – в утвердительной посылке – меньшим крайним, то с необходимостью получится совершенный силлогизм. Если же термин, взятый в общей посылке, является меньшим крайним или оба термина находятся в каком-либо другом отношении, силлогизм невозможен. Большим крайним термином я называю тот, в котором содержится средний термин, меньшим же – тот, который подчинен среднему. Пусть А будет присуще всем Б, а Б – некоторым В, в таком случае если выражение «быть приписываемым всем» понимать в указанном выше смысле, то А будет необходимо присуще некоторым В. Если же А не присуще ни одному Б, а Б присуще некоторым В, то А необходимо не присуще некоторым В, ибо что значит «не быть приписываемым ни одному», – это также было определено; так что (и здесь) получится совершенный силлогизм. Точно так же обстоит дело, когда суждение БВ является неопределенным и утвердительным. Ведь силлогизм здесь будет таким же – берется ли (Б В) как неопределенное или как частное…

2 Истинный метод (1677)

Готфрид Вильгельм Лейбниц

Готфрид Вильгельм Лейбниц (1646–1716) был человеком универсальных знаний – философ с широчайшим кругом интересов, юрист и политический мыслитель, а также выдающийся и плодовитый математик. Он мог бы участвовать в конкурсе за звание первого специалиста по информатике. Паскаль участвовал бы в этом конкурсе благодаря изобретению суммирующей машины. Были и другие – как до, так и после Лейбница, но именно Лейбниц построил калькулятор с вложенными циклами, который умел умножать и делить (см. стр. 95). Еще важнее то, что он изобрел двоичную арифметику (рис. 3.1) и сконструировал двоичный калькулятор (который так никогда и не был построен).

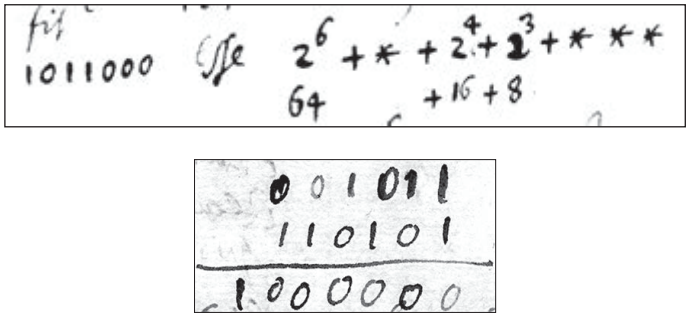


Рис. 2.1. Преобразование из двоичной системы
в десятичную и двоичное суммирование.
Из работы Лейбница «De progressione dyadica» (1679)

Лейбниц делит с Исааком Ньютоном честь открытия того, что впоследствии было названо исчислением бесконечно малых. В наши дни это исчисление так прочно ассоциируется с математикой, что мы уже забыли о вычис-

лениях, ставших побудительным мотивом для его открытия, например о том, как найти площадь фигуры посредством суммирования площадей тонких полосок. Предложенное Лейбницем обозначение *dx* для бесконечно малого приращения *x* дожило до наших дней, поскольку оно чрезвычайно упрощает запись правил, почти не выразимых в точечной нотации Ньютона. Например, тождество $\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$ очень неудобно записывать с помощью обозначения $\dot{y}$ для производной *y*, которое было предложено Ньютоном.

Лейбниц понимал, как важна хорошая система обозначений для ясного мышления. Он познакомился с сочинениями Аристотеля в возрасте 14 лет и в процессе получения образования написал диссертацию на тему использования систематической логики в юридических рассуждениях (Leibniz 1666). Он разработал формальную систему обозначений для логических рассуждений, раннюю форму математической логики (Struik, 1969, стр. 123). Его старания оформить рассуждения в виде логической системы с формальными правилами вылились в грандиозный план унификации всех человеческих знаний в систему, которая устранила бы все споры; факты, будучи один раз установлены, давали бы затем неопровержимые ответы. Тогда все рассуждения свелись бы к тривиальной подстановке, и в результате мир стал был бесспорно лучше. Знаменитый оптимизм Лейбница – его уверенность в том, что мы живем в лучшем из миров, – смешивался, таким образом, с этим ранним техноутопизмом.

И хотя математические работы принесли ему признание, его оптимизм подвергался осмеянию. В 1759 году Вольтер выставил его в карикатурном виде под личиной доктора Панглоса в «Кандиде». Сегодня его мечта о создании совершенного мира посредством логики и механизированных рассуждений кажется наивной и плохо согласующейся с теологической картиной мира, на который он хотел ее распространить. И тем не менее его идея сведения всего и вся к логике возникает вновь и вновь; наиболее известные примеры дают работы Мак-Каллока и Питтса (McCulloch and Pitts 1943, здесь стр. 120) и Буша (Bush 1945a, здесь стр. 156), а также все автоматизированные системы поддержки принятия решений.

Так как счастье заключается в удовлетворении желаний и так как непреходящая удовлетворенность зависит от уверенности в будущем – уверенности, которая основана на наших знаниях о природе Бога и души, – то отсюда вытекает, что для истинного счастья необходимо знание.

Но знание опирается на доказательство и на придумывание доказательств *некоторым методом*, который не всем известен. Ибо хотя любой человек способен судить о доказательстве (так как оно не заслуживало бы такого названия, если бы не каждый из тех, кто внимательно рассмотрел его, был им убежден и доволен), все же не каждый способен ни придумать доказательство самостоятельно, ни ясно изложить его, когда оно найдено, в силу отсутствия досуга или метода.

Истинный метод во всей своей широте представляется мне вещью, до настоящего времени неизвестной и нигде, кроме математики, не практи-

ковавшейся. Даже в самой математике он все еще далек от совершенства, что мне повезло доказать некоторым (по общему мнению, входящим ныне в число первейших математиков этого столетия) посредством вызывающих удивление доказательств. И я рассчитываю привести несколько примеров этого, которые, быть может, будут небезынтересны будущим поколениям.

И тем не менее, если метода, применяемого математиками, оказалось недостаточно для открытия всего того, что можно было ожидать от них, он, по крайней мере, смог предохранить их от ошибок, и если они не сказали всего того, что должны были бы сказать, то не сказали и ничего такого, чего говорить были не должны.

Если бы все те, кто занимается другими науками, подражали математикам хотя бы в этом пункте, то мы были бы очень счастливы и давно имели бы незыблемую метафизику, а также добрые нравы, которые зависят от нее, потому что метафизика включает в себя знание о Боге и душе, знание, которое должно направлять нашу жизнь.

Более того, мы имели бы науку о движении, которая является ключом к физике, а стало быть, и к медицине. Я верю, что мы в настоящее время сейчас пребываем в состоянии устремления к ней, и некоторые мои первые мысли, в силу их удивительной простоты, были приняты с такими рукоплесканиями наиболее образованными людьми нашего времени, что я полагаю, нам надлежит лишь провести некоторые опыты, правильно спланированные и продуманные (а не случайные, методом проб и ошибок, как часто бывает), чтобы на их основе возвести твердыню неоспоримой и доказательной физики.

Ныне же причина того, что искусство доказательства до сих пор практикуется только в математике, еще не осознана всеми, ибо если бы причина затруднений была известна, то уже давно было бы придумано лекарство. Причина же в том, что математика несет внутри себя средства собственной поверки. Ибо когда мне предъявляют ложную теорему, мне не нужно ни исследовать ее, ни даже знакомиться с доказательством, потому что я обнаружу ее ложность апостериори с помощью простого опыта, не требующего ничего, кроме бумаги и чернил, а именно вычисления, которое вскроет ошибку, сколь бы незначительной она ни была. Если бы в других предметах было так же просто рассуждать на основе опытов, то и не было бы столь различных мнений. Беда, однако, в том, что в физике опыты трудны и стоят дорого, а в метафизике вообще невозможны, если только Бог, ради нашего блага, не совершит чуда и не сделает тайные, невещественные вещи доступными для нашего познания.

Эта трудность не является непреодолимой, хотя на первый взгляд кажется нам именно таковой. Но те, кто готов склонить слух к тому, что я собираюсь сказать об этом, вскоре переменят свое мнение. Должно отметить тогда, что испытания или опыты, производимые в математике, дабы защититься от ложных рассуждений (как, например, вычеркивание девяток, вычисление Лудольфа ван Цейлена длины окружности, таблицы синусов и прочее), применяются не к самой вещи, а к символам, подставленным нами вместо вещи. [Примечание редактора: «вычеркивание девяток» – это способ проверки делимости числа на 9 путем складывания всех его цифр. Аристотель аппроксимировал число π, вычислив площадь правильного 96-угольника; Лудольф ван Цейлен (1540–1610) использовал с этой целью правильный многоугольник

с 2^{62} сторонами и вычислил π с 35 десятичными знаками, что заняло у него несколько лет (Ludolph van Ceulen 1596).] Ибо произвести вычисление над числами, например удостовериться в том, что произведение 1677 на 365 равно 612 105, было бы невозможно, если бы пришлось для этого сложить 365 кучек по 1677 камушков в каждой, а затем пересчитать их все, чтобы проверить, получилось ли указанное число. Вот почему мы с готовностью делаем это со знаками на бумаге, применяя проверку девяток или еще какой-нибудь способ. Аналогично, когда кто-то предлагает предположительно точную квадратуру круга, нам не нужно изготавливать материальный круг и обвязывать его веревкой, чтобы проверить, действительно ли длина этой веревки или отношение длины окружности к диаметру равно указанной величине; это было бы трудно, потому что если ошибка составляет тысячную (или того меньшую) часть диаметра, потребовалось бы построить большую окружность и с высокой точностью. И все же мы опровергаем неверную квадратуру кругу с помощью опыта и путем вычисления или проверки чисел. Но эта проверка производится только на бумаге и, следовательно, над знаками, представляющими вещь, а не над самой вещью.

Это соображение фундаментально в данном вопросе, и хотя многие очень умные люди, особенно в нашем столетии, заявляли, что предоставят нам доказательства в физике, метафизике, этике и даже в политике, юриспруденции и медицине, тем не менее они либо ошибались (поскольку все шаги скользкие и трудно не упасть, если не руководствоваться какими-то указаниями), либо даже если им удавалось найти доказательство, они не могли сделать свои рассуждения убедительными для всех (поскольку не существовало еще способа проверить аргументы с помощью каких-то простых испытаний, доступных каждому).

Отсюда ясно, что если бы мы могли найти знаки или символы, пригодные для выражения всех наших мыслей столь же ясно и точно, как арифметика выражает числа, а аналитическая геометрия выражает линии, то мы могли бы достигнуть во всех предметах, *поддающихся рассуждениям*, всего того, что можно сделать в арифметике и геометрии.

Ибо все вопросы, опирающиеся на рассуждение, можно было бы разрешить путем перегруппировки этих символов и с помощью некоторого вычисления, которое сделало бы изобретение красивых вещей очень простым делом. Ибо нам не пришлось бы напрягать наши мозги в такой степени, в какой мы принуждены делать это сегодня, и тем не менее испытывать уверенность в своей способности осуществить все осуществимое *в соответствии с имеющимися фактами*.

Более того, всякий должен был бы согласиться с тем, что мы обнаружили или к чему пришли путем логических умозаключений, потому что было бы легко проверить вычисление, либо повторив его, либо попытавшись произвести какие-то испытания наподобие вычеркивания девяток в арифметике. А буде кто усомнился бы в содеянном мной, я сказал бы ему: «Давайте займемся вычислениями, сэр», – а затем, взяв перо и чернила, мы вскоре уладили бы дело.

Я всегда добавляю: *при условии, что это может быть достигнуто путем рассуждения, в соответствии с имеющимися фактами*. Ибо хотя какие-то

опыты всегда необходимы, дабы служить основою для рассуждений, тем не менее, коль скоро эти опыты проделаны, мы могли бы извлечь из них все, что был бы в состоянии извлечь из них кто-то другой, и даже придумать опыты, которые нужно было бы проделать для разрешения остающихся сомнений. Это было бы достойное восхищения подспорье, даже в политике и в медицине, рассуждениям об имеющихся симптомах и свидетельствах надежным и безошибочным способом. Ибо пусть даже у нас не будет достаточных свидетельств, чтобы сформировать неопровержимое суждение, мы всегда сможем определить наиболее вероятное следствие *из имеющихся фактов*. И это все, что может сделать рассуждение.

Пойдем далее. Знаки, способные выразить все наши мысли, образуют новый язык, на котором можно будет писать и разговаривать; этот язык будет очень трудно построить, но очень легко выучить. Он скоро будет принят всеми в силу своего широкого использования и поразительной полезности и чудесно послужит цели общения между многими народами, что будет еще больше способствовать его принятию. Те, кто станет писать на этом языке, не будут допускать ошибок, при условии что будут избегать ошибок в вычислениях, варваризмов, нарушения правил и других ошибок грамматического и синтаксического характера. Более того, этот язык будет обладать замечательным свойством, а именно принуждать к молчанию невежественных. Ибо никто не будет способен говорить или писать на этом языке ни о чем, кроме того, что понимает, а если попытается сделать, то произойдет одно из двух: либо тщета оглашаемого будет очевидна всякому, либо предмет будет изучен в результате письменной или устной речи. Точно так же, как те, кто вычисляет, научаются посредством письма, а те, кто говорит, иногда встречают успех, о котором не помышляли, когда *язык прежде ума рыщет*. Особенно часто это будет случаться в нашем языке ввиду его точности. Настолько, что не будет никаких двусмысленностей и амфиболий, и все, что можно выразить на нем вразумительно, будет сказано с полной обоснованностью.

Осмелюсь сказать, что это высочайшее напряжение человеческого разума и что когда проект будет осуществлен, от самого человека будет зависеть, быть ли ему счастливым, потому что у людей появится средство, которое будет служить превознесению разума не в меньшей степени, чем телескоп служит совершенствованию зрения.

Я страстно желал бы завершить этот проект, если Бог дарует мне время. Я обязан им только самому себе, и первые мысли о нем появились у меня, когда мне было восемнадцать лет, что я засвидетельствовал несколько позже в печатном рассуждении (Leibniz 1666). И так как я уверен, что не существует изобретения, сколько-нибудь близкого к этому, я полагаю, что нет ничего, способного в такой же степени обессмертить имя изобретателя. Но у меня есть и более основательные причины думать об этом, потому что религия, которую я исповедую, убеждает меня, что любовь к Богу заключается в горячем желании распространять общее благо, а разум говорит мне, что нет ничего, что было бы лучшим общим благом для всех людей, чем то, что совершенствует разум.

3 Набросок Аналитической машины (1843)

Л. Ф. Менабреа
с замечаниями переводчика, Ады Августы, графини Лавлейс

Аналитическая машина Чарльза Бэббиджа (1791–1871) была первым устройством, которое с полным основанием можно было бы назвать компьютером, а не калькулятором, поэтому помощница Бэббиджа Ада Лавлейс была первым программистом. В этой главе будет много других «первых», в т. ч. первая компьютерная система, которую можно было бы назвать почти операционной, хотя толком она так и не заработала.

Бэббидж был выдающимся британским ученым, лукасовским профессором математики в Кембриджском университете – должность, которую занимали Исаак Ньютон и Стивен Хокинг, среди прочих. Его «Разностная машина» представляла собой систему шестерней и колес и позволяла вычислять значения многочленов, а следовательно, приближений к различным математическим функциям. В ее основе лежал принцип вычисления конечных разностей. Простейший пример дает тождество $\sum_{i=0}^{n}(2i + 1) = (n + 1)^2$, означающее, что последовательность полных квадратов можно получить, суммируя последовательные нечетные числа, начиная с 1. В середине 1830-х годов Бэббидж начал размышлять об Аналитической машине – устройстве, способном на гораздо большее. Большее даже, чем мог вообразить себе сам Бэббидж.

Ада Августа (1815–1852) была дочерью поэта-романтика лорда Байрона, который бросил семью вскоре после ее рождения и отправился в Грецию, где через восемь лет и умер. Оскорбленная мать Ады, твердо решив, что Ада не станет романтической глупышкой, дала ей лучшее математическое образование, доступное тогда женщинам. Ада стала ученицей Бэббиджа и после знакомства с его счетными машинами начала задумываться о возможностях Аналитической машины. Устройство задумывалось по образцу ткацкого

станка Жаккарда, в котором для формирования на ткани сложных рисунков использовались перфокарты. Бэббидж и Ада Августа поняли, что управляющий механизм был настолько общим, что Аналитическая машина смогла бы выполнить вычисления почти неограниченной сложности.

Смогла бы – если бы была построена. Увы, тогдашней точности механической обработки было недостаточно для построения работоспособной Аналитической машины в масштабе, описанном в этом документе. Аналитическая машина так никогда и не заработала, как планировалось. (Построение полномасштабной Разностной машины было приостановлено, когда Бэббидж обратил все свое внимание на Аналитическую машину. Но ее версия была доведена до конца с использованием методов механической обработки деталей, придуманных в процессе работы над неудавшейся Аналитической машиной.)

В 1840 году Бэббидж читал в Италии лекции о своей Аналитической машине. Итальянский инженер Луиджи Федерико Менабреа (1809–1896) слушал эти лекции и законспектировал их, а Ада (к тому времени вышедшая замуж за графа Лавлейса и потому именовавшаяся Ада Августа Лавлейс) перевела конспект Менабреа и снабдила его своими комментариями. Количество упомянутых в ней идей программирования, ныне всем знакомых, поражает: компиляция (рис. 3.1 на стр. 40 по существу является компиляцией алгебраического выражения на язык ассемблера); подсчет шагов (стр. 47); циклы, управляемые целым индексом (стр. 51); обработка исключений (стр. 40); механическое решений линейных систем (стр. 51); дихотомия данных и кода (стр. 38); нечисловые вычисления (стр. 43 и 50); недооценка трудностей программирования (стр. 41); и раз за разом повторяющиеся мысли о неоткрытой онтологии вычислений.

Ада Лавлейс умерла в возрасте 36 лет от рака матки и примененного для его лечения кровопускания. Хотя впоследствии Бэббидж воображал машины, которые были бы способны играть в настольные игры и автоматизировать другие виды деятельности, его изобретения были прочно забыты вплоть до наступления цифровой революции. Говард Эйкен знал о работах Бэббиджа и считал себя продолжателем его дела, но эта оценка, вероятно, была дана уже после того, как Эйкен приступил к проектированию своего автоматического калькулятора. Бэббидж и Лавлейс шли по нехоженой пустоши, но за недостатками применявшейся ими технологии и прошествием времени протоптанная ими тропа заросла и забылась, пока другие не прошли по ней снова.

3.1. Аналитическая машина Бэббиджа...

Машина Паскаля, которой так восхищались, ныне является просто любопытным курьезом, который хотя и демонстрирует мощь интеллекта своего изобретателя, сам по себе не особенно полезен. Ее возможности не простираются далее выполнения первых четырех арифметических действий,

а на самом деле ограничены только первыми двумя, поскольку умножение и деление являлись результатом серии сложений и вычитаний. До настоящего времени главным недостатком большинства подобных машин было то, что они требовали постоянного вмешательства человека для регулирования движения их частей, а это является источником ошибок; поэтому если они не стали общепринятым средством для объемных числовых расчетов, то это потому, что они не разрешили двойной проблемы, возникающей в этой связи: *правильность* результатов в сочетании с *экономией* времени.

Движимый подобными соображениями, м-р Бэббидж посвятил несколько лет реализации грандиозной идеи. Он задался целью не более и не менее, как сконструировать машину, способную выполнять не только арифметические вычисления, но и вычисления анализа, коль скоро их законы известны. Поначалу воображение пасует перед идеей такого масштаба, но чем спокойнее мы размышляем о ней, тем все более возможным представляется успех, и возникает чувство, что он может зависеть от открытия принципа столь общего, что, будучи применен к машине, он наделит ее возможностью механически транслировать операции, сообщенные ей в алгебраической форме. <...>

Когда для решения такой проблемы применяется анализ, обычно оказывается, что есть два класса подлежащих выполнению операций: во-первых, численное вычисление различных коэффициентов, а во-вторых, их соотнесение с соответствующими величинами. Например, если требуется вычислить произведение двух двучленов $(a + bx)(m + nx)$, то результат будет представлен выражением $am + (an + bm)x + bnx^2$, в котором мы должны сначала вычислить am, an, bm, bn, затем взять сумму $an + bm$ и, наконец, сопоставить полученные таким образом коэффициенты со степенями переменной. Чтобы воспроизвести эти операции с помощью машины, последняя должна, следовательно, располагать двумя разными видами возможностей: во-первых, уметь выполнять численные вычисления, а во-вторых, правильно распределять полученные в результате значения.

Но если бы для направления каждой из этих частичных операций было необходимо вмешательство человека, то не было бы получено никакого выигрыша в плане правильности и экономии времени; поэтому у машины должна быть дополнительная возможность самостоятельно выполнять всю последовательность операций, необходимых для решения предложенной ей задачи, после того как были предъявлены исходные числовые данные. Следовательно, поскольку начиная с того момента, как машине была сообщена природа вычислений, нуждающихся в выполнении, или задачи, подлежащей решению, она должна самостоятельно, пользуясь заложенными в нее средствами, произвести все промежуточные операции, ведущие к искомому результату, то должны быть исключены любые методы проб и ошибок, а допускаются только прямые процессы вычисления.

Это необходимое условие; ибо машина – не мыслящее существо, а лишь автомат, действующий согласно заложенным в него законам. Коль скоро это так фундаментально, одним из первых изысканий, которые надлежало предпринять автору, был поиск средств для деления одного числа на другое без использования метода угадывания, предписываемого обычными правилами арифметики. Трудности, возникшие при попытке воплотить в жизнь такое

сочетание условий, были далеко не пустяшными, но от результата зависел успех всего остального. Ввиду невозможности объяснить здесь процесс, приведший к достижению этой цели, мы должны ограничиться признанием того, что первые четыре арифметические операции, т. е. сложение, вычитание, умножение и деление, можно выполнить непосредственно путем задействования машины. Коль скоро это так, машина способна выполнить любой вид численного вычисления, т. к. все такие вычисления в конечном итоге сводятся к четырем поименованным выше операциям. Теперь, чтобы понять, как машина может исполнить свои функции в соответствии с заложенными законами, мы начнем с идеи о том, как в ней фактически представлены числа.

Рассмотрим штабель, или вертикальный столбик, составленный из бесконечного числа круглых дисков, нанизанных на общую ось, вокруг которой все они могут независимо вращаться. Если вдоль края каждого диска написать десять цифр, образующих наш цифровой алфавит, то, расположив нужные цифры на вертикальной прямой, мы сможем представить любое число. Для этого нужно лишь, чтобы первый диск представлял единицы, второй – десятки, третий – сотни и т. д. Если таким образом записать два числа на двух разных столбиках, то мы сможем предложить процедуру их арифметического комбинирования и получить результат на третьем столбике. В общем случае, если имеется ряд столбиков, составленных из дисков, которые мы обозначим V_0, V_1, V_2, V_3, V_4, и т. д., то мы можем, например, запросить деление числа, записанного на столбике V_1, на число, записанное на столбике V_4, с получением результата на столбике V_7. Чтобы выполнить эту операцию, мы должны применить к машине две разные конфигурации: первая подготавливает машину к *делению*, а вторая сообщает, над какими столбиками производить операцию и на каком столбике представлять результат. Если вслед за делением необходимо выполнить, например, сложение двух чисел, записанных на других столбиках, то обе прежние конфигурации должны быть одновременно изменены. Если, с другой стороны, требуется выполнить последовательность операций одного и того же вида, то первую конфигурацию следует оставить, а изменить только вторую. Следовательно, конфигурации различных частей машины можно разделить на два основных класса:

- первый относится к *Операциям*;
- второй относится к *Переменным*.

Последнее подразумевает, что мы сообщаем машине, к каким столбикам она должна применять операцию. Что до самих операций, то они выполняются специальным механизмом, называемым *мельницей*, который и сам состоит из нескольких столбиков, похожих на те, что служат для представления Переменных. Прежде чем комбинировать два числа, машина стирает прежнее содержимое с соответствующих столбиков, т. е. помещает *нуль* на ту вертикальную линию каждого диска, на которой записываются числа, и передает числа мельнице. После того как механизм нужным образом подготовлен к совершению требуемой операции, эта операция выполняется, и по завершении результат передается в столбик Переменных, который должен быть заранее указан. Таким образом, мельница – исполнительная часть машины, а столбики Переменных – часть, служащая для задания конфигурации

и представления результатов. Из сказанного выше понятно, что все дробные и иррациональные результаты будут представлены десятичными дробями. В предположении, что каждый столбик состоит из сорока дисков, точности будет достаточно для той степени приближения, которая обычно требуется.

Теперь надлежит задаться вопросом, как машина самостоятельно, без участия человека, организует последовательность конфигураций, необходимую для выполнения операций. Решение этой проблемы было заимствовано у станка Жаккарда, который служит для изготовления парчовых тканей и применяется следующим образом.

Обычно в ткацком станке имеются две нити: *основа*, или продольная нить, и *уток*, или поперечная нить. Уток протягивается устройством, которое называется челноком, и переплетается с основой. Если требуется получить парчовую ткань, то уток не должен пропускаться под каждой нитью основы, и последовательность пропусков определяется рисунком, который требуется воспроизвести. Изначально этот процесс был долгим и трудоемким; рабочий должен был внимательно следить за воспроизводимым рисунком и сам регулировать перемещения нитей. Поэтому описания рисунков стоили очень дорого, особенно если ткань состояла из разноцветных нитей. Чтобы упростить изготовление, Жаккард придумал, как соединять группы нитей, к которым применяются одинаковые операции, с помощью отдельной иглы, своей для каждой группы. Каждая игла завершается крючком, и все они объединены в одну пачку, обычно имеющую форму параллелепипеда с прямоугольным основанием. Крючки имеют цилиндрическое сечение и разделены небольшими промежутками. Таким образом, процесс поднятия нитей сводится к перемещению игл в требуемом порядке. Для этого берется прямоугольная картонная карта, по размеру чуть больше пачки игл. Если эту карту приложить к основанию пачки и затем сообщить карте поступательное движение, то она увлечет за собой все крючки пачки, а вместе с ними и нити, прикрепленные к каждому крючку. Но если в сплошной карте просечь отверстия, соответствующие концам встречающих ее игл, то поскольку каждая игла пройдет сквозь карту во время движения последней, то все они останутся на своих местах. Таким образом, мы видим, что нетрудно задать положения отверстий в карте так, чтобы в каждый момент времени было поднято определенное число игл и, стало быть, пучков нитей, а остальные остались там, где были. В предположении, что этот процесс повторяется по правилу, определенному заданным узором, мы заключаем, что такой узор можно перенести на ткань. Для этого нужно лишь подготовить последовательность карт, соответствующую правилу, и расположить их в нужном порядке одну за другой. Затем, заставив их проходить над многогранной балкой, которая закреплена так, чтобы поворачиваться новой гранью после каждого хода челнока, каковая грань будет тогда двигаться параллельно самой себе против пачки игл, мы сможем упорядоченно выполнять операцию поднятия нитей. Как видим, изготавливать парчовые ткани теперь можно с такой точностью и скоростью, которых раньше достичь было трудно.

Механизмы, аналогичные описанным выше, были включены и в Аналитическую машину. Она содержит два основных вида карт: во-первых, карты Операций, с помощью которых части машины располагаются так, чтобы вы-

полнить любую заданную серию операций, таких как сложение, вычитание, умножение и деление; во-вторых, карты Переменных, которые сообщают машине, на каких столбиках должны быть представлены результаты. После приведения машины в действие карты последовательно организуют различные части машины в соответствии с природой подлежащих выполнению процессов, и в то же время машина выполняет эти процессы с помощью различных частей своего механизма.

Чтобы лучше понять, как устроена работа машины, возьмем в качестве примера решение системы двух уравнений первой степени с двумя неизвестными. Рассмотрим следующие два уравнения, в которых x и y – неизвестные величины:

$$\begin{cases} mx + ny = d \\ m'x + n'y = d' \end{cases}.$$

Отсюда получаем $x = \frac{dn' - d'n}{n'm - nm'}$ и аналогичное выражение для y. Будем и дальше представлять буквами V_0, V_1, V_2 и т. д. различные столбики, содержащие числа, и предположим, что первые восемь столбиков выбраны для записи на них чисел, представленных переменными m, n, d, m', n', d', n и n', откуда следует, что $V_0 = m$, $V_1 = n$, $V_2 = d$, $V_3 = m'$, $V_4 = n'$, $V_5 = d'$, $V_6 = n$, $V_7 = n'$.

Последовательность операций, описываемых картами, и полученные результаты можно представить таблицей на рис. 3.1.

Число операций	Карты операций	Карты переменных		Ход выполнения операций
	Символы, обозначающие природу операций	Столбики, к которым применяются операции	Столбики, на которые записываются результаты операций	
1	×	$V_2 \times V_4 =$	V_8	$= dn'$
2	×	$V_5 \times V_1 =$	V_9	$= d'n$
3	×	$V_4 \times V_0 =$	V_{10}	$= n'm$
4	×	$V_1 \times V_3 =$	V_{11}	$= nm'$
5	–	$V_8 - V_9 =$	V_{12}	$= dn' - d'n$
6	–	$V_{10} - V_{11} =$	V_{13}	$= n'm - mn'$
7	÷	$\frac{V_{12}}{V_{13}} =$	V_{14}	$= x = \frac{dn' - d'n}{n'm - nm'}$

Рис. 3.1

<...> Мы можем вывести из этих объяснений следующее важное следствие, а именно: поскольку одни лишь карты сообщают природу подлежащих выполнению операций и столбики Переменных, к которым эти операции применяются, сами эти карты несут в себе всю общность анализа, по отношению к которому выступают просто в роли перевода на другой язык. Далее мы рассмотрим некоторые трудности, которые должна преодолеть машина, чтобы ее приспособленность к анализу была полной. Существуют функции, природа которых по необходимости изменяется, когда они проходят через нуль или

бесконечность, а также такие, которые не принимают никаких значений при прохождении через эти пределы. Столкнувшись с такими случаями, машина может с помощью звукового сигнала известить о том, что имеет место прохождение через нуль или бесконечность, после чего останавливается в ожидании того, что обслуживающее лицо снова приведет ее в действие, дабы она могла выполнить следующий процесс. Если этот процесс можно было предвидеть, то машина вместо подачи сигнала сконфигурирует себя так, чтобы подать новые карты, имеющие отношение к операции, которая должна следовать за прохождением через нуль или бесконечность. Эти новые карты могут следовать за первой, но могут также вступать в игру лишь при условии возникновения одной из двух описанных выше ситуаций.

Рассмотрим член вида ab^n; поскольку карты – всего лишь перевод на другой язык аналитической формулы, их количество в этом конкретном случае должно быть одинаково вне зависимости от значения n, т. е. от числа операций умножения, необходимых для возведения b в n-ю степень (в данный момент мы предполагаем, что n – целое число). Поскольку показатель степени n указывает, что b следует умножить на себя n раз, а все эти операции имеют одну и ту же природу, достаточно будет всего одной карты операции, а именно той, что заказывает умножение. <...>

Возвращаясь к нашим объяснениям Аналитической машины, мы можем заключить, что она основана на двух принципах: первый заключается в том, что любое арифметическое вычисление в конечном итоге зависит от четырех основных операций – сложения, вычитания, умножения и деления, а второй – в том, что любое аналитическое вычисление можно свести к вычислению нескольких коэффициентов ряда. Если этот последний принцип верен, то все операции анализа оказываются подвластны машине. Взглянем на это по-другому: использование карт обеспечивает такую же общность, как использование алгебраических формул, поскольку формула просто описывает природу и порядок выполнения операций, необходимых для получения некоего определенного результата, и точно так же карты просто приказывают машине выполнить те же самые операции; но чтобы механизмы были способны действовать во имя достижения какой-то цели, в каждом конкретном случае должны быть введены числовые данные. Таким образом, одна и та же последовательность карт будет пригодна для ответа на все вопросы, постановка которых не отличается ничем, кроме числовых данных. В свете этого карты – это просто перевод алгебраических формул на другой язык, или, лучше сказать, другая форма аналитической нотации.

Поскольку режим работы машины своеобразный, в каждом конкретном случае придется организовывать вычисления, сообразуясь со средствами, которыми машина располагает; иногда процесс, очень простой для калькулятора, может оказаться долгим и сложным для машины, а иногда *наоборот*.

С самой общей точки зрения, из того, что главная цель машины – вычислять по сообщенным ей правилам значения числовых коэффициентов, которые затем распределяются по столбикам, представляющим переменные, следует, что интерпретация формул и результатов выходит за рамки ее компетенции, если только сама эта интерпретация не может быть выражена с помощью символов, которыми машина оперирует. Таким образом, хотя

машина и не является рассуждающим существом, ее все же можно рассматривать как сущность, исполняющую некоторые функции, свойственные разуму. Карты несут на себе оттиск этих функций и передают различным элементам механизма, образующего машину, приказы, необходимые для выполнения ими тех или иных действий. Стоит лишь построить машину, как вся трудность будет сведена к изготовлению карт; но так как они представляют собой просто перевод алгебраических формул на другой язык, то, придумав какую-то простую систему обозначений, будет нетрудно поручить решение этой задачи ремесленнику. И значит, на долю интеллектуального труда выпадет только подготовка формул, адаптированных для вычисления машиной.

В предположении, что такую машину можно построить, имеет смысл задать вопрос: в чем будет состоять ее польза? Резюмируем: она будет иметь следующие преимущества: во-первых, высокая точность. Мы знаем, что численные расчеты зачастую являются сдерживающим фактором при решении задач, т. к. в них легко могут вкрасться ошибки, а обнаружить эти ошибки не всегда просто. Машина же, по самому принципу своего действия, не нуждается во вмешательстве человека в ход выполнения операций и потому обеспечивает полную безопасность в плане корректности. Кроме того, она предлагает средства контроля, ибо в конце каждой операции распечатывает не только результаты, но и числовые данные для нее, поэтому легко проверить, правильно ли был поставлен вопрос. Во-вторых, экономия времени: чтобы убедиться в этом, достаточно вспомнить, что для умножения двух двадцатизначных чисел потребно как минимум три минуты. Добавим еще, что когда требуется выполнить длинную цепочку одинаковых вычислений, как, например, при построении числовых таблиц, машину можно настроить так, что она будет выдавать несколько результатов одновременно, что заметно сократит общее потребное время. В-третьих, экономия умственных усилий: простое арифметическое вычисление доступно человеку, владеющему лишь скромными навыками; когда же мы переходим к более сложным вычислениям и, быть может, даже желаем использовать алгебраические формулы, необходимы более обширные познания, предполагающие определенную математическую подготовку. Машина же, благодаря своей возможности выполнять все эти чисто механические операции, экономит умственные усилия, которые можно приложить к чему-то другому с большей пользой. Таким образом, машину можно рассматривать как фабрику по обработке чисел, что делает ее полезным подспорьем в тех многочисленных науках и искусствах, которые опираются на числа. Да и кто может предвидеть все последствия такого изобретения? В самом деле, сколь много драгоценных наблюдений остаются недоступными для прогресса наук, потому что не хватает средств для вычисления результатов! А сколь сильное уныние может вызвать перспектива длительного и скучного вычисления у гения, который нуждается во времени для размышлений, а время, потраченное на приземленную рутину операций с числами, считает украденным! Однако же постигать истину он вынужден тернистым путем анализа, но пройти по нему не может, не руководствуясь числами, ибо без чисел нам не дано приподнять завесу, скрывающую тайны природы. Таким образом, идея построения механизма, способного восполнить неспособность человека к таким изысканиям, – задумка, воплощение

которой ознаменовало бы славную эпоху в истории науки. Имеются планы, касающиеся всех ее частей, в том числе всех зубчатых передач, составляющих этот огромный механизм. Изучено и его действие. Но пока что эти части не соединены между собой в чертежах и механических обозначениях. Уверенность, которую вселяет гений м-ра Бэббиджа, позволяет небезосновательно надеяться, что это предприятие увенчается успехом, и, отдавая должное могучему интеллекту, направляющему его, будем, затаив дыхание, ждать исполнения этого начинания.

3.2. Замечание А переводчика

Конкретной функцией, для табулирования интеграла которой была построена Разностная машина, является $\Delta^7 u_z = 0$. Цель, для достижения которой задумывалась и настраивалась машина, – вычисление навигационных и астрономических таблиц. Интеграл $\Delta^7 u_z = 0$ имеет вид

$$u_z = a + bx + cx^2 + dx^3 + ex^4 + fx^5 + gx^6;$$

постоянные a, b, c и т. д. представлены на семи столбиках дисков, составляющих машину. Поэтому она может точно и *с неограниченной протяженностью* табулировать все ряды, общий член которых описывается вышеприведенной формулой; она также может *приближенно* табулировать промежуточные *интервалы большей или меньшей протяженности* всех прочих рядов, допускающих табулирование методом конечных разностей.

Напротив, Аналитическая машина настраивается не просто для *табулирования* результатов вычисления какой-то одной и только одной функции, а для *вычисления и табулирования* любой функции. На самом деле машину можно описать как материальное выражение любой неопределенной функции произвольной сложности и общности, например $F(x, y, z, \log x, \sin y, x^p, \ldots)$, которая, заметим, является функцией от любых других возможных функций в любом количестве.

В этом, назовем его *нейтральным*, или *нулевым*, состоянии машина готова в любой момент, с помощью карт, составляющих часть ее механизма (и применяемых по тому же принципу, что и в ткацком станке Жаккарда), принять оттиск любой специальной функции, которую мы хотим вычислить или табулировать. Эти карты содержат внутри себя (так, как объяснено в самой памятной записке) закон вычисления конкретной функции, и они понуждают механизм действовать в соответствующем порядке. Рассмотрим простейший пример; пусть $F(x, y, z, \ldots)$ – конкретная функция $\Delta^n u_z = 0$, которую Разностная машина табулирует для значений n, не превышающих 7. В этом случае карты понудят механизм выполнить последовательность операций, которая будет табулировать

$$u_z = a + bx + cx^2 + \ldots + mx^{n-1},$$

где число n произвольно.

Однако эти карты не имеют ничего общего с заданием конкретных *числовых* данных. Они лишь определяют, какие *операции* следует выполнить, а эти операции, разумеется, можно применить к бесконечному разнообразию конкретных числовых значений и не выводить никаких числовых результатов, пока исходные числовые данные не были поданы на соответствующие части механизма. В примере выше первым важным шагом на пути получения арифметического результата была бы подстановка конкретных значений *n* и других примитивных величин, которые мы вводим в функцию. <…>

Исполнительный механизм можно привести в действие даже без объекта применения (конечно, в этом случае не будет получено никакого *результата*). Он также может применяться к другим объектам, а не только к *числам*, лишь бы между этими объектами существовали фундаментальные соотношения, которые можно выразить с помощью соотношений между абстрактными научными операциями, а сами объекты допускали адаптацию к нотации операций и действию механизма машины. Например, в предположении, что фундаментальные соотношения между звуками разной высоты в науке о гармонии и музыкальной композиции поддаются такому выражению и адаптации, машина могла бы создавать затейливые и научно обоснованные музыкальные произведения любой степени сложности и длительности. <…>

Отличительной особенностью Аналитической машины, благодаря которой стало возможно наделить механизм такими развитыми способностями, чтобы сделать машину незаменимым помощником в решении задач общей алгебры, является внедрение в нее принципа, изобретенного Жаккардом для формирования, посредством перфорированных карт, сложнейших рисунков в процессе изготовления парчовых тканей. Именно здесь проходит различие между двумя машинами. В Разностной машине ничего подобного нет. Образно говоря, Аналитическая машина *выплетает алгебраические узоры*, как ткацкий станок Жаккарда выплетает цветы и листья. В этом, как нам кажется, заключена гораздо большая оригинальность, чем та, на которую могла бы притязать Разностная машина. Мы не собираемся отказывать последней во всех таких притязаниях. Мы полагаем, что это единственная когда-либо предпринятая попытка построить счетную машину, *основанную на принципе разностей последовательных порядков*, которая могла бы *печатать полученные ей результаты*. Мы полагаем также, что эта машина превосходит своих предшественников и в разнообразии доступных ей вычислений, и в надежности и точности их выполнения, и в отсутствии необходимости вмешательства в ее работу со стороны человека *в процессе выполнения вычислений*. Однако по самой своей природе она ограничена одними лишь арифметическими операциями и является далеко не первой и не единственной схемой конструирования *арифметических* счетных машин, которые можно было бы назвать более или менее успешными.

Но в тот момент, когда пришла идея применить карты, произошел выход за пределы *арифметики*; Аналитическая машина не стоит в одном ряду с простыми «счетными машинами». У нее есть свое собственное место, и размышления, на которые она наводит, весьма интересны. Наделив механизм способностью объединять *общие* символы в последовательности неограни-

ченного разнообразия и протяженности, мы создали связующее звено между предметными операциями и абстрактными умственными процессами, относящимися к *наиболее абстрактной* ветви математической науки. Новый, обширный и мощный язык разрабатывается для будущего использования в анализе; благодаря его могуществу, возможно, появятся более быстрые и точные практические приложения – во благо человечества, которое до сих пор таких средств было лишено. Таким образом, не только между умственным и вещественным, но также между теоретическим и практическим в мире математики устанавливается более прочная и полезная связь. Нам неизвестно ни о чем существующем в природе, что бы так же хорошо, как предложенная выше Аналитическая машина, в практическом плане отвечало идее мыслящей или рассуждающей машины. <…>

Склонные к строго утилитарному взгляду на мир, возможно, полагают, что уникальные возможности Аналитической машины имеют отношение только к абстрактной и умозрительной науке, но не к повседневным и обыденным человеческим потребностям. Эти люди, скорее всего, не сочувствующие и, возможно, даже не знакомые ни с какими отделами науки, которую они не находят *полезной* (в их собственном понимании этого слова), наверное, думают, что разработка этой машины теперь, когда над другой уже ведутся работы, была бы пустым и бесполезным вложением еще большего количества денег и труда, по существу сверхдолжным делом. Но даже при таком, утилитарном взгляде мы не сомневаемся, что с помощью расширенных возможностей Аналитической машины можно было бы получить весьма ценные практические результаты; на некоторые мы могли бы намекнуть уже теперь, если бы у нас было больше места, но есть и другие, которые сейчас еще невозможно предвидеть, но которые обязательно были бы выдвинуты на первый план в силу каждодневно растущих потребностей науки и вследствие более тесного практического знакомства с возможностями машины, коль скоро она была бы создана. <…> А. А. Л.

3.3. Замечание B переводчика

Та часть Аналитической машины, о которой здесь идет речь, называется хранилищем. Она содержит неопределенное число столбиков дисков, описанных М. Менабреа. Читатель может представить себе стопку большого числа шашек большого диаметра, положенных друг на друга перпендикулярно земле; на ребре каждой шашки через равные интервалы нанесены цифры от 0 до 9. Если затем он вообразит себе, что шашки не касаются друг друга, а с небольшими промежутками насажены на общую вертикальную ось, вокруг которой могут свободно вращаться, так что любую цифру, нанесенную на ребро, можно поместить перед глазами, то получит довольно точное представление об одном из таких столбиков. Самый нижний диск в каждом столбике соответствует единицам, следующий – десяткам, следующий – сотням и т. д. Таким образом, если бы мы захотели написать на столбике машины число 1345, то это выглядело бы так:

1
3
4
5

В Разностной машине есть семь таких столбиков, расположенных в ряд, а за ними располагается исполнительный механизм; в целом машина имеет форму четырехугольной призмы (более-менее близкой к кубу). Результаты всегда появляются на грани машины, противоположной той, со стороны которой расположился наблюдатель. В Аналитической машине таких столбиков было бы гораздо больше, наверное, не меньше двухсот. Точная форма и пространственная организация всего механизма еще окончательно не определены.

Столбики дисков можно изобразить на бумаге, как показано на рис. 3.2.

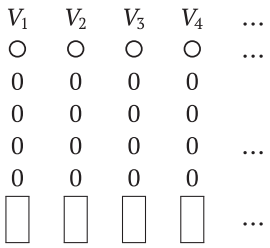


Рис. 3.2

Буквами *V* обозначаются столбики как на письме, так и в устной речи, и все они последовательно пронумерованы. Предпочтение отдано букве *V*, а не какой-нибудь другой, потому что столбики соответствуют (как сказано выше в памятной записке) *Переменным* (**V**ariable), и иногда они называются *столбиками Переменных*, или *Переменными столбиками*. Такое название выбрано, потому что значениям на столбиках предопределено *изменяться*. Однако не следует впадать в естественное заблуждение, думая, будто столбики предназначены только для приема значений *переменных* в аналитической формуле, но не *постоянных*. Столбики называются Переменными по причинам, совершенно не связанным с различием между постоянными и переменными *в анализе*. Чтобы предотвратить саму возможность путаницы, мы в переводе и в замечаниях записываем слово Переменная с большой буквы, когда оно обозначает *столбик машины*, и переменная с маленькой буквы, когда имеется в виду *переменная в формуле*. Аналогично *карты-Переменные* обозначают карты, принадлежащие столбику машины.

Вернемся к объяснению рисунка: каждый кружочек в верхнем ряду может представлять знак + или – в зависимости от того, является ли число в расположенном ниже столбике положительным или отрицательным. В этих кружочках можно было бы также размещать другие чисто символические результаты алгебраических процессов. В замечании A была затронута практическая возможность получения символических результатов с такой же легкостью,

как числовых. Каждый нуль под кружочками символов представляет диск, на котором предположительно выставлена цифра 0. На рисунке изображено только четыре ряда нулей, но их может быть и тридцать, и сорок, и вообще любое потребное количество. Поскольку каждый диск может представлять любую цифру, а каждый кружочек – любой знак, диски в каждом столбике можно расположить так, что они будут выражать любое положительное или отрицательное число в пределах возможностей механизма, каковые возможности зависят от *вертикальной* протяженности механизма, т. е. от количества дисков в одном столбике.

Каждый прямоугольник под рядами нулей предназначен для вписывания произвольного символа или комбинации символов, значением которого (или которой) является число, представленное в расположенном над ним столбике. Например, представим три величины a, n, x в предположении, что $a = 5$, $n = 7$, $x = 98$. Получится картина, изображенная на рис. 3.3.

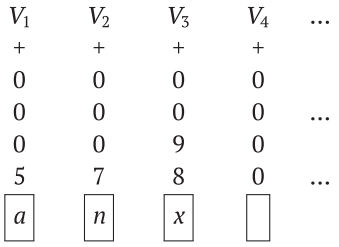


Рис. 3.3

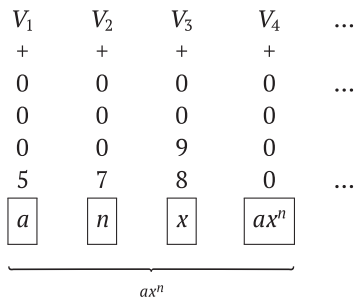


Рис. 3.4

Эти символы можно далее комбинировать различными способами, образуя из них любую нужную функцию или несколько функций. Эту функцию (или функции) можно затем записать под фигурными скобками, группирующими величины (и только их), которые подаются на вход функции, написанной под ними. Решив, для какой функции мы хотим вычислять числовое значение, мы должны также отвести справа еще один столбик для приема результатов, а в прямоугольник под этим столбиком вписать саму функцию. В рассматриваемом примере можно было бы взять любую из следующих функций:

$$ax^n,\ x^{an},\ a \cdot n \cdot x.\ \frac{a}{n}x,\ a + n + x,\ \ldots$$

Сначала выберем первую. До вычисления картина должна быть такой, как показано на рис. 3.4. Получив данные, мы должны ввести в машину карты, инструктирующие ее выполнить операции, соответствующие конкретной выбранной функции. В данном случае это будут такие операции.

Во-первых, шесть умножений, необходимых для получения x^n (= 98^7 для наших данных).

Во-вторых, одно умножение для получения $a \cdot xn$ (= $5 \cdot 98^7$).

Для выполнения всего процесса необходимо семь операций умножения. Их можно представить следующим образом:

(×, ×, ×, ×, ×, ×, ×) или 7(×).

Однако на разных этапах решения задачи умножение будет применяться к парам чисел, взятых из *различных* столбцов. Иными словами, *одна и та же операция* будет выполняться для разных *субъектов операции*. И здесь мы снова видим иллюстрацию положений, высказанных в предыдущем замечании, о независимости способа задания машинных операций от данных. В определении значения ax^n *операции однородны*, но на последовательных этапах вычисления распределяются между разными *субъектами операции*. А как следует *распределить* операции для вычисления конкретной функции, определяют перфорированные карты, принадлежащие самим Переменным. *Карты-Операции* только задают общую последовательность операций. По сути дела, они последовательно переводят части механизма, образующие *мельницу*, в различные *состояния*, которые мы могли бы назвать *состоянием сложения*, *состоянием умножения* и т. д. В каждом из этих состояний механизм готов воздействовать способом, характерным именно для этого состояния, на любую пару чисел, которая попадет в сферу его действия. В каждый момент времени может иметь место только *одно* из таких рабочих состояний, а природа механизма такова, что в каждый момент времени лишь *одна пара чисел* может быть принята и подвергнута операции. Чтобы мельница получала постоянный запас нужных пар чисел одну за другой и располагала результат операции над каждой парой в правильном месте, с каждой Переменной связаны карты, принадлежащие только ей. Во-первых, имеется класс карт, назначение которых – *обеспечить* передачу числа на Переменной в мельницу, где оно будет подвергнуто операции. Эти карты можно назвать *Подающими*. Они снабжают мельницу пищей. Во-вторых, с каждой Переменной связан еще один класс карт, задача которых – дать Переменной возможность *принять* число *из* мельницы. Эти карты можно назвать *Принимающими*. Они задают местоположение результатов, временных или окончательных. Карты-Переменные вообще (включая оба вышеупомянутых класса), как нам кажется, было бы более уместно назвать Распределительными картами, т. к. с их помощью правильно *распределяются* действия операций и результаты этих действий.

Существует *две разновидности Подающих* карт-Переменных, предназначенные для двух совершенно разных вспомогательных целей, но поскольку эти детали не влияют на рассматриваемую здесь тему, мы поговорим о них в другом месте.

В рассматриваемом случае функции ax^n карты-Операции просто заказывают семь умножений, т. е. инструктируют мельницу переходить в состояние умножения семь раз подряд (без каких-либо отсылок к конкретным столбикам, над числами в которых нужно производить операции). Нужные Распределительные карты-Переменные подаются на каждом шаге умножения и задают распределения, необходимые в каждом конкретном случае (рис. 3.5).

Для	x^{an}	Операции будут такими			$34(\times)$
⋯	$a \cdot n \cdot x$	⋯	⋯	⋯	$(\times, \times)$ или $2(\times)$
⋯	$\frac{a}{n} \cdot x$	⋯	⋯	⋯	$(\div, \times)$
⋯	$a + n + x$	⋯	⋯	⋯	$(+, +)$ или $2(+)$

Рис. 3.5

Машину можно заставить вычислять все эти результаты последовательно. После вычисления ax^n под фигурной скобкой можно было бы записать функцию x^{an} вместо ax^n и начать новое вычисление (разумеется, должны быть поданы новые карты-Операции и карты-Переменные). Результаты появились бы в столбике V_5. И так далее для любого числа различных функций от величин a, n, x. Каждый *результат* мог бы оставаться сколь угодно долго на своем столбике в процессе последующих вычислений, так что после вычисления всех функций их значения были бы одновременно представлены на столбиках V_4, V_5, V_6, ..., или же каждый результат (после печати или использования еще каким-то способом) можно было бы стереть, чтобы освободить место для следующего. Во втором случае в прямоугольник под V_4 следовало бы последовательно вписывать функции ax^n, x^{an}, anx и т. д. <...>

Чем дольше мы анализируем способ, которым такая машина функционирует и получает результаты, тем лучше понимаем, как ясно она представляет в истинном свете взаимные отношения и связи между различными шагами математического анализа; как отчетливо она разделяет сущности, которые в действительности различны и независимы, и объединяет те, которые зависят друг от друга. А. А. Л.

3.4. Замечание С переводчика

Те, кто хотел бы изучить принцип работы ткацкого станка Жаккарда самым эффективным способом, т. е. путем практического наблюдения, должны лишь дойти до галереи Аделаиды или до Политехнического института. И в том, и в другом собрании научных *иллюстраций* на станке Жаккарда постоянно работает ткач, готовый дать любую информацию о конструкции и режимах работы своего инструмента. Том Кабинетной энциклопедии Ларднера, посвященной производству шелка, содержит главу о станке Жаккарда, которую также можно с пользой прочитать.

Однако, как выяснилось, способ применения карт, который до сих пор используется в ткачестве, недостаточно эффективен для всех тех упрощений, которые необходимы для достижения целей, стоящих перед Аналитической машиной. Был придуман метод *обратной подачи* карты определенными группами в соответствии с определенными правилами. Цель этого усовершенствования – обеспечить возможность использования одной карты или группы карт *много раз подряд* в процессе решения одной задачи. Дает ли эта возможность преимущество в конкретном случае, зависит от природы операций, необходимых для решения рассматриваемой задачи. Этот процесс упоминается М. Менабреа и является очень важным упрощением. Было пред-

ложено использовать это изобретение и в самом искусстве ткачества, которое, хотя и не имеет видимой связи с абстрактной наукой, оказалось весьма ценным для последней, поскольку предложило принципы, которые в своей новой и специальной области применения, кажется, привносят в устройство механизма *алгебраические* комбинации, присутствовавшие ранее в интересных узорах, образуемых *переплетением нитей*. После включения системы обратной подачи в сам станок Жаккарда регулярно повторяющихся узоров, обладающих симметрией, стало возможно ткать на сколь угодно длинном отрезке с помощью сравнительно небольшого числа карт.

Те, кто знаком с устройством ткацкого станка, поймут, что описанное усовершенствование легко реализуется на практике, нужно лишь, чтобы призма, поверх которой проходит череда карт, описывающих узор, могла при определенных обстоятельствах вращаться *назад*, а не *вперед*, и это вращение должно продолжаться до тех пор, пока конкретная карта или набор карт, которые один раз уже сделали свое дело, будучи поданы в обычном прямом порядке, не вернутся в положение, которое занимали раньше. После этого призма возобновляет вращение в *прямом* направлении и таким образом повторяет действие карты или набора карт. Очевидно, что этот процесс можно повторять произвольное число раз. А. А. Л.

<...>

3.6. Замечание E переводчика

Многие люди, не сведущие в занятиях математикой, полагают, что коль скоро задача машины – выдавать результаты в *числовой записи*, то и сама *природа выполняемых ей процессов* должна быть *арифметической* и *числовой*, а не *алгебраической* и *аналитической*. Это ошибка. Машина может организовывать и комбинировать числовые величины в точности так, как если бы они были *буквами* или вообще *любыми* символами; на самом деле она могла бы выводить свои результаты в алгебраической нотации, если принять соответствующие меры. Она могла бы одновременно вырабатывать три набора результатов: *символических* (о чем уже было сказано в замечаниях A и B), *числовых* (основной ее объект) и *алгебраических* в *буквенной* нотации.

<...> Под *циклом* операций ... надлежит понимать набор операций, повторяемый более одного раза. Цикл остается циклом все равно, повторяется он всего два раза или неопределенное число раз, ибо его идея заключается именно в факте повторения. В анализе часто встречается *повторяющаяся группа*, состоящая из одного или нескольких *циклов*, т. е. цикл с *внутренним циклом*, который сам может содержать *внутренние циклы*. <...> А. А. Л.

3.7. Замечание F переводчика

Существует прекрасно вытканный портрет Жаккарда, для изготовления которого понадобилось 24 000 карт.

Возможность *повторения* карт, о которой упоминал М. Менабреа и которая была более подробно объяснена в замечании С, очень сильно уменьшает количество потребных карт. Очевидно, что это механическое усовершенствование особенно ценно, когда в математических операциях встречаются циклы, и что при подготовке данных для машинных вычислений желательно организовать порядок и сочетание процессов так, чтобы они были по возможности *симметричны* и цикличны; тогда механические преимущества системы *обратной подачи* можно будет задействовать в полной мере. Здесь интересно наблюдать за тем, как ценность *аналитического* ресурса *признается* и *усиливается* изобретательным механическим устройством. Мы видим в этом пример *взаимовлияния и взаимодополнения* чисто математических и механических частей машины, упомянутых в замечании А, и считаем, что это главное и самое существенное условие успешного изобретения счетной машины. Природа выигрышей от такого взаимодополнения двоякая. В некоторых случаях трудность (быть может, сама по себе непреодолимая) в одной части могла бы быть компенсирована средствами, относящимися к другой части, а иногда (как в данном случае) сильная сторона одной части может быть подкреплена и сделана еще сильнее и доступнее в результате сочетания с соответствующей сильной стороной другой части.

В качестве простого примера того, до какой степени комбинация циклов с обратной подачей может уменьшить *количество* потребных карт, рассмотрим ситуацию, которая делает это предельно наглядным и имеет еще и дополнительное преимущество, поскольку демонстрирует задачу, совершенно непохожую на те, что упоминались в других замечаниях. Предположим, что требуется исключить девять переменных из системы десяти простых уравнений вида

$$ax_0 + bx_1 + cx_2 + dx_3 + \ldots = p$$
$$a'x_0 + b'x_1 + c'x_2 + d'x_3 + \ldots = p'$$
$$\ldots$$

Прежде чем продолжить, мы должны пояснить, что нашей целью является не рассмотрение этой задачи с точки зрения фактического размещения данных в Переменных машины, а лишь абстрактный вопрос о природе и количестве операций, которые необходимо выполнить в ходе ее полного решения.

Первым шагом будет исключение первой неизвестной величины x_0 из первых двух уравнений. Для этого нужно привести их к виду

$$(a'a - aa')x_0 + (a'b - ab')x_1 + (a'c - ac')x_2 + \\ + (a'd - ad')x_3 + \ldots\ldots\ldots\ldots = a'p - ap',$$

для чего необходимо 10 операций (×, ×, –). Вторым шагом будет исключение x_0 из второго и третьего уравнений, для чего понадобятся точно такие же операции. Следовательно, всего нужно будет выполнить следующие операции:

$$10(\times, \times, -), 10(\times, \times, -) = 20(\times, \times, -).$$

Продолжая в том же духе, получим, что общее количество операций, необходимых для полного исключения x_0 из всех пар соседних уравнений, равно

$$9 \times 10(\times, \times, -) = 90(\times, \times, -).$$

После этого у нас останется девять простых уравнений с девятью переменными, из которых нужно будет исключить следующую переменную x_1; количество необходимых для этого операций равно

$$8 \times 9(\times, \times, -) = 72(\times, \times, -).$$

Теперь останется восемь уравнений с восьмью переменными, из которых нужно исключить x, для чего понадобится

$$7 \times 8(\times, \times, -) = 56(\times, \times, -)$$

операций и т. д. Таким образом, общее число операций, необходимых для исключения всех переменных, равно

$$9 \times 10 + 8 \times 9 + 7 \times 8 + 6 \times 7 + 5 \times 6 + 4 \times 5 + 3 \times 4 + 2 \times 3 + 1 \times 2 = 330.$$

Стало быть, *три* карты-Операции сделают то же, что 330 таких карт.

Если взять n простых уравнений с $n - 1$ переменными, где величина n не ограничена, то пример становится еще более наглядным, поскольку те же самые три карты могли бы заменить тысячи или миллионы таких карт.

Теперь мы хотим привлечь внимание к тому факту, уже отмеченному выше, что совершенно необязательно выводить точную формулу как необходимое условие для решения задачи машиной. Достаточно знать, какую *последовательность операций* следует выполнить. В примере выше это очевидно при самом поверхностном знакомстве. И это обстоятельство заслуживает отдельного упоминания, поскольку в нем может быть заключена скрытая ценность машины, которую сейчас даже невозможно в полной мере оценить. Мы уже знаем, что существуют функции, установить численное значение которых важно для целей как абстрактной, так и практической науки, но для их определения требуются процессы столь длительные и сложные, что, хотя достичь результата все же можно, затратив много времени, труда и денег, с практической точки зрения его можно считать почти недостижимым. И мы можем представить себе, что некоторых результатов достичь на практике с приемлемой точностью *абсолютно невозможно*, а между тем знание их точных значений может оказаться чрезвычайно важным для каких-то будущих потребностей науки – ее разнообразных, сложных и быстро развивающихся отраслей.

Не вступая, однако, в область гадания, упомянем конкретную проблему, которая в данный момент представляется нам убедительной иллюстрацией того, как подобную машину можно было бы применить в области, где человеческому мозгу трудно или даже невозможно работать без ошибок. В решении знаменитой задачи трех тел из приблизительно 295 коэффициентов лунных возмущений, найденных Т. Клаузеном (Astro^e. Nachrichten, No. 406) по результатам вычислений Бурга, двух, найденных Дамуазо, и од-

ного, найденного Буркхардтом, четырнадцать коэффициентов отличаются природой алгебраического знака, а из остальных всего 101 (или примерно треть) точно согласуются по знаку и по величине. Эти несогласованности, в общем и целом по отдельности небольшие, могут возникать либо из-за неправильного определения абстрактных коэффициентов в формулировке задачи, либо из-за расхождений в данных, выведенных из наблюдений, либо вследствие обеих причин. Первая причина – наиболее распространенный источник ошибок в астрономических вычислениях, и его-то машина могла бы полностью устранить.

Мы могли бы даже совершенно произвольно придумать правила для последовательности формул, настроить машину для работы с ними и таким образом вывести числовые результаты, о которых иначе даже не помыслили бы; но это упражнение вряд ли имело бы большую практическую ценность, и расценивать его стоило бы не выше, чем философическое развлечение. А. А. Л.

3.8. Замечание G переводчика

Крайне желательно остерегаться преувеличенных ожиданий относительно могущества Аналитической машины. При рассмотрении любого нового предмета часто возникает побуждение, во-первых, *превозносить* то, что мы уже считаем интересным или замечательным, а во-вторых, – и это естественная реакция – *принижать* истинное состояние дел, когда выясняется, что мы ожидали большего, чем следовало бы по зрелом размышлении.

Аналитическая машина не претендует на *сотворение* чего бы то ни было. Она может делать только то, что *мы знаем, как заставить ее делать*. Она может выполнить анализ, но не обладает возможностью *предвидеть* какие-то связи или истины. Ее удел – делать доступным то, с чем мы уже знакомы. Конечно, это в основном результаты вычислений, осуществляемых с помощью ее исполнительных механизмов, но не исключено, что машина может *косвенно* оказать обратное влияние на саму науку и другим способом. Ибо в процессе распределения и комбинирования истин и формул анализа, так чтобы они оказались наиболее просты и удобны для машинной обработки, природа многих предметов научного изучения и связей между ними обязательно предстанет в новом свете и будет исследована более тщательно. Это, безусловно, косвенное и в какой-то мере *умозрительное* следствие описанного изобретения. Однако довольно очевидно, исходя из общих соображений, что в ходе изобретения для математических истин новой формы, в которой они будут отливаться для фактического использования, с большой вероятностью появятся новые подходы, которые опять-таки должны реагировать на более теоретическую сторону предмета. При любом вырастании могущества человека или увеличении суммы наших знаний, помимо главной и основной цели, возникают различные *побочные* явления. <…> А. А. Л.

4 Исследование законов мышления, на которых основаны математические теории логики и вероятностей (1854)

Джордж Буль

Когда Джордж Буль (1815–1864) был студентом, через два тысячелетия после Аристотеля, «Первая аналитика» по-прежнему оставалась основным учебником логики. XVIII век был временем поразительного прогресса в области непрерывной математики. Орбиты планет и поведение других движущихся тел стало возможно точно рассчитать – пусть даже это было долго и утомительно. Но вопреки представлениям Лейбница об исчислении идей, аналогичном его исчислению бесконечно малых, и робкой попытке разработать формальные правила рассуждения, логика оставалась по существу такой же, как в античные времена, разве что чуть-чуть более систематизированной. Буль осознал, что логика – часть математики, а значит, подчиняется правилам, которые могли бы быть положены в основу исчисления высказываний. Его целью было «дать в этом трактате выражение фундаментальных законов рассуждения на символическом языке Исчисления» (§ 4.1).

Предшественником Буля был математик Джордж Пикок, входивший в кружок кембриджских интеллектуалов наряду с Чарльзом Бэббиджем, – в своем «Трактате по алгебре» (Peacock 1830) он предлагал рассматривать алгебру как «науку об общем рассуждении посредством символического языка». И все же публикация «Законов мышления» Буля стала поворотным пунктом в развитии информатики. Осуществленное им сведение логики к алгебре истинных и ложных переменных оказало влияние на других современных ему логиков, в т. ч. на Огастеса де Моргана (все мы помним его правила) и Джона Венна (запомнившегося по диаграммам). Через несколько лет после публикации «Законов мышления» земляк Буля Уильям Джевонс (1835–1882) построил «логическое пианино» для вычислений со значениями истинности высказываний. Устройство было не особенно полезным, ограничивалось всеми четырьмя высказываниями, но стало первой попыткой механизировать то, что теперь мы называем булевой логикой.

Булева концепция логической алгебры на самом деле выходит за пределы булевой логики и охватывает то, что теперь мы назвали бы наивной теорией множеств и теорией вероятностей. На стр. 58 Буль объясняет слово «класс» – сегодня мы называем его множеством – и явно утверждает, что класс может быть пустым, содержать один элемент или совпадать с универсумом. Буль четко формулирует, что переменную *x* можно равным образом рассматривать как высказывание, утверждающее, что индивидуум обладает данным свойством, или как класс сущностей, обладающих этим свойством. Далее он переходит к использованию знаков «+» для объединения, «–» для разности и «×» (или просто записи рядом) для пересечения, к выводу законов коммутативности и дистрибутивности и к выводу принципа двойственности, или дихотомии (ничто не может принадлежать одновременно множеству и его дополнению) алгебраически из идемпотентности (пересечение множества с самим собой дает то же самое множество). Его книга устроена как последовательность определений, предложений и «правил», все они имеют общую цель – научить алгебраическим манипуляциям с логическими формулами (а затем и с вероятностями).

Джордж Буль был математиком-самоучкой, сыном английского сапожника. Он получил только начальное образование. Он сумел опубликовать серьезные математические труды, когда ему еще не исполнилось и тридцати лет, а на жизнь приходилось зарабатывать профессией школьного учителя. Через десять лет он получил звание профессора в новом Квинз-колледже в Корке, Ирландия, одном из трех колледжей, которые одновременно учредила королева Виктория (два других находятся в Белфасте и Голуэе). Именно в этом захолустье, вдали от крупных математических центров Оксфорда и Кембриджа, Буль и написал свое выдающееся сочинение, из которого мы выбрали лишь небольшой фрагмент.

На этих страницах Буль терпеливо объясняет некоторые правила и методы логики высказываний, в т. ч. и закон коммутативности. Сейчас все это кажется нам очевидным, так мы привыкли к казавшейся тогда новаторской идее о том, что к значениям «истина» и «ложь» можно применять алгебраические операции. Спустя почти сто лет Клод Шеннон включил все идеи Буля в только нарождающийся мир цифровых схем (глава 8). Выбранный нами фрагмент

заканчивается идеей функции истинности и глубокой мыслью о том, что такую функцию можно разложить на подфункции, положив значение одной переменной равным true или false. Буль сравнивал этот процесс с разложением дифференцируемой функции в ряд Тейлора.

Буль умер в возрасте 49 лет от инфекции, которую подхватил, когда пешком шел три мили на свою лекцию под холодным ирландским дождем (MacHale, 2014).

4.1. Природа и построение этой работы

Цель следующего далее трактата – исследовать фундаментальные законы тех операций разума, с помощью которых осуществляется рассуждение; дать им выражение на символическом языке Исчисления и на этом фундаменте основать науку Логики и построить ее метод; сделать сам этот метод основой общего метода применения математической доктрины Вероятностей; и, наконец, извлечь из различных элементов истины, явленных взору в ходе этих исследований, некоторые правдоподобные мысли о природе и складе человеческого разума. <…>

4.2. О знаках вообще и о знаках, применимых к науке Логики в частности; также и о законах, которым этот класс знаков подчинен

4.2.1. То, что Язык является инструментом человеческого разума, а не просто средством выражения мыслей, – общепризнанная истина. В этой главе предлагается задаться вопросом, что именно делает Язык столь полезным подспорьем для наших самых важных умственных способностей. Поиск ответа на этот вопрос на различных этапах приведет нас к изучению строения Языка, рассматриваемого как система, предназначенная для достижения некоей конечной цели; к исследованию его элементов; к попытке определить их взаимосвязи и зависимости; и к вопросу о том, каким образом они вносят вклад в достижения цели, к которой, будучи взаимоувязанными частями системы, имеют отношение. <…>

4.2.2. Элементами, составляющими любой язык, являются знаки, или символы. Слова – это знаки. Иногда их произносят, чтобы обозначить предметы; иногда – операции, посредством которых ум комбинирует простые понятия предметов в более сложные концепты; иногда они выражают отношения действия, страсти или просто качество, которые, как нам представляется, существуют между предметами, известными нам по опыту; иногда – эмоции постигающего разума. Но слова, хотя они этим и другими способами выпол-

няют функции знаков, или представительных символов, – не единственные знаки, которыми мы можем воспользоваться. Любые приметы, что-то говорящие только глазу, любые звуки или действия, обращенные к другим органам чувств, равным образом имеют природу знаков, коль скоро их предназначение определено и понятно. В математических науках буквы и символы +, –, = и т. д. используются в роли знаков, хотя сам термин «знак» применяется к последнему классу символов, которые представляют операции или отношения, а не к первому, представляющему число и количество. Поскольку истинный смысл знака ни в коей мере не зависит от его конкретной формы или выражения, таковы же и законы, определяющие его использование. Однако в настоящем трактате мы вынуждены иметь дело с письменными знаками, и только по отношению к ним и будет употребляться термин «знак».

Существенные свойства знаков перечислены в следующем определении.

Определение. Знаком называется произвольный отпечаток, имеющий фиксированное толкование и допускающий комбинирование с другими знаками в рамках фиксированных законов, зависящих от их взаимного толкования.

<…>

4.2.4. Анализ и классификация тех знаков, посредством которых осуществляются операции рассуждения, являются предметом следующего Предложения:

Все операции Языка как инструмента рассуждения могут быть выполнены системой знаков, состоящей из следующих элементов:

1. *Литеральные символы, например x, y и т. д., представляющие вещи как субъекты наших концептов.*

2. *Знаки операций, например* +, –, ×, *обозначающие те умственные операции, с помощью которых концепты вещей комбинируются или разрешаются так, чтобы образовать новые концепты, образованные теми же элементами.*

3. *Знак тождества* =.

И эти символы Логики в своем использовании подчиняются определенным законам, частично совпадающим, а частично отличающимся от законов действий с соответствующими элементами в науке Алгебры.

Примем в качестве критерия пригодности элемента к рациональному рассуждению, что элементы должны допускать комбинирование в простейших формах и подчиняться простейшим законам и что при таком комбинировании должны порождаться все известные и мыслимые формы языка, и, приняв этот принцип, рассмотрим следующую классификацию.

4.2.5. Класс I. *Нарицательные или описательные знаки, выражающие имя вещи или какое-то присущее ей качество или обстоятельство.*

К этому классу мы, очевидно, должны отнести имена существительные, нарицательные или собственные, и имена прилагательные. Можно считать, что они отличаются только в том отношении, что первое выражает независимое существование индивидуальной вещи или вещей, к которым относится, а второе подразумевает такое существование. Если присоединить к прилагательному универсально понимаемый субъект – «существо» или «вещь», – то оно фактически становится существительным и для всех действительно важных целей рассуждения может быть заменено существительным. Не важно,

можно ли во всех умственных отношениях считать фразы «вода есть жидкая вещь» и «вода жидкая» одним и тем же; по крайней мере, при выражении процессов рассуждения они эквивалентны.

Ясно также, что к вышеупомянутому классу мы должны отнести любой знак, который традиционно используется для выражения какого-то обстоятельства или отношения, детальное раскрытие которого потребовало бы использования многих знаков. Таковы очень часто поэтические эпитеты. Обычно это составные прилагательные, одним словом выражающие многословное описание. Гомеровский «глубокопучинный Океан» содержит целое описание в одном слове βαθυδίνηζ. И по традиции любое другое описание, обращенное к воображению или к разуму, может быть эквивалентно представлено одним знаком, использование которого во всех существенных аспектах подчинялось бы тем же законам, что использование прилагательного «хороший» или «великий». В сочетании с субъектом «вещь» такой знак, по сути дела, стал бы именем существительным; и посредством единственного существительного можно было бы выразить значение, объединяющее в себе вещь и качество.

4.2.6. Теперь, когда определено, что знак – это произвольный отпечаток, разрешается заменить все знаки описанного выше вида буквами. Условимся представлять класс объектов, к которым применимо некоторое название или описание, одной буквой, например *x*. Если это название, например, «люди», то пусть *x* представляет «всех людей», или класс «люди». Под классом обычно понимается набор объектов, к каждому из которых можно применить определенное название или описание; но в настоящей работе этот термин будет употребляться в обобщенном смысле, включая случай, когда существует всего один объект, отвечающий требуемому названию или описанию, а также случаи, обозначаемые терминами «ничто» или «универсум», когда под «классами» следует понимать соответственно «ни одной сущности» и «все сущности». Также если прилагательное, например «хороший», употребляется как часть описания, будем представлять буквой, скажем *y*, все вещи, к которым приложимо описание «хороший», т. е. «все хорошие вещи» или класс «хорошие вещи». Условимся далее, что комбинация *xy* будет представлять класс вещей, к которым одновременно приложимы названия или описания, представленные *x* и *y*. Таким образом, если *x* само по себе означает «белые вещи», а *y* – «овца», то *xy* будет обозначать «белая овца», и точно так же если *z* обозначает «рогатые вещи», а *x* и *y* – то же, что и выше, то *zxy* будет обозначать «рогатая белая овца», т. е. совокупность вещей, к которым приложимы название «овца» и описания «белый» и «рогатый».

Рассмотрим теперь законы, которым подчиняются символы *x*, *y* и т. д., используемые в указанном выше смысле.

4.2.7. Во-первых, очевидно, что в описанных выше комбинациях порядок записи символов не важен. Оба выражения *xy* и *yx* представляют класс вещей, к членам которого одновременно приложимы названия или описания *x* и *y*. Следовательно, мы имеем

$$xy = yx. \tag{4.1}$$

В случае, когда *x* представляет белые вещи, а *y* – овцу, любая часть этого равенства представляет класс «белых овец». Способы формирования этого концепта могут различаться, но это никак не отражается на отдельных объектах, которые он охватывает. Аналогично, если *x* представляет «устья», а *y* – «реки», выражения *xy* и *yx* будут представлять соответственно «реки, являющиеся устьями» и «устья, являющиеся реками»; в обыкновенном языке это комбинация двух существительных, а не существительного и прилагательного, как в предыдущем примере.

Пусть имеется третий символ, *z*, представляющий класс вещей, к которым применим термин «судоходный»; тогда любое из следующих выражений, *zxy*, *zyx*, *xyz* и т. д., будет представлять класс «судоходных рек, являющихся устьями».

Если один из описательных членов каким-то образом подразумевает ссылку на другой, то нужно только явно включить эту ссылку в ее подразумеваемом значении, и тогда все сделанные выше замечания останутся в силе. Так, если *x* представляет «мудрый», а *y* – «советник», то мы должны определить, подразумевает ли *x* мудрость в абсолютном смысле или только применительно к совету. При таком определении закон *xy* = *yx* остается верным.

Следовательно, разрешено употреблять символы x, y, z и т. д. вместо существительных, прилагательных и описательных фраз при условии выполнения правила толкования, согласно которому любое выражение, в котором несколько символов записаны вместе, представляет все объекты, или индивидуумы, к которым одновременно приложимо несколько значений, и закона, по которому порядок следования символов не важен.

Поскольку для иллюстрации правила толкования было приведено достаточно примеров, я считаю излишним всякий раз добавлять субъект «вещи» при определении толкования символа, используемого в качестве прилагательного. Когда я говорю, что *x* будет представлять «хороший», следует понимать, что *x* представляет «хороший», только когда с помощью какого-то другого символа указан субъект этого качества, и что, будучи употреблен самостоятельно, *x* толкуется как «хорошие вещи».

4.2.8. Говоря об определенном выше законе, можно добавить следующие наблюдения, в большей или меньшей степени справедливые и для некоторых других законов, которые будут выведены ниже.

Во-первых, я бы отметил, что этот закон, строго говоря, является законом мышления, а не законом вещей. Разница в порядке перечисления качеств или атрибутов объекта, если отвлечься от вопросов причинности, – это всего лишь разница в постижении. Закон (4.1) выражает общую истину – одну и ту же вещь можно постигать разными способами – и описывает природу различия; ничего сверх того в нем нет.

Во-вторых, будучи законом мышления, он фактически представлен в виде закона Языка, продукта и инструмента мышления. Хотя в прозаических текстах наблюдается тенденция к монотонности, даже там порядок следования прилагательных, абсолютных по смыслу и примененных к одному и тому же субъекту, безразличен, но поэтическая речь своим богатством в немалой степени обязана распространению такой же узаконенной свободы и на сущест-

вительные. Особенно этим видом разнообразия славится язык Мильтона. Мало того что существительные у него часто предшествуют прилагательным, их определяющим, так они еще нередко помещаются между ними. В первых же строках его обращения к Свету мы встречаем такие примеры:

«*Offspring of heaven first-born*».
«The rising world of *waters dark and deep*».
«Bright effluence of *bright essence increate*»[1].

Но такие инвертированные формы – не прерогатива одной лишь поэзии. Они суть естественное выражение свободы, данной сокровенными законами мышления, но по причинам удобства не применяемой при обыденном использовании языка.

В-третьих, закон (4.1) можно охарактеризовать, сказав, что литеральные символы x, y, z коммутативны, как символы Алгебры. Говоря так, мы не утверждаем, что процесс умножения в Алгебре, фундаментальный закон которого выражается равенством $xy = yx$, несет в себе какую-то аналогию с процессом логического комбинирования, представленного выражением xy; утверждается лишь, что если арифметический и логический процессы записываются одинаково, то их символические выражения должны подчиняться одному и тому же формальному закону. Проявления такого подчинения в обоих случаях отчетливо видны.

4.2.9. Из того, что комбинация двух литеральных символов в форме xy выражает весь класс объектов, к которым одновременно приложимы названия или качества, представленные x и y, следует, что если оба символа имеют в точности одинаковый смысл, то их комбинация выражает не более того, что можно было бы выразить с помощью любого из этих символов по отдельности. А раз так, то должно быть $xy = x$. Но поскольку предполагается, что y имеет такой же смысл, как x, мы можем заменить его в этом равенстве на x, получив при этом $xx = x$. В общей Алгебре комбинацию xx записывают короче в виде x^2. Применим и здесь такой же принцип обозначения; ибо способ выражения определенной последовательности умственных операций – вещь столь же произвольная, как выражение одной идеи или операции. При таких обозначениях приведенное выше равенство принимает вид

$$x^2 = x \tag{4.2}$$

и фактически является вторым общим законом символов, которыми представлены названия, качества или описания.

Читатель должен иметь в виду, что хотя символы x и y в примерах выше получали разные смысловые значения, ничто не препятствует нам приписать им в точности одно и то же значение. Очевидно, что чем ближе их смысловые значения, тем ближе класс вещей, обозначаемых комбинацией xy, оказы-

---

[1] О, Свет святой! О, первенец Небес!
Безмерной, – мир глубоких, черных вод.
О, излученный блеск субстанции несозданной.
(*Дж. Мильтон.* Потерянный рай / пер. А. Штейнберга. Кн. 3.)

вается к тождественности с классом, обозначаемым *x*, равно как и с классом, обозначаемым *y*. Случай, описываемый равенством (4.2), – это случай абсолютной тождественности значений. Выражаемый им закон находит практическое подтверждение в языке. Фраза «хороший, хороший» в отношении любого предмета является громоздким и бесполезным плеоназмом, но означает она то же самое, что просто «хороший». Таким образом, фразы «хорошие, хорошие люди» и «хорошие люди» эквивалентны. Подобное повторение слов иногда действительно используется, чтобы усилить качество или эмоционально подчеркнуть согласие. Но этот эффект вторичен и условен; он не коренится во внутренних связях языка и мышления. Большинство операций, наблюдаемых нами в природе или выполняемых нами самими, такого рода, что их эффект усиливается от повторения, благодаря этому обстоятельству мы и подготовлены ко встрече с тем же самым в языке и даже используем повторение, когда хотим что-то подчеркнуть. Но ни в строгом рассуждении, ни в точном повествовании нет ровно никакого обоснования для такой практики.

4.2.10. Перейдем теперь к рассмотрению еще одного класса знаков речи и законов, связанных с их использованием.

4.2.11. Класс II. *Знаки умственных операций, посредством которых мы собираем части в целое или разделяем целое на части.*

Нам может быть интересна не только идея объектов, характеризуемых названиями, качествами или обстоятельствами, общими для каждого элемента рассматриваемой группы, но также идея агрегирования, т. е. образования группы объектов, состоящей из частичных групп, каждая из которых имеет отдельное название или описание. Для этой цели мы используем союзы «и», «или» и т. д. Примерами могут служить «деревья и минералы», «бесплодные горы или плодородные долины». Строго говоря, слова «и», «или», располагающиеся между членами, описывающими два или более классов объектов, подразумевают, что эти классы совершенно различны, т. е. ни один элемент одного не принадлежит другому. В этом и во всех прочих отношениях слова «и», «или» аналогичны знаку + в алгебре и подчиняются тем же самым законам. Таким образом, выражение «мужчины и женщины», если отбросить традиционно сопровождающие его смыслы, эквивалентно выражению «женщины и мужчины». Пусть *x* представляет класс «мужчины», *y* – класс «женщины», и пусть знак + может заменять «и» и «или»; тогда имеем $x + y = y + x$, равенство, которое точно так же имеет место, если *x* и *y* – числа, а + – знак арифметического сложения.

Пусть символ *z* обозначает прилагательное «европейский», тогда, поскольку фраза «европейские мужчины и женщины» означает то же самое, что «европейские мужчины и европейские женщины», имеем

$$z(x + y) = zx + zy. \tag{4.3}$$

И это равенство тоже было бы истинным, если бы символы *x*, *y* и *z* были числами, а запись двух литеральных символов подряд представляла бы их алгебраическое произведение, точно так же, как в описанном выше логиче-

ском истолковании оно представляет класс объектов, к которым приложимы оба эпитета.

Сформулированные выше законы определяют использование знака +, который здесь служит для обозначения позитивной операции агрегирования частей в единое целое. Но сама идея позитивной операции, вносящей некоторые изменения, наводит на мысль об обратной, или негативной, операции, действие которой заключается в отмене результата первой операции. Таким образом, мы не можем признать возможным объединение частей в целое, не признав также возможным отделение части от целого. В обыденном языке мы выражаем эту операцию знаком «кроме», например «все мужчины, кроме азиатских» или «все государства, кроме монархических». Здесь имеется в виду, что исключенные вещи образуют часть той совокупности вещей, из которой они исключены. Поскольку операцию агрегирования мы выразили знаком +, то обратную ей операцию можно описать знаком –. Следовательно, если считать, что x представляет мужчин, а y – азиатских, т. е. азиатских мужчин, то концепт «все мужчины, кроме азиатских» будет выражаться в виде $x - y$. А если x представляет «государства», а y – описательное свойство «имеющие монархическую форму правления», то концепт «все государства, кроме монархических» будет выражаться в виде $x - xy$.

Поскольку для всех *существенных* целей рассуждения безразлично, упоминаем ли мы исключенные случаи в разговоре сначала или потом, так же безразлично и то, в каком порядке записываются члены, некоторые из которых предварены знаком –. Следовательно, как и в общей алгебре, мы имеем

$$x - y = -y + x.$$

Пусть x по-прежнему представляет класс «мужчины», а y – класс «азиатские», и пусть z представляет прилагательное «белый». Тогда применить прилагательное «белый» к совокупности мужчин, определяемой фразой «мужчины, кроме азиатских», – то же самое, что сказать «белые мужчины, кроме азиатских». Следовательно, имеем

$$z(x - y) = zx - zy. \tag{4.4}$$

И это тоже находится в полном согласии с законами обычной алгебры.

Равенства (4.3) и (4.4) можно рассматривать как проявления одного общего закона, который можно сформулировать, сказав, что *литеральные символы x, y, z дистрибутивны относительно операций над ними*. Общий факт, выражаемый этим законом, таков: если какое-то качество или обстоятельство приписано всем членам группы, образованной агрегированием или исключением частичных групп, то получающийся в результате концепт будет таким же, как если бы это качество или обстоятельство было сначала приписано каждому члену частичных групп, а затем применено агрегирование или исключение. То, что приписано членам целого, приписано также членам всех его частей, как бы эти части ни были соединены друг с другом.

4.2.12. Класс III. *Знаки, посредством которых выражается отношение и образуются высказывания.*

Хотя к этому классу можно с полным основанием отнести все глаголы, для целей Логики достаточно считать, что он включает только субстантивный глагол *является* или *являются*, потому что любой другой глагол можно свести к этому элементу и одному из знаков, включенных в Класс I. Ибо поскольку эти знаки используются для выражения качества или обстоятельства любого вида, их можно использовать для выражения активного или пассивного отношения к субъекту глагола, дополненного ссылкой на прошлое, настоящее или будущее время. Так, высказывание «Цезарь победил галлов» можно привести к виду «Цезарь является тем, кто победил галлов». Основания для такого анализа мне представляются следующими: если мы не понимаем, что значит победить галлов, т. е. смысла выражения «тот, кто победил галлов», то не сможем понять смысла рассматриваемого предложения. Это, следовательно, неотъемлемый элемент предложения; другим элементом является «Цезарь», и нужен еще один – связка, показывающая, как они соотносятся друг с другом. Я, однако, не утверждаю, что не существует никакого другого способа истолковать отношение, выраженное высказыванием «Цезарь победил галлов»; я лишь говорю, что приведенный здесь анализ правилен для принятой нами точки зрения и что его достаточно для целей логической дедукции. Можно отметить, что пассивные причастия и причастия будущего времени в греческом языке подразумевают наличие отстаиваемого тезиса, а именно что знак *является* или *являются* можно рассматривать как элемент любой личной формы глагола.

4.2.13. Рассмотренный выше знак *является* или *являются* можно выразить символом =. Законы, или, как обычно говорят, аксиомы, вводимые этим символом, будут рассмотрены ниже. Возьмем высказывание «Звездами являются солнца и планеты», и пусть x представляет звезды, y – солнца, а z – планеты; тогда

$$x = y + z. \tag{4.5}$$

Если верно, что звездами являются солнца и планеты, то отсюда следует, что все звезды, кроме планет, являются солнцами. Это дало бы равенство $x - z = y$, которое, следовательно, должно быть дедукцией из (4.5). Таким образом, член z был бы перенесен из одной части равенства в другую с изменением его знака. Это вполне согласуется с алгебраическим правилом переноса.

Но вместо того чтобы задерживаться на частных случаях, мы можем сразу ввести общие аксиомы:

1) если равные вещи прибавляются к равным вещам, то целые равны;
2) если равные вещи отнимаются от равных вещей, то остатки равны.

И отсюда следует, что мы можем складывать и вычитать равенства и применять описанное выше правило переноса, как в обычной алгебре.

Повторю: если два класса вещей, x и y, идентичны, т. е. если каждый член одного класса является также членом другого, то те члены одного класса, которые обладают данным свойством z, совпадают с членами другого класса, обладающими тем же свойством z. Следовательно, если мы имеем равенство

$x = y$, то какой бы класс или свойство ни представлял знак z, имеет место также равенство $zx = zy$.

Формально это то же самое, что алгебраический закон: если обе части равенства умножить на одну и ту же величину, то произведения будут равны.

Точно так же можно показать, что если соответственные части двух равенств перемножить, то в результате также получится истинное равенство.

4.2.14. Здесь, однако, аналогия представленной системы с обычной алгеброй заканчивается. Предположим, что те члены класса x, которые обладают некоторым свойством z, совпадают с членами класса y, обладающими тем же свойством z; отсюда не следует, что все члены класса x совпадают с членами класса y. Значит, из равенства $zx = zy$ нельзя вывести, что равенство $x = y$ также истинно. Иными словами, аксиома алгебры, согласно которой обе части равенства можно разделить на одну и ту же величину, здесь не имеет формального эквивалента. Я говорю об отсутствии *формального эквивалента*, потому что в соответствии с общим духом этого исследования даже не преследуется цель определить, верно ли, что умственная операция, представляемая исключением логического символа z из комбинации zx, аналогична операции деления в Арифметике. Эта умственная операция в действительности идентична тому, что принято называть Абстракцией, и поэтому ее законы зависят от законов, выведенных ранее в этой главе. Мы только что показали, что среди этих законов не существует ничего аналогичного по форме одной общепринятой аксиоме Алгебры.

Но небольшое замечание покажет, что даже в обычной алгебре эта аксиома не обладает общностью, присущей другим рассмотренным аксиомам. Вывод равенства $x = y$ из равенства $zx = zy$ допустим только тогда, когда известно, что z не равно 0. Коль скоро значение $z = 0$ признается допустимым в алгебраической системе, эта аксиома перестает быть применимой, и продемонстрированная выше аналогия не нарушается.

4.2.15. Однако не является сколько-нибудь важным, если оставить в стороне умозрительные размышления, прослеживать такие сходства для символов количества. Мы видели (§ 4.2.9), что символы Логики подчиняются специальному закону $x^2 = x$. Среди символов Чисел есть всего два, 0 и 1, подчиняющихся тому же формальному закону. Мы знаем, что $0^2 = 0$ и что $1^2 = 1$ и что уравнение $x^2 = x$, рассматриваемое как алгебраическое, не имеет других корней, кроме 0 и 1. Поэтому вместо того чтобы определять меру формального сходства символов Логики с символами Числа в общем случае, гораздо разумнее сравнить их с символами количества, допускающими значения 0 и 1. Будем тогда говорить об Алгебре, в которой символы x, y, z и т. д. допускают только значения 0 и 1 и никакие другие. Законы, аксиомы и процессы такой Алгебры будут во всем идентичны законам, аксиомам и процессам Алгебры Логики. Отличает их только различие в интерпретации. Этот принцип положен в основу последующей работы.

4.2.16. Остается показать, что те составные части обыденного языка, которые не были рассмотрены в предыдущих разделах этой главы, либо сводятся к тем элементам, которые были рассмотрены, либо подчинены этим элементам и уточняют их определение.

Существительное, прилагательное и глагол, а также частицы *и*, *кроме* мы уже рассмотрели. Местоимение можно рассматривать как частный случай существительного или прилагательного. Наречие изменяет значение глагола, но не изменяет его природу. Предлоги привносят выражение обстоятельств или отношений между объектом и субъектом, а потому уточняют и детализируют значение литеральных символов. Союзы *если*, *либо*, *или* используются главным образом для выражения отношения между высказываниями, и ниже будет показано, что те же самые отношения можно в полной мере выразить с помощью элементарных символов, аналогичных по толкованию и тождественных по форме и законам символам, использование и смысл которых уже были объяснены в этой главе. Что же до остальных частей речи, то при ближайшем рассмотрении обнаруживается, что они используются либо для того, чтобы придать большую важность элементам повествования, и, таким образом, подпадают под толкование уже рассмотренных литеральных символов, либо для того, чтобы выразить какую-то эмоцию или ощущение, сопровождающее высказывание, а потому не относятся к области понимания, которая только и интересна нам в этой работе. Опыт применения свидетельствует о достаточности принятой нами классификации.

4.3. Вывод законов символов логики из законов операций человеческого разума...

4.3.12. Оставшуюся часть этой главы мы посвятим вопросам, относящимся к закону мышления, имеющему выражение $x^2 = x$ (§ 4.2.9), закону, который, как было установлено (§ 4.2.15), составляет характеристическое отличие операций разума, когда он функционирует в обычном режиме обсуждения и рассуждения, по сравнению с его операциями, когда он занят общей алгеброй величин. Важная часть последующего изложения заключается в доказательстве того, что символы 0 и 1 занимают место среди символов Логики и допускают толкование как таковые; сначала необходимо показать, каким образом такие символы, как эти, могут с полным основанием и к вящей выгоде быть использованы в представлении различных систем мышления.

Такое основание не может заключаться ни в какой общности толкования. Ибо в системах мышления столько очевидно различающихся, как Логика и Арифметика (я буду употреблять последний термин в самом широком смысле, как науку о Числе), не существует, по сути дела, никакой общности предмета. Одна из них имеет дело с самими концептами вещей, а другая принимает во внимание только их числовые отношения. Но ввиду того, что формы и методы любой системы рассуждений непосредственно зависят от законов, которым подчиняются символы, и только опосредованно, посредством упомянутого выше звена связи, от их толкования, возможно и основание, и выгода в использовании одних и тех же символов в различных системах мышления, при условии что им могут быть приписаны такие толкования, при которых их формальные законы будут одинаковы, а использование согласовано. Тогда основанием для такого использования будет не общность тол-

кования, а общность формальных законов, которым символы подчиняются в своих системах. И эта общность формальных законов не должна устанавливаться ни на каком другом основании, кроме тщательного наблюдения и сравнения результатов, которые, как показывает наблюдение, независимо вытекают из толкований рассматриваемых систем.

Эти наблюдения объяснят процесс исследования, принятый в следующем Предложении. Литеральные символы Логики универсально подчиняются закону, выражением которого является $x^2 = x$. Из символов Чисел существует только два, 0 и 1, удовлетворяющих этому закону. Но каждый из этих символов также подчиняется своеобычному закону в системе числовых величин, и это наводит на мысль, что должны быть даны толкования литеральным символам Логики, чтобы одни и те же своеобычные и формальные законы могли быть реализованы также в логической системе.

4.3.13. ***Определить логическое значение и смысл символов 0 и 1.*** В Алгебре символ 0 удовлетворяет следующему формальному закону

$$0 \times y = 0, \text{ или } 0y = 0, \tag{4.6}$$

какое бы число ни было представлено символом y. Чтобы этот формальный закон выполнялся в системе Логики, мы должны приписать символу 0 такое толкование, чтобы класс, представленный $0y$, совпадал с классом, представленным 0, каким бы ни был класс y. Легко показать, что это условие выполняется, если символ 0 представляет Ничего. В соответствии с приведенным выше определением мы можем назвать Ничего классом. На самом деле Ничего и Универсум – два крайних случая классов, ибо они представляют пределы возможных толкований общих имен: ни одно не может относиться к меньшему числу объектов, чем содержится в Ничего, или к большему числу объектов, чем содержится в Универсуме.

Понятно, что каким бы ни был класс y, объекты, общие для него и класса Ничего, совпадают с объектами, содержащимися в классе Ничего, потому что их нет. И таким образом, приписав символу 0 толкование Ничего, мы удовлетворим закон (4.6); а по-другому удовлетворить его для любого произвольного класса y невозможно.

Далее, символ 1 в системе Чисел удовлетворяет закону

$$1 \times y = y, \text{ или } 1y = y,$$

какое бы число ни было представлено символом y. И если предположить, что это формальное равенство справедливо и в рассматриваемой в настоящей работе системе, где 1 и y представляют классы, то окажется, что символ 1 должен представлять такой класс, что все объекты, присутствующие в *любом* предложенном классе y, являются также объектами $1y$, общими для класса y и класса, представленного символом 1. Нетрудно понять, что класс, представленный символом 1, должен быть классом Универсум, т. к. это единственный класс, в котором присутствуют все объекты, существующие в любом классе. Итак, толкования символов 0 и 1 в системе Логики – *Ничего* и *Универсум*.

4.3.14. Если имеется класс объектов «люди», то на ум приходит идея противоположного класса существ, не являющихся людьми; а поскольку в совокупности эти два класса образуют Универсум, т. к. для любого входящего в него объекта можно утверждать, что он является либо человеком, либо не человеком, то возникает вопрос, как выразить эти противоположные названия. Это и есть предмет следующего Предложения.

Если x представляет любой класс объектов, то пусть 1 – *x представляет противоположный, или дополнительный, класс объектов, т. е. класс, включающий все объекты, не входящие в класс x.*

Чтобы лучше понять смысл, пусть *x* представляет класс «люди», а Универсум, согласно предыдущему Предложению, представлен символом 1. Тогда если из концепта Универсум исключить концепт «люди», то в результате получится концепт противоположного класса, «не люди». Следовательно, концепт «не люди» будет представлен как 1 – *x*. И вообще, какой бы класс объектов ни был представлен символом *x*, противоположный класс выражается как 1 – *x*.

4.3.15. Хотя следующее Предложение, строго говоря, относится к следующей главе этой работы, посвященной теме максим, или необходимых истин, все же, исходя из большой важности закона мышления, к которому оно относится, было сочтено уместным включить его сюда.

Аксиома метафизики, называемая законом противоречия и утверждающая, что ни одно существо не может одновременно обладать и не обладать некоторым качеством, является следствием фундаментального закона мышления, выражаемого в виде $x^2 = x$.

Запишем это равенство в форме $x - x^2 = 0$, откуда получаем

$$x(1 - x) = 0. \tag{4.7}$$

Оба этих преобразования основаны на аксиоматических законах комбинирования и переноса в другую часть (§ 4.2.13). Придадим, для простоты объяснений, символу *x* конкретное толкование «люди», тогда 1 – *x* будет представлять класс «не люди» (§ 4.3.14). Формальное произведение выражений обоих классов представляет класс объектов, общих для обоих сомножителей (§ 4.2.9). Следовательно, $x(1 - x)$ представляет класс, члены которого должны быть одновременно людьми и не людьми, и, таким образом, равенство (4.7) выражает принцип, согласно которому *класса, члены которого одновременно являются людьми и не людьми, не существует*. Иными словами, никакой объект не может быть в одно и то же время человеком и не человеком. Теперь обобщим смысл символа *x*, позволив ему представлять любой класс существ, обладающих определенным свойством, не важно каким; тогда равенство (4.7) выражает невозможность одновременно обладать и не обладать свойством. Но это то же самое, что «закон противоречия», который Аристотель называл фундаментальной аксиомой философии. «Не может быть, чтобы одно и то же качество принадлежало и не принадлежало одной и той же вещи. <...> Это самый важный из всех законов. <...> Посему всякий доказывающий ссылается на него как на непреложный факт. Ибо по природе своей это источник всех прочих аксиом».

Приведенное выше толкование введено не потому, что оно имеет непосредственную ценность в настоящей системе, но как иллюстрация важного факта в философии умственной силы, а именно: то, что принято считать фундаментальной аксиомой метафизики, на самом деле является следствием из закона мышления, математического по своей форме. Я хочу также привлечь внимание к тому обстоятельству, что равенство (4.7), выражающее этот фундаментальный закон мышления, является уравнением второй степени. <...>

4.3.16. Закон мышления, выражаемый равенством (4.7), по причинам, понятным из предыдущего обсуждения, иногда будет называться «законом двойственности».

4.4. О разделении высказываний на два класса: «первичные» и «вторичные», о характеристических свойствах этих классов и о законах выражения вторичных высказываний

4.4.1. Исследовав законы умственных операций, участвующих в процессах Постижения или Воображения, и объяснив соответствующие им законы символов, которыми они представлены, мы подходим к рассмотрению практического применения полученных результатов: во-первых, к выражению сложных членов высказываний, во-вторых, к выражению высказываний и, наконец, к построению общего метода дедуктивного анализа. В этой главе мы кратко рассмотрим первый из трех предметов, как введение, необходимое, чтобы установить следующее Предложение.

Все логические высказывания можно рассматривать как принадлежащие одному или другому из двух больших классов, которые можно назвать «Первичными», или «Конкретными», Высказываниями и «Вторичными», или «Абстрактными», Высказываниями.

Любое наше утверждение можно отнести к одному из следующих двух видов. Либо оно выражает отношение между вещами, либо выражает или эквивалентно выражению отношения между высказываниями. Утверждение, касающееся свойств вещей или явлений, проявлениями которых они являются, или обстоятельств, в которые они помещены, является, строго говоря, утверждением о связи между вещами. Сказать «снег белый» – с точки зрения логики все равно, что сказать «снег является белой вещью». Утверждение, касающееся фактов или событий, их взаимосвязи и взаимозависимости, с точки зрения той же логики в общем случае эквивалентно утверждению о том, что такие-то и такие-то высказывания, касающиеся этих событий, находятся в некотором отношении между собой, не противоречащем их взаимной истинности или ложности. Первый класс высказываний, относящихся к вещам, я называю «Первичным», а второй, относящийся к высказываниям, – «Вторичным». На практике различие между ними очень близко, но не

совсем совпадает с обычным логическим различием между категорическими и гипотетическими высказываниями.

Например, высказывания «солнце светит», «земля прогревается» первичны, а высказывание «если солнце светит, то земля прогревается» вторично. Сказать «солнце светит» – все равно, что сказать «солнце есть то, что светит», это высказывание выражает отношение между двумя классами вещей: «солнце» и «то, что светит». Однако второе из приведенных выше высказываний выражает отношение зависимости между двумя первичными высказываниями: «солнце светит» и «земля прогревается». Я не хочу этим сказать, что отношение между этими высказываниями подобно отношению, существующему между выражаемыми ими фактами, т. е. отношению причинности, но лишь то, что отношение между высказываниями влечет за собой и является следствием отношения между фактами и что, с точки зрения логики, оно может считаться приемлемым представителем этого отношения.

4.4.2. Если вместо высказывания «солнце светит» мы говорим «истинно то, что солнце светит», то речь идет не о самих вещах, а о высказывании, касающемся вещей, а именно о высказывании «солнце светит». И потому такое высказывание является вторичным. Таким образом, каждое первичное высказывание порождает вторичное высказывание, а именно то, которое утверждает его истинность или ложность.

Обычно союзы *если*, *либо*, *или* указывают на вторичность высказывания, но это не обязательно. Высказывание «животные либо разумны, либо неразумны» является первичным. Его можно переформулировать в виде «Либо животные разумны, либо животные неразумны», и потому оно не выражает отношения зависимости между двумя высказываниями, соединенными во втором дизъюнктивном предложении. Союзы *либо*, *или* на самом деле не являются *критериями* природы высказывания, хотя, по стечению обстоятельств, чаще встречаются во вторичных высказываниях. Высказывание «люди если мудры, то воздержаны» дает пример такого рода. Его нельзя переформулировать в виде «если все люди мудры, то все люди воздержаны».

4.4.3. Я не планирую обсуждать достоинства и недостатки общепринятого разделения высказываний. Я просто отмечаю, что принцип, на котором зиждется описанная выше классификация, ясен, и его применение недвусмысленно, что он подразумевает реально существующее и фундаментальное различие высказываний и что он играет существенную роль в разработке общего метода рассуждения. И тот факт, что первичное высказывание можно преобразовать в форму, при которой оно становится вторичным, ни в коей мере не противоречит изложенным здесь соображениям. Ибо в таком случае во внимание принимаются не вещи, связанные в первичном высказывании, а только само высказывание, рассматриваемое как *истинное* или *ложное*.

4.4.4. В выражении как первичных, так и вторичных высказываний в этой работе будут использоваться одни и те же символы, подчиняющиеся, как выясняется, одним и тем же законам. Разница между двумя случаями коренится не в форме, а в толковании. В обоих случаях фактическое отношение, которое имеет целью выразить высказывание, будет обозначаться знаком =.

В выражении первичных высказываний так соединенные части обычно будут представлять «члены» высказывания, или, точнее, его субъект и предикат.

4.4.5. *О выводе общего метода, основанного на перечислении возможных вариантов, для выражения любого класса или совокупности вещей, который мог бы являться «членом» Первичного Высказывания.*

Во-первых, если подлежащий выражению класс или совокупность вещей определен только названиями или качествами, общими для всех составляющих его объектов, то его выражение будет состоять из единственного члена, в котором символы, выражающие эти названия или качества, будут комбинироваться без соединительного знака, как в алгебраическом процессе умножения. Так, если x представляет непрозрачные вещества, y – отполированные вещества, а z – камни, то мы будем иметь:

xyz = непрозрачные отполированные камни;

$xy(1 - z)$ = непрозрачные отполированные вещества, не являющиеся камнями;

$x(1 - y)(1 - z)$ = непрозрачные вещества, не отполированные и не являющиеся камнями

и т. д. для других комбинаций. Заметим, что каждое из этих выражений удовлетворяет тому же закону двойственности, что и отдельные символы, которые оно содержит. Так,

$$xyz \times xyz = xyz;$$

$$xy(1 - z) \times xy(1 - z) = xy(1 - z)$$

и т. д. Любой такой член мы будем называть «классовым членом», потому что он выражает класс вещей посредством общих свойств или названий отдельных элементов такого класса.

Во-вторых, если говорить о совокупности вещей, различные части которой определены с помощью различных свойств, названий или атрибутов, то выражения для этих отдельных частей должны быть образованы независимо, а затем связаны знаком +. Но если совокупность, о которой мы хотим вести речь, была образована исключением из некоторой более широкой совокупности определенной части ее элементов, то символическому выражению, описывающему исключенную часть, должен предшествовать знак –. По поводу правильного использования этих символов можно сделать дополнительные замечания.

4.4.6. Вообще говоря, символ + эквивалентен союзам «и», «или», а символ – эквивалентен предлогу «кроме». Из союзов «и» и «или» первый обычно используется, когда описываемая совокупность образует субъект, а второй – когда она образует предикат высказывания. Высказывание «ученый *и* умудренный человек желают счастья» может служить примером одного случая. «Вещи, обладающие полезностью, *либо* доставляют удовольствие, *либо* предотвращают боль» – пример второго. Всякий раз, как выражение, включающее эти союзы, представляет собой первичное высказывание, очень

важно знать, подразумеваются ли группы или классы, разделенные ими, совершенно отличными друг от друга и взаимно исключающими или нет. Выражение «ученые и умудренные люди» включает или исключает тех, кто является тем и другим? Выражение «либо доставляют удовольствие, либо предотвращают боль» включает или исключает вещи, обладающие обоими качествами? Мне кажется, что при строгом понимании союзы «и», «или» действительно обладают свойством разделения или исключения, о котором идет речь; что формула «все x являются либо y, либо z» при строгом толковании означает «все x являются либо y, но не z, либо z, но не y». Но в то же время следует признать, что jus et norma loquendi[1] склоняют нас отдать предпочтение противоположному толкованию. Выражение «либо y, либо z», как правило, понимают так, что включены вещи, являющиеся одновременно y и z, а равно вещи, являющиеся одним из них, но не являющиеся другим. Вспоминая, однако, что символ + обладает способностью разделять, которая и была предметом обсуждения, мы должны разложить любое представленное нам дизъюнктивное выражение на элементы, действительно мысленно разделенные, а затем соединить соответствующие им выражения символом +.

И таким образом, в зависимости от подразумеваемого истолкования выражение «вещи, которые являются либо x, либо y» будет иметь два разных символических эквивалента. Если мы имеем в виду «вещи, которые являются x, но не являются y, или являются y, но не являются x», то выражение будет иметь вид $x(1 - y) + y(1 - x)$, где символ x означает все x, а y – все y. Если, однако, имеются в виду «вещи, которые либо являются x, либо если не x, то y», то выражение будет иметь вид $x + y(1 - x)$. Это выражение предполагает допустимость вещей, которые являются x и y одновременно. Более полно его можно было бы записать в форме $xy + x(1 - y) + y(1 - x)$, но это выражение после сложения первых двух членов просто сводится к первому.

Заметим, что приведенные выше выражения удовлетворяют фундаментальному закону двойственности. Таким образом, мы имеем:

$$\{x(1 - y) + y(1 - x)\}^2 = x(1 - y) + y(1 - x),$$
$$\{x + y(1 - x)\}^2 = x + y(1 - x).$$

Ниже станет ясно, что это просто конкретное проявление общего закона выражений, представляющих «классы или совокупности вещей». [Примечание редактора: в оригинале второе равенство записано в виде $\{x + (1 - x)\}^2 = x + y(1 - x)$, что, очевидно, является ошибкой.]

4.4.7. Результаты этих изысканий можно облечь в следующее правило формирования выражений.

Правило. *Выражать простые названия или качества символами x, y, z и т. д., их противоположности – символами $1 - x$, $1 - y$, $1 - z$ и т. д., классы вещей, определенные общими названиями или качествами, – путем соединения соответствующих символов, как при умножении, совокупности вещей, состоящие из различных частей, – путем соединения выражений этих частей знаком +.*

---

[1] Правила и нормы речи. – *Прим. перев.*

В частности, будем считать, что фразе «либо x, либо y» соответствует выражение $x(1-y)+y(1-x)$*, когда классы, обозначенные символами x и y, взаимно исключающие, и выражение* $x+y(1-x)$*, когда они не взаимно исключающие. Аналогично фразе «либо x, либо y, либо z» соответствует выражение* $x(1-y)(1-z)+y(1-x)(1-z)+z(1-x)(1-y)$*, когда классы, обозначенные символами x, y и z, взаимно исключающие, выражение* $x+y(1-x)+z(1-x)(1-y)$*, когда они не взаимно исключающие, и т. д.*

4.4.8. На этом правиле формирования выражений основано обратное правило толкования. Оба они будут проиллюстрированы, хочется надеяться, с достаточно полнотой в следующих примерах. Опуская для краткости универсальные субъекты «вещи» или «существа», положим x = твердый, y = упругий, z = металлы; тогда получим следующие результаты:

«Неупругие металлы» выражается в виде $z(1-y)$;

«Упругие вещества и неупругие металлы» – в виде $y+z(1-y)$;

«Твердые вещества, кроме металлов» – в виде $x-z$;

«Металлические вещества, кроме тех,
что не являются ни твердыми, упругими» –

$z-z(1-x)(1-y)$ или $z\{1-(1-x)(1-y)\}$.

В последнем примере мы на самом деле выразили фразу «металлы, кроме нетвердых, неупругих металлов». Союзы между прилагательными обычно избыточны, а потому не должны выражаться символически.

Таким образом, «металлы твердые и упругие» эквивалентно «твердые упругие металлы» и выражается в виде xyz.

Возьмем далее выражение «твердые вещества, кроме тех, что являются металлическими и неупругими, и тех, что являются упругими и неметаллическими». Здесь слово *тех* означает твердые вещества, так что выражение на самом деле означает *твердые вещества, кроме металлических неупругих твердых веществ и неметаллических упругих твердых веществ*; слово *кроме* распространяется на оба следующих за ним класса. Полное выражение имеет вид $x-\{xz(1-y)+xy(1-z)\}$ или $x-xz(1-y)-xy(1-z)$.

4.5. О фундаментальных принципах символического рассуждения и о разложении выражений, содержащих логические символы...

4.5.8. *Определение.* Любое алгебраическое выражение, содержащее символ x, называется функцией от x и может быть представлено сокращенной общей формой $f(x)$. Любое выражение, содержащее два символа x и y, аналогично называется функцией от x и y и может быть представлено сокращенной общей формой $f(x, y)$ и т. д. для прочих случаев. <...>

4.5.9. *Определение.* Любая функция $f(x)$, в которой x – логический символ или символ количества, принимающий только значения 0 и 1, называется разложенной, если она приведена к форме $ax + b(1 - x)$, где a и b определены так, чтобы сделать результат эквивалентным функции, из которой эта форма выведена.

Это определение предполагает, что любую функцию $f(x)$ можно представить в указанном виде. Это предположение доказано в следующем Предложении.

4.5.10. ***О разложении любой функции f(x), в которой x – логический символ.*** В силу принципа, установленного в этой главе, можно трактовать x как количественный символ, принимающий только значения 0 и 1.

Предположим тогда, что $f(x) = ax + b(1 - x)$; полагая $x = 1$, получаем $f(1) = a$. Полагая в том же равенстве $x = 0$, получаем $f(0) = b$. Тем самым значения a и b определены, и, подставляя их в исходное равенство, получаем

$$f(x) = f(1)x + f(0)(1 - x),$$

т. е. искомое разложение. Правая часть равенства адекватно представляет функцию $f(x)$, какую бы форму она ни принимала. <…>

5 Математические проблемы (1900)

Давид Гильберт[1]

Ничто так не важно для прогресса науки, как умение задать вопрос. Великий немецкий математик Давид Гильберт (1862–1943) из Гёттингенского университета воспользовался Международным конгрессом математиков на рубеже XX века, чтобы в общем виде охарактеризовать математические проблемы и их решения и предложить коллегам список из 23 нерешенных проблем. Некоторые проблемы были решены довольно быстро, но на решение десятой понадобилось 70 лет, а решение было дано в форме доказательства рекурсивной неразрешимости – в 1900-х годах Гильберт даже слов таких не знал. Восьмая проблема, гипотеза Римана, до сих пор не решена, а некоторые другие проблемы, формулировка которых требует специальных знаний, решены частично.

В 2000 году математический институт Клея добавил в список Гильберта семь проблем – задач тысячелетия, – назначив за решение каждой приз в миллион долларов (Clay Mathematics Institute, 2000). На сегодняшний день решена только одна – гипотеза Пуанкаре. Гипотеза Римана вошла в список тысячелетия, равно как и проблема равенства $\mathcal{P} = \mathcal{NP}$ (глава 34).

В 1900 году еще не было точного понятия алгоритма, а уж тем более никто не знал, что такое рекурсивная неразрешимость. И тем не менее в постановке Гильберта имеется множество намеков на грядущее: ссылки на процессы с «конечным числом шагов» и на «невозможность найти решение при заданных гипотезах или в предполагаемом смысле», в связи с чем он приводит в пример античное доказательство иррациональности числа $\sqrt{2}$.

---

[1] Перевод текста Гильберта М. Г. Шестопал и А. В. Дорофеевой. Воспроизведено по изданию: Проблемы Гильберта: сб. / под общ. ред. П. С. Александрова. М.: Наука, 1969.

В конце XIX века логическая революция, предвестником которой стали работы Буля, развилась в новую дисциплину, метаматематику, с амбициозной программой – построить твердый математический фундамент, на котором могла бы покоиться сама математика. Готлоб Фреге (Frege 1879) предложил первую строгую аксиоматику современной логики, опубликовав сочинение «Begriffsschrift» (Исчисление понятий), а Уайтхед и Рассел (Whitehead and Russell 1910) опубликовали первую часть толстенной «Principia Mathematica» (Основания математики) – попытки всеобъемлющей формализации всей математики. Казалось, осуществление мечты Лейбница не за горами.

Но стоило попыткам формализовать математику стать более настойчивыми, как начали обнаруживаться противоречия и парадоксы. Гильберт, обеспокоенный шаткостью почвы, на которой утвердился его оптимизм, призвал математическое сообщество «раз и навсегда избавиться от подобных фундаментальных вопросов математики» (Hilbert 1928; Van Heijenoort 1967). Этот призыв получил название программы Гильберта, и лишь после того, как цель была поставлена, стало возможно заподозрить, что она может оказаться недостижимой.

В обращении Гильберта 1900 года не было мыслей о такой невозможности. Его тон окрыляющий и обнадеживающий. Любую проблему можно решить, так или иначе. Не существует никакого *ignorabimus* (буквально «нам не дано знать» – т. е. не пристало сдаваться и отказываться от поиска ответа).

На самом деле дух этого обращения отражал состояние культуры в то время. Рубеж XX века был временем расцвета всеобщего позитивизма. Войны удалось ограничить и сделать контролируемыми. Промышленная революция принесла прогресс и процветание, по крайней мере западному миру. Пятьдесят миллионов человек посетили Всемирную выставку в Париже, чтобы своими глазами увидеть чудеса современного мира. Романтические, кружащие голову произведения искусства в стиле ар-нуво воздействовали на настроение общества, говоря, что жизнь спокойна, прекрасна в своем слиянии с природой, а дальше будет только лучше.

Мы включили в книгу отрывки из вводной части обращения Гильберта, а из 23 проблем – только вторую и десятую. В десятой проблеме требуется определить, существует ли универсальный алгоритм решения диофантовых уравнений в целых числах. Так называются уравнения вида $p(x_1, \ldots, x_k) = 0$, где p – многочлен с целыми коэффициентами, например $17x_1x_2^2 - 13x_1^2x_2 + 3x_1 + 11 = 0$. Требование целочисленности $x_1, \ldots, x_k$ превращает алгебраическую задачу в комбинаторную. В результате растянувшихся на несколько десятилетий усилий Мартина Дэвиса, Хилари Патнэма, Джулии Робинсон и Юрия Матиясевича удалось доказать, что общая задача – определить, имеет ли решение такое уравнение, – неразрешима. Последний удар теоретико-числовым оружием нанес Матиясевич в 1970 году (Matiyasevich, 1993).

Через 15 лет после обращения Гильберта Европа погрузилась в кровавую, кажущуюся бессмысленной и бесконечной войну. Литература стала ироничной и циничной, в изобразительных искусствах преобладали мрачные темы насилия. И словно бы отражая этот сдвиг в сторону пессимизма, между двумя мировыми войнами Гёдель и Тьюринг продемонстрировали неожиданные следствия из символической формализации идеи конечного

процесса с дискретными шагами. Казалось, что у самих формальных систем существуют пределы.

В 1920-х годах здоровье Гильберта пошатнулось, а великая математическая школа, сложившаяся вокруг него в Гёттингене, распалась после изгнания нацистами профессоров-евреев. На могиле Гильберта выбиты (по-немецки) его смелые слова: «Мы должны знать. Мы будем знать». Он произнес их в 1930 году, за день до того, как Курт Гёдель объявил о результатах череды исследований, вызванных к жизни второй проблемой Гильберта, – о независимости и непротиворечивости аксиом. Гёдель доказал, что в любой непротиворечивой логической системе, допускающей рекурсивную аксиоматизацию, существуют истинные высказывания, которые невозможно доказать, и тем самым поставил крест на любых попытках дать простой ответ на вопрос о том, что может быть известно.

Но, закрыв одну дверь, Гёдель и Тьюринг открыли бесчисленное множество других. Гёделево доказательство того, что математика никогда не станет единой замкнутой системой, означало также, что ее возможности безграничны. Тьюринг определил, что такое алгоритм, – с целью установить, чего алгоритмы не могут сделать. И при этом положил начало научному анализу алгоритмов, теоретическому и практическому, проложив дорогу в сегодняшний мир компьютерных вычислений.

Кто из нас не хотел бы приоткрыть завесу, за которой скрыто наше будущее, чтобы хоть одним взглядом проникнуть в предстоящие успехи нашего знания и тайны его развития в ближайшие столетия? Каковы будут те особенные цели, которые поставят себе ведущие математические умы ближайшего поколения? Какие новые методы и новые факты будут открыты в новом столетии на широком и богатом поле математической мысли?

История учит, что развитие науки протекает непрерывно. Мы знаем, что каждый век имеет свои проблемы, которые последующая эпоха или решает, или отодвигает в сторону как бесплодные, чтобы заменить их новыми. Чтобы представить себе возможный характер развития математического знания в ближайшем будущем, мы должны перебрать в нашем воображении вопросы, которые еще остаются открытыми, обозреть проблемы, которые ставит современная наука и решения которых мы ждем от будущего. Такой обзор проблем кажется мне сегодня, на рубеже нового столетия, особенно своевременным. Ведь большие даты не только заставляют нас оглянуться на прошедшее, но и направляют нашу мысль в неизвестное будущее.

Невозможно отрицать глубокое значение, какое имеют определенные проблемы для продвижения математической науки вообще, и важную роль, которую они играют в работе отдельного исследователя. Всякая научная область жизнеспособна, пока в ней избыток новых проблем. Недостаток новых проблем означает отмирание или прекращение самостоятельного развития. Как вообще каждое человеческое начинание связано с той или иной целью, так и математическое творчество связано с постановкой проблем. Сила исследователя познается в решении проблем: он находит новые методы, новые точки зрения, он открывает более широкие и свободные горизонты.

Трудно, а часто и невозможно заранее правильно оценить значение отдельной задачи; ведь в конечном счете ее ценность определится пользой, которую она принесет науке. Отсюда возникает вопрос: существуют ли общие признаки, которые характеризуют хорошую математическую проблему? Один старый французский математик сказал: «Математическую теорию можно считать совершенной только тогда, когда ты сделал ее настолько ясной, что берешься изложить ее содержание первому встречному». Это требование ясности и легкой доступности, которое здесь так резко ставится в отношении математической теории, я бы поставил еще резче в отношении математической проблемы, если она претендует на совершенство; ведь ясность и легкая доступность нас привлекают, а усложненность и запутанность отпугивают.

Математическая проблема, далее, должна быть настолько трудной, чтобы нас привлекать, и в то же время не совсем недоступной, чтобы не делать безнадежными наши усилия; она должна быть путеводным знаком на запутанных тропах, ведущих к сокрытым истинам; и она затем должна награждать нас радостью найденного решения. Математики прошлого столетия со страстным рвением отдавались решению отдельных трудных задач; они знали цену трудной задаче. Я напомню только поставленную Иоганном Бернулли *задачу о линии быстрейшего падения.* «Как показывает опыт, – говорит Бернулли, оповещая о своей задаче, – ничто с такой силой не побуждает высокие умы к работе над обогащением знания, как постановка трудной и в то же время полезной задачи». И поэтому он надеется заслужить благодарность математического мира, если он, – следуя примеру таких мужей, как Мерсенн, Паскаль, Ферма, Вивиани и другие, которые (до него) поступали так же, – предложит задачу выдающимся аналитикам своего времени, чтобы они могли на ней, как на пробном камне, испытать достоинства своих методов и измерить свои силы. Этой задаче Бернулли и другим аналогичным задачам обязано своим зарождением вариационное исчисление.

Известно утверждение Ферма о том, что диофантово уравнение

$$x^n + y^n = z^n$$

неразрешимо в целых числах x, y, z, если не считать известных очевидных исключений. *Проблема доказательства этой неразрешимости* являет разительный пример того, какое побуждающее влияние на науку может оказать специальная и на первый взгляд малозначительная проблема. Ибо, побужденный задачей Ферма, Куммер пришел к введению идеальных чисел и к открытию теоремы об однозначном разложении чисел в круговых полях на идеальные простые множители – теоремы, которая теперь, благодаря обобщениям на любую алгебраическую числовую область, полученным Дедекиндом и Кронекером, является центральной в современной теории чисел и значение которой выходит далеко за пределы теории чисел в область алгебры и теории функций.

Напомню еще об одной интересной проблеме – *задаче трех тел.* То обстоятельство, что Пуанкаре предпринял новое рассмотрение и значительно продвинул эту трудную задачу, привело к плодотворным методам и далеко идущим принципам, введенным этим ученым в небесную механику,

методам и принципам, которые сейчас признаются и применяются также и в практической астрономии.

Обе упомянутые проблемы – проблема Ферма и проблема трех тел – являются в нашем запасе проблем как бы противоположными полюсами: первая представляет свободное достижение чистого разума, принадлежащее области абстрактной теории чисел, вторая выдвинута астрономией и необходима для познания простейших основных явлений природы.

Часто, однако, случается, что одна и та же специальная проблема появляется в весьма различных областях математики. Так, *проблема о кратчайшей линии* играет важную историческую и принципиальную роль одновременно в основаниях геометрии, в теории кривых и поверхностей, в механике и в вариационном исчислении. А как убедительно демонстрирует Ф. Клейн в своей книге об икосаэдре, задача о правильных многогранниках имеет важное значение одновременно для элементарной геометрии, теории групп, теории алгебраических и теории линейных дифференциальных уравнений. <…>

Остановимся еще кратко на вопросе о том, каковы могут быть общие требования, которые мы вправе предъявить к решению математической проблемы. Я имею в виду прежде всего требования, благодаря которым удается убедиться в правильности ответа с помощью конечного числа заключений, и притом на основании конечного числа предпосылок, которые кладутся в основу каждой задачи и которые должны быть в каждом случае точно сформулированы. Это требование логической дедукции с помощью конечного числа заключений есть не что иное, как требование строгости проведения доказательств. Действительно, требование строгости, которое в математике уже вошло в поговорку, соответствует общей философской потребности нашего разума; с другой стороны, только выполнение этого требования приводит к выявлению полного значения существа задачи и ее плодотворности. Новая задача, особенно если она вызвана к жизни явлениями внешнего мира, подобна молодому побегу, который может расти и приносить плоды, лишь если он будет заботливо и по строгим правилам искусства садоводства взращиваться на старом стволе – твердой основе нашего математического знания. <…>

Сделаем еще несколько замечаний относительно трудностей, которые могут представлять математические проблемы, и о преодолении этих трудностей.

Если нам не удается найти решение математической проблемы, то часто причина этого заключается в том, что мы не овладели еще достаточно общей точкой зрения, с которой рассматриваемая проблема представляется лишь отдельным звеном в цепи родственных проблем. Отыскав эту точку зрения, мы часто не только делаем более доступной для исследования данную проблему, но и овладеваем методом, применимым и к родственным проблемам. Примерами могут служить введенное Коши в теорию определенного интеграла интегрирование по криволинейному пути и установление Куммером понятия идеала в теории чисел. Этот путь нахождения общих методов наиболее удобный и надежный, ибо если ищут общие методы, не имея в виду какую-нибудь определенную задачу, то эти поиски, по большей части, напрасны.

При исследовании математических проблем специализация играет, как я полагаю, еще более важную роль, чем обобщение. Возможно, что в большинстве случаев, когда мы напрасно ищем ответа на вопрос, причина нашей неудачи заключается в том, что еще не разрешены или не полностью решены более простые и легкие проблемы, чем данная. Тогда все дело заключается в том, чтобы найти эти более легкие проблемы и осуществить их решение наиболее совершенными средствами, при помощи понятий, поддающихся обобщению. Это правило является одним из самых мощных рычагов для преодоления математических трудностей, и мне кажется, что в большинстве случаев этот рычаг и приводят в действие, подчас бессознательно.

Вместе с тем бывает и так, что мы добиваемся ответа при недостаточных предпосылках или идя в неправильном направлении, и вследствие этого не достигаем цели. Тогда возникает задача доказать неразрешимость данной проблемы при принятых предпосылках и выбранном направлении. Такие доказательства невозможности проводились еще старыми математиками, например когда они обнаруживали, что отношение гипотенузы равнобедренного прямоугольного треугольника к его катету есть иррациональное число. В новейшей математике доказательства невозможности решений определенных проблем играют выдающуюся роль; там мы констатируем, что такие старые и трудные проблемы, как доказательство аксиомы о параллельных, как квадратура круга или решение уравнения пятой степени в радикалах, получили все же строгое, вполне удовлетворяющее нас решение, хотя и в другом направлении, чем то, которое сначала предполагалось. Этот удивительный факт наряду с другими философскими основаниями создает у нас уверенность, которую разделяет, несомненно, каждый математик, но которую до сих пор никто не подтвердил доказательством, – уверенность в том, что каждая определенная математическая проблема непременно должна быть доступна строгому решению, или в том смысле, что удается получить ответ на поставленный вопрос, или же в том смысле, что будет установлена невозможность ее решения и вместе с тем доказана неизбежность неудачи всех попыток ее решить. Представим себе какую-либо нерешенную проблему, скажем вопрос об иррациональности константы *C* Эйлера–Маскерони или вопрос о существовании бесконечного числа простых чисел вида $2^n + 1$. Как ни недоступными представляются нам эти проблемы и как ни беспомощно мы стоим сейчас перед ними, мы имеем все же твердое убеждение, что их решение с помощью конечного числа логических заключений все же должно удасться.

Является ли эта аксиома разрешимости каждой данной проблемы характерной особенностью только математического мышления или, быть может, имеет место общий, относящийся к внутренней сущности нашего разума закон, по которому все вопросы, которые он ставит, способны быть им разрешены? Встречаются ведь в других областях знания старые проблемы, которые были самым удовлетворительным образом и к величайшей пользе науки разрешены путем доказательства невозможности их решения. Я вспоминаю проблему о *perpetuum mobile* (вечный двигатель). После напрасных попыток конструирования вечного двигателя стали, наоборот, исследовать соотношения, которые должны существовать между силами природы, в предполо-

жении, что *perpetuum mobile* невозможно. И эта постановка обратной задачи привела к открытию закона сохранения энергии, из которого и вытекает невозможность *perpetuum mobile* в первоначальном понимании его смысла.

Это убеждение в разрешимости каждой математической проблемы является для нас большим подспорьем в работе; мы слышим внутри себя постоянный призыв: вот проблема, ищи решение. Ты можешь найти его с помощью чистого мышления; ибо в математике не существует *ignorabimus*. <...>

2. Непротиворечивость арифметических аксиом

Когда речь идет о том, чтобы исследовать основания какой-нибудь науки, следует установить систему аксиом, содержащих точное и полное описание тех соотношений, которые существуют между элементарными понятиями этой науки. Эти аксиомы являются одновременно определениями этих элементарных понятий, и мы считаем правильными только такие высказывания в области науки, основания которой исследуем, какие получаются из установленных аксиом с помощью конечного числа логических умозаключений. При более близком рассмотрении возникает вопрос: *не являются ли некоторые из этих аксиом зависящими друг от друга, не содержат ли некоторые из этих аксиом общие части, которые следовало бы изъять, если ставить задачу об установлении системы аксиом, полностью независимых друг от друга?*

Из многочисленных вопросов, которые могут быть поставлены относительно системы аксиом, мне хотелось бы прежде всего указать на важнейшую проблему, именно на доказательство того, что система аксиом непротиворечива, т. е. *что на основании этих аксиом никогда нельзя с помощью конечного числа логических умозаключений получить результаты, противоречащие друг другу*. <...>

10. Задача о разрешимости диофантова уравнения

Пусть задано диофантово уравнение с произвольными неизвестными и целыми рациональными числовыми коэффициентами. *Указать способ, при помощи которого возможно после конечного числа операций установить, разрешимо ли это уравнение в рациональных числах.* <...>

6 О вычислимых числах с приложением к проблеме разрешения (1936)

Алан Мэтисон Тьюринг

Проблема разрешения, или по-немецки «Entscheidungsproblem», заключается в решении вопроса о том, доказуемо ли любое математическое утверждение. В 1928 году логические основания математики были уже достаточно прочны, чтобы Давид Гильберт и его ученик Вильгельм Аккерман могли говорить, что рассматриваемая аксиоматическая система является исчислением предикатов (ниже она называется «функциональным исчислением K»). Требовалось найти процедуру, которая могла бы по существу ответить на все математические вопросы и подвести итог под программой Гильберта. Различные исследователи предлагали процедуры решения для частей исчисления предикатов, но поначалу не было никакой серьезной работы в направлении доказательства невозможности – отчасти потому, что понятие «процедуры решения» было всего лишь интуитивным. Представление о том, что может не существовать никакой процедуры, требовало описания класса всех процедур.

В 1930 году Курт Гёдель объявил о своем революционном отрицательном результате (Gödel 1931), состоящем в том, что в любой системе типа описанной в «Основаниях математики» (Whitehead and Russell 1910) или в исчислении предикатов, некоторые предложения не могут быть ни доказаны, ни опровергнуты (если только сама система не является противоречивой, и тогда возможно доказать любое утверждение). Для этого Гёдель воспользовался

вариантом диагонального метода, использованного Кантором в 1891 году для доказательства несчетности множества вещественных чисел (его «неперечислимости» в терминологии Тьюринга). Другой новаторский результат Гёделя состоит в кодировании конечной строки символов (например, формулы исчисления предикатов) целым положительным числом; для этого он использовал для кодирования строки длины *k* произведение первых *k* простых чисел, в котором показатель степени *k*-го простого числа равен цифре, кодирующей *k*-й символ строки.

Результат Гёделя был ошеломляющим, но проблему разрешимости он не решил. (Хотя как объясняет Тьюринг на стр. 92, если бы Гёдель доказал утверждение, противоположное тому, которое он доказал на самом деле, то проблема разрешимости была бы решена положительно.) Почти одновременно Алонзо Чёрч и Тьюринг формализовали понятие «алгоритма» совершенно различными, но эквивалентными способами. Чёрч (Church 1936b) был первым со своим лямбда-исчислением, а вскоре после него Тьюринг сделал это с помощью того, что мы теперь называем машинами Тьюринга. Обоим приписывается честь доказательства неразрешимости Entscheidungsproblem.

Оба автора столкнулись с одной и той же трудностью: доказать свою теорему и убедить читателя в том, что его класс вычислительных процедур достаточно велик, чтобы охватить все вычислительные процессы, которые можно себе представить. Из-за мудрености лямбда-исчисления работа Чёрча не достигла второй цели – и это несмотря на то, что впоследствии лямбда-исчисление легло в основу функционального программирования (глава 21).

Таким образом, машины Тьюринга должны были быть достаточно простыми, чтобы можно было доказать, чего они не могут сделать, и в то же время достаточно мощными, чтобы можно было продемонстрировать возможность выполнения ими всего, что разумный человек решился бы назвать вычислением. Мы включили части обоих доказательств, но опустили весь код. Как оказалось, редукционизм машин Тьюринга «запиши символ в ячейку и перейди к соседней ячейке» предвидел математик Эмиль Пост, который, однако, не предпринял дальнейших шагов, ведущих к доказательству. Работа Поста, которая ссылается на статью (Church 1936b), но в свое время не была опубликована, впервые появилась в сборнике (Davis 1965, стр. 289–291).

Будет полезно дать пояснения к языку Тьюринга. Его машины (он использовал термин «computer» только для обозначения человека, выполняющего вычисления) вначале получают пустую ленту и приступают к работе. Они печатают нули и единицы (он называл их «цифрами» – figure), быть может, вперемежку с другими символами. При чтении потенциально бесконечного ряда цифр предполагается, что в начале стоит десятичная точка, так что он представляет вещественное число от 0 до 1. Некоторые из таких вещественных чисел вычислимы, а некоторые – нет (потому что множество машин счетно, а множество вещественных чисел на отрезке [0,1] нет). Вычислимое число можно рассматривать как способ представления вычислимого множества неотрицательных целых чисел (вычисление *i*-го бита вещественного числа, принадлежащего отрезку [0,1], эквивалентно определению того, принадлежит ли целое число *i* множеству, соответствующему этому вещественному

числу) или как вычислимую функцию, отображающую множество целых чисел в себя с помощью другой системы кодирования, и т. д.

То, что Тьюринг называет m-конфигурацией, сегодня мы назвали бы состоянием машины, обычно оно обозначается каким-то вариантом буквы q. С другой стороны, «конфигурация» состоит из состояния и сканируемого символа, т. е. всего того, что определяет следующий переход, а «полная конфигурация» включает также полное содержимое ленты и положение головки сканирования – это все, что можно сказать о машине в конкретный момент ее работы.

«Циклической» называется машина, печатающая только конечную последовательность цифр, а «свободной от циклов» – машина, печатающая бесконечную последовательность цифр.

«Универсальной» машиной Тьюринг называет машину, которая может смоделировать любую другую. Чтобы подать машину на вход универсальной машине, Тьюрингу нужен был способ кодирования произвольных машин над фиксированным алфавитом. Он представлял состояние q_i как DA^i, а символ S_j как DC^j. Строку, являющуюся конкатенацией так закодированных четверок, Тьюринг называл «стандартным описанием» (СО), это первое отчетливое упоминание об экземпляре хранимой программы; для хранения программы можно было использовать память общего назначения, так что между программой и данными не было существенного различия. Заменяя немногие индивидуальные символы в этом универсальном коде десятичными цифрами, Тьюринг получал численное значение каждой машины, которое он называл ее «числовым описанием» (ЧО). Эта техника кодирования строк числами отличается от придуманной Гёделем. Из существования универсальной машины Тьюринг с помощью диагонального метода смог вывести, что никакая машина не способна надежно отличить циклическую машину от машины, свободной от циклов. Еще два шага – и неразрешимость Entscheidungsproblem была доказана. (Мы опустили большую часть построения, которое представляло машинное вычисление логической формулой, – оно содержало ошибку и впоследствии было исправлено [Turing, 1938].)

Жизнь Алана Мэтисона Тьюринга (1912–1954) подробно документирована в биографии, написанной Эндрю Ходжесом (Hodges 1983), по которой был снят фильм «Игра в имитацию». Тьюринг написал эту статью через два года после получения диплома с отличием первой степени по математике в Кембридже, а затем работал над докторской диссертацией под руководством Чёрча в Принстоне. Его математическая карьера резко сменила направление, когда пришлось работать над вскрытием кодов во время Второй мировой войны. После изобличения в гомосексуализме он был лишен допуска и арестован. Согласившись на химическую кастрацию в качестве альтернативы тюремному заключению, он умер, отравившись цианидом, – это выглядело как самоубийство, хотя есть некоторые свидетельства в пользу того, что смерть могла быть случайной. Скончавшийся, когда ему был всего 41 год, он, по словам популяризатора науки Чета Раймо (Chet Raymo 1996), был «гигантом логики, принесенным в жертву иррациональному». Королева Елизавета даровала Тьюрингу посмертное помилование в 2014 году, изъяв из его дела

обвинения в непристойном поведении, а в 2019 году Банк Англии объявил, что его изображение появится на купюре достоинством 50 фунтов.

«Вычислимые» числа можно вкратце описать как вещественные числа, чье десятичное выражение можно вычислить конечными средствами. Хотя, на первый взгляд, предметом этой статьи являются вычислимые числа, почти так же легко определить и исследовать вычислимые функции целой переменной или вещественной, или вычислимой переменной, вычислимые предикаты и т. д. Фундаментальные проблемы, однако, во всех случаях одинаковы, и я выбрал для явного рассмотрения вычислимые числа, поскольку необходимая для этого техника наименее громоздкая. Я надеюсь вскоре представить отчет о связях между вычислимыми числами, функциями и всем прочим. В него войдет разработка теории функций вещественной переменной, выраженная в терминах вычислимых чисел. Согласно моему определению, число является вычислимым, если его десятичное представление можно записать с помощью машины.

В § 6.9 и § 6.10 я приведу некоторые аргументы, имеющие целью показать, что вычислимые числа включают все числа, которые можно было бы естественно назвать вычислимыми. В частности, я покажу, что вычислимы некоторые большие классы чисел, например вещественные части всех алгебраических чисел, вещественные части нулей бесселевых функций, числа π, e и т. д. Однако не все определимые числа являются вычислимыми, и будет приведен пример определимого, но не вычислимого числа.

Хотя класс вычислимых чисел очень велик и во многих отношениях похож на класс вещественных чисел, он тем менее является перечислимым. В § 6.8 я рассмотрю некоторые аргументы, которые, по видимости, доказывают противное. Путем корректного применения одного из этих аргументов можно прийти к выводам, внешне похожим на умозаключения Гёделя (Gödel 1931). У этих результатов есть ценные применения. В частности, показано (§ 6.11), что проблема разрешимости Гильберта не может иметь решения.

В недавней статье Алонзо Чёрч (Church 1936b) ввел идею «эффективной вычислимости», которая эквивалентна моей «вычислимости», но определена совершенно иначе. Чёрч пришел к аналогичным выводам относительно проблемы разрешимости (Church 1936a). Набросок доказательства эквивалентности «вычислимости» и «эффективной вычислимости» включен в приложение к настоящей статье. [Примечание редактора: приложение здесь опущено.]

6.1. Вычислительные машины

Мы сказали, что вычислимыми называются числа, десятичное представление которых можно вычислить конечными средствами. Тут необходимо более точное определение. Никакой попытки обосновать данные определения не будет предпринято до § 6.9. А пока я скажу только, что обоснование связано с тем, что память человека по необходимости ограничена.

Мы можем сравнить человека, занятого вычислением вещественного числа, с машиной, которая может находиться лишь в конечном числе условий q_1, q_2, ..., q_R, – которые мы будем называть «m-конфигурациями». Машина снабжена «лентой» (аналог бумаги), которая проходит через нее и разделена на секции (называемые «ячейками»), в каждой из которых может находиться «символ». В любой момент времени существует только одна ячейка, скажем r-я с символом $\mathfrak{S}(r)$, которая находится «внутри машины». Мы можем назвать эту ячейку «сканируемой ячейкой». Символ в сканируемой ячейке можно назвать «сканируемым символом». Сканируемый символ – единственный, который машина, так сказать, «непосредственно видит». Однако, изменив свою m-конфигурацию, машина может запомнить некоторые символы, которые она «видела» (сканировала) раньше. Возможное поведение машины в любой момент времени определяется m-конфигурацией q_n и сканируемым символом $\mathfrak{S}(r)$. Эту пару q_n, $\mathfrak{S}(r)$ будем называть «конфигурацией»; таким образом, конфигурация определяет возможное поведение машины. В некоторых конфигурациях, где сканируемая ячейка пуста (т. е. не содержит никакого символа), машина записывает новый символ в сканируемую ячейку; в других конфигурациях она стирает сканируемый символ. Машина может также изменять сканируемую ячейку, но только сдвигая ее на одну позицию вправо или влево. В дополнение к любой из этих операций можно изменять m-конфигурацию. Некоторые записанные символы образуют последовательность цифр, являющуюся десятичным представлением вещественного числа, которое вычисляется. Остальные – просто черновики «в помощь памяти». Только эти черновики и подлежат стиранию.

Я утверждаю, что эти операции включают все, что используется при вычислении числа. Защитить это утверждение будет проще, если читателю знакома теория машин. Поэтому в следующем разделе я разовью эту теорию в предположении, что уже понятен смысл терминов «машина», «лента», «сканируемый» и т. д.

6.2. Определения

6.2.1. Автоматические машины. Если на каждом этапе движение машины (в смысле, определенном в § 6.1) полностью определено конфигурацией, то будем называть такую машину «автоматической машиной» (или a-машиной).

Для некоторых целей мы могли бы использовать машины (машины с выбором, или c-машины), движение которых лишь частично определяется конфигурацией (отсюда использование слова «возможное» в § 6.1). Когда такая машина достигает одной из подобных неоднозначных конфигураций, она не может продвигаться дальше, пока внешний оператор не сделает какой-то произвольный выбор. Так обстояло бы дело, если бы мы использовали машины для работы с аксиоматическими системами. В этой статье я буду иметь дело только с автоматическими машинами, а потому буду часто опускать префикс a-.

6.2.2. Вычислительные машины. Если *a*-печатает два вида символов, из которых первый вид (называемый цифрами) включает только 0 и 1 (все остальные называются символами второго рода), то такую машину будем называть вычислительной. Если машина снабжена чистой лентой и приведена в действие, находясь в начальный момент в корректной *m*-конфигурации, то подпоследовательность печатаемых ей символов, состоящая из символов первого рода, будет называться *последовательностью, вычисленной машиной*. Вещественное число, десятичная запись которого в двоичной форме получается путем дописывания десятичной точки в начало этой последовательности, называется *числом, вычисленным машиной*.

На любом этапе движения машины число в сканируемой ячейке, полная последовательность символов на ленте и *m*-конфигурация, по определению, описывают *полную конфигурацию* на этом этапе. Изменения машины и ленты при переходе от предыдущей полной конфигурации к следующей называются *тактами* машины.

6.2.3. Циклические и свободные от циклов машины. Вычислительная машина, которая никогда не записывает более чем конечное число символов первого рода, называется *циклической*. В противном случае машина называется *свободной от циклов*.

Машина будет циклической, если она достигает конфигурации, в которой невозможно совершить ни одного такта, или если она продолжает совершать такты и, возможно, печатать символы второго рода, но не может напечатать ни одного символа первого рода. Смысл термина «циклический» будет объяснен в § 6.8.

6.2.4. Вычислимые последовательности и числа. Последовательность называется вычислимой, если ее можно вычислить машиной, свободной от циклов. Число называется вычислимым, если оно отличается от числа, вычисляемого свободной от циклов машиной, на целое число.

Мы будем избегать путаницы, чаще говоря о вычислимых последовательностях, чем о вычислимых числах.

6.3. Примеры вычислительных машин

... «R» (right – вправо) означает «машина совершает такт, сканируя ячейку, находящуюся непосредственно справа от той, что она сканировала ранее». И аналогично для «L» (left – слева). «E» (erase – стирать) означает «сканируемый символ стерт». «P» означает «печатает». <...> [Примечание редактора: детали кодирования и программирования опущены.]

6.4. Сокращенные таблицы

<...>

6.5. Перечисление вычислимых последовательностей

<...> Выпишем все выражения, образованные описанным выше способом, из таблицы для машины и разделим их точками с запятой. Таким образом, мы получим полное описание машины. В этом описании заменим q_i буквой «D», за которой следует буква «A», повторенная i раз, а S_j – буквой «D», за которой следует буква «C», повторенная j раз. Это новое описание машины можно назвать стандартным описанием (СО). Оно целиком состоит из букв «A», «C», «D», «L», «R», «N» и «;».

Если, наконец, заменить «A» на «1», «C» на «2», «D» на «3», «L» на «4», «R» на «5», «N» на «6» и «;» на «7», то мы получим описание машины в виде строки арабских цифр. Целое число, представленное этой строкой, можно назвать числовым описанием (ЧО) машины. ЧО однозначно определяет СО и структуру машины. Машину, для которой ЧО равно n, можно обозначить $\mathcal{M}(n)$.

Каждой вычислимой последовательности соответствует по крайней мере одно числовое описание, тогда как отсутствию числового описания соответствует более одной вычислимой последовательности. Таким образом, вычислимые последовательности и числа перечислимы. <...>

3133253117311335311173111332253111173111133531731323253117 – числовое описание, как и 3133253117311335311173111332253111173111133531731323253117.

Число, являющееся числовым описанием машины, свободной от циклов, будем называть *удовлетворительным*. В § 6.8 показано, что не существует никакого общего процесса, определяющего, является заданное число удовлетворительным или нет.

6.6. Универсальная вычислительная машина

Можно придумать одну машину, которая сможет вычислить любую вычислимую последовательность. Если эту машину $\mathcal{U}$ снабдить лентой, в начале которой записано СО некоторой вычислительной машины $\mathcal{M}$, то $\mathcal{U}$ вычислит ту же последовательность, что и $\mathcal{M}$. <...>

6.7. Детальное описание универсальной машины

<...>

6.8. Применение диагонального процесса

Можно подумать, что аргументы, доказывающие неперечислимость вещественных чисел, смогли бы также доказать неперечислимость вычислимых чисел и последовательностей (Hobson 1921, стр. 87–88). Например, можно

было бы подумать, что предел последовательности вычислимых чисел должен быть вычислимым числом. Очевидно, что это верно только в том случае, когда последовательность вычислимых чисел определена некоторым правилом.

Или мы могли бы применить диагональный процесс. «Если вычислимые последовательности перечислимы, то обозначим a_n n-ю вычислимую последовательность, и пусть $\phi_n(m)$ – m-я цифра в a_n. Обозначим β последовательность, в которой n-й цифрой является $1 - \phi_n(n)$. Поскольку β вычислима, существует такое число K, что $1 - \phi_n(n) = \phi_K(n)$ для всех n. Положим $n = K$, получим $1 = 2\phi_K(K)$, т. е. 1 – четное число. Это невозможно. Таким образом, вычислимые последовательности неперечислимы».

Ошибочность этого доказательства заключается в предположении, что β вычислима. Это было бы так, если бы мы могли перечислить вычислимые последовательности конечными средствами, но проблема перечисления вычислимых последовательностей эквивалентна проблеме определения того, является ли заданное число ЧО свободной от циклов машины, а мы не располагаем общим процессом, который позволил бы ответить на этот вопрос за конечное число шагов. На самом деле, корректно применив диагональный процесс, мы сможем показать, что такого общего процесса вообще не существует.

Простейшее и самое прямое доказательство этого факта – показать, что если такой общий процесс существует, то существует машина, вычисляющая β. Это доказательство, хотя и совершенно корректно, обладает одним недостатком: оно может оставить читателя с ощущением «что-то здесь не так». Доказательство, которое я приведу, лишено этого недостатка и позволяет лучше понять смысл термина «свободный от циклов». Оно опирается не на построение β, а на построение последовательности β', n-я цифра которой равна $\phi_n(n)$.

Предположим, что такой процесс существует, т. е. что можно придумать машину $\mathcal{D}$ такую, что если снабдить ее СО любой вычислительной машины $\mathcal{M}$, то она проверит это СО и, если $\mathcal{M}$ циклическая, пометит его символом «u», а если свободная от циклов – то символом «s». Комбинируя машины $\mathcal{D}$ и $\mathcal{U}$, мы могли бы построить машину $\mathcal{H}$ для вычисления последовательности β'. <…>

Движение машины $\mathcal{H}$ разделено на секции. В первых $N - 1$ секциях, среди прочих действий, целые числа $1, 2, \ldots, N - 1$ записываются и проверяются машиной $\mathcal{D}$. Обнаружено, что сколько-то из этих чисел, скажем $R(N - 1)$, являются ЧО свободных от циклов машин. В N-й секции машина $\mathcal{D}$ проверяет число N. Если N удовлетворительно, т. е. является ЧО свободной от циклов машины, то $R(N) = 1 + R(N - 1)$ и вычисляются первые $R(N)$ цифр последовательности, для которой N является ЧО. $R(N)$-я цифра этой последовательности записывается как одна из цифр последовательности β', вычисленной $\mathcal{H}$. Если N неудовлетворительно, то $R(N) = R(N - 1)$ и машина переходит к $(N + 1)$-й секции своего движения.

Из построения $\mathcal{H}$ видно, что $\mathcal{H}$ свободна от циклов. Каждая секция движения $\mathcal{H}$ завершается после конечного числа шагов. Ибо, в силу нашего предположения о $\mathcal{D}$, решение о том, является ли N удовлетворительным, достигается за конечное число шагов. Если N неудовлетворительно, то N-я секция завер-

шается. Если N удовлетворительно, значит, машина $\mathcal{M}(N)$, для которой N является ЧО, свободна от циклов, и потому ее $R(N)$-ю цифру можно вычислить за конечное число шагов. После того как эта цифра вычислена и записана в качестве $R(N)$-й цифры β', N-я секция завершается. Поэтому $\mathcal{H}$ свободна от циклов.

Пусть теперь K – числовое описание $\mathcal{H}$. Что делает $\mathcal{H}$ в K-й секции своего движения? Она должна проверить, является ли K удовлетворительным, и вынести вердикт «*s*» или «*u*». Поскольку K – это ЧО $\mathcal{H}$ и поскольку $\mathcal{H}$ свободна от циклов, вердикт «*u*» не может быть вынесен. С другой стороны, вердикт не может быть и «*s*». Ибо если бы это было так, то в K-й секции своего движения $\mathcal{H}$ должна была бы вычислить первые $R(K-1)+1=R(K)$ цифр последовательности, вычисленной машиной, ЧО которой равно K, и записать $R(K)$-ю цифру в качестве цифры последовательности, вычисленной $\mathcal{H}$. Вычисление первых $R(K)-1$ цифр прошло бы без проблем, но инструкции для вычисления $R(K)$-й цифры сводились бы к «вычислить первые $R(K)$ цифр, вычисленных $\mathcal{H}$, и записать $R(K)$-ю». Эту $R(K)$-ю цифру никогда не удалось бы найти. То есть машина $\mathcal{H}$ циклическая в противоречии с тем, что мы обнаружили в предыдущем абзаце, и вердиктом «s». Таким образом, оба вердикта невозможны, и мы заключаем, что машины $\mathcal{D}$ не может существовать.

Далее мы можем показать, что *не может существовать машины $\mathcal{E}$ такой, что если снабдить ее СО произвольной машины $\mathcal{M}$, то она определит, напечатает ли $\mathcal{M}$ заданный символ (к примеру 0)*. <...>

6.9. Сколько существует вычислимых чисел

До сих пор не было сделано никакой попытки показать, что множество «вычислимых» чисел включает все числа, которые было бы естественно считать вычислимыми. Все аргументы, которые можно привести, по существу апеллируют к интуиции и по этой причине являются неудовлетворительными с точки зрения математики. На самом деле вопрос формулируется так: «Какие процессы могут выполняться при вычислении числа?»

Я буду использовать аргументы трех видов.

(a) Прямая апелляция к интуиции.
(b) Доказательство эквивалентности двух определений (в случае когда новое определение имеет большую интуитивную привлекательность).
(c) Примеры больших классов вычислимых чисел.

Если принять, что все вычислимые числа действительно «вычислимы», то можно вывести несколько других предложений того же характера. В частности, отсюда следует, что если существует общий процесс, определяющий, является ли доказуемой формула исчисления Гильберта, то такое определение может быть выполнено машиной.

I. [Тип (a)]. Этот аргумент представляет собой лишь развитие идей из § 6.1.

Вычисление обычно заключается в записи некоторых символов на бумаге. Мы можем предположить, что эта бумага разделена на клетки, как школьная тетрадка по арифметике. В элементарной арифметике иногда используется

двумерность листа бумаги. Но такого использования всегда можно избежать, и я думаю, все согласятся, что двумерность бумаги не является существенной для вычисления. Я предполагаю, что вычисление выполняется на одномерной бумаге, т. е. на ленте, разделенной на ячейки. Я также буду предполагать, что число символов, которые могут быть напечатаны, конечно. Если бы мы допустили бесконечное множество символов, то нашлись бы символы, сколь угодно мало отличающиеся друг от друга. Последствия такого ограничения на число символов не слишком серьезны. Всегда можно использовать последовательности символов вместо одного символа. Так, записанное арабскими цифрами число, например 17 или 999999999999999, обычно рассматривается как один символ. Аналогично в любом европейском языке слова считаются отдельными символами (в китайском, однако, сделана попытка ввести неперечислимое бесконечное множество символов). С нашей точки зрения, разница между одиночным и составным символами заключается в том, что составные символы, если они слишком длинные, нельзя обозреть одним взглядом. Это согласуется с нашим опытом. Мы не можем с одного взгляда понять, верно ли, что 9999999999999999 и 999999999999999 – одно и то же число.

Поведение вычислителя в любой момент времени определяется символами, которые он видит, и «состоянием его ума» в этот момент. [Примечание редактора: под «вычислителем» (в оригинале «computer») здесь понимается человек, выполняющий вычисления.] Мы можем предположить, что существует граница B количества символов, или клеток, которые вычислитель может видеть одновременно. Если он пожелает увидеть больше, то должен будет воспользоваться несколькими последовательными наблюдениями. Будем также предполагать, что число состояний ума, принимаемых во внимание, также конечно. Причины такие же, как заставившие нас ограничить количество символов. Если допустить бесконечность состояний ума, то некоторые состояния будут «сколь угодно близки», так что их можно будет спутать. И это ограничение тоже не оказывает серьезного влияния на вычисление, потому что использования более сложных состояний ума можно избежать, записав больше символов на ленту.

Представим себе, что все множество операций, выполняемых вычислителем, можно разделить на «простые операции», которые настолько элементарны, что нелегко представить себе их дальнейшее членение. Каждая такая операция заключается в некотором изменении физической системы, состоящей из вычислителя и его ленты. Мы знаем состояние этой системы, если нам известна последовательность символов на ленте, известно, какие символы видит вычислитель (возможно, вместе с их порядком), и известно состояние ума вычислителя. Мы можем предположить, что в простой операции изменяется не более одного символа. Все прочие изменения можно разбить на простые изменения такого вида. Ситуация в отношении клеток, символы в которых можно изменять таким способом, точно такая же, как в отношении видимых клеток. Поэтому без ограничения общности можно предположить, что клетки, символы в которых изменяются, всегда «видимы».

Помимо этих изменений символов, простые операции могут включать

изменение распределения видимых клеток. Новые видимые клетки должны немедленно распознаваться вычислителем. Я думаю, будет разумно предположить, что таковыми могут быть только клетки, удаленные от предыдущих непосредственно видимых клеток не более чем на фиксированное расстояние. Будем считать, что каждая новая видимая клетка отстоит от какой-то ранее непосредственно видимой клетки не далее, чем на L клеток.

Говоря о «немедленном распознавании», можно представить себе и другие виды немедленно распознаваемых клеток. В частности, клетки, помеченные специальными символами, можно считать немедленно распознаваемыми. Если все такие клетки помечены одиночными символами, то их число может быть лишь конечным, и нет нужды усложнять нашу теорию, присоединяя эти помеченные клетки к видимым. Если, с другой стороны, они помечены последовательностью символов, то процесс распознавания нельзя считать простым. Это фундаментальный момент, нуждающийся в иллюстрации. В большинстве математических работ уравнения и теоремы нумеруются. Обычно номера не превышают (скажем) 1000. Потому теорему можно сразу распознать по ее номеру. Но если бы статья была очень длинной, то можно было бы добраться до теоремы 157767733443477, а затем, читая дальше, встретить фразу «...поэтому (в силу теоремы 157767733443477) имеем...». Чтобы понять, какая теорема имелась в виду, нам пришлось бы сравнить два числа цифра за цифрой, да еще, возможно, отчеркивать цифры карандашом, чтобы не посчитать какую-то дважды. Если, несмотря на эти соображения, все еще возникает мысль о других «немедленно распознаваемых» клетках, то это не расстраивает мои планы, при условии что такие клетки можно найти с помощью какого-то процесса, доступного машине рассматриваемого мной типа. <...>

Таким образом, простые операции могут включать:

(a) изменения символа в одной из видимых клеток;
(b) замену одной из видимых клеток другой клеткой, отстоящей от одной из ранее видимых клеток не далее, чем на L клеток.

Может случиться, что некоторые из этих изменений требуют изменения состояния ума. Поэтому в качестве самой общей одиночной операции следует принять одну из следующих:

(A) возможное изменение (a) символа вкупе с возможным изменением состояния ума;
(B) возможное изменение (b) видимых клеток вкупе с возможным изменением состояния ума.

Фактически выполняемая операция определяется, как и было предложено, ... состоянием ума вычислителя и видимыми символами. В частности, они определяют состояние ума вычислителя после выполнения операции.

Теперь мы можем сконструировать машину, которая сделает работу вычислителя. Каждому состоянию ума вычислителя соответствует «m-конфигурация» машины. Машина сканирует B ячеек, соответствующих B клеткам, видимым вычислителю. На любом такте машина может изменить символ в сканируемой ячейке или заменить любую из сканируемых ячеек другой

ячейкой, отстоящей от одной из сканируемых ячеек не далее, чем на L ячеек. Выполняемый такт и последующая конфигурация определяются сканируемым символом и m-конфигурацией. Описанные только что машины не слишком существенно отличаются от вычислительных машин, описанных в § 6.2, и для каждой машины такого типа можно сконструировать соответствующую вычислительную машину, которая будет вычислять ту же самую последовательность, т. е. последовательность, которую вычисляет вычислитель. <…>

6.10. Примеры больших классов вычисляемых чисел

<…>

6.11. Приложение к проблеме разрешения

У результатов § 6.8 имеются важные приложения. В частности, с их помощью можно показать, что проблема разрешения Гильберта не может иметь решения. <…>

Поэтому я намерен показать, что не может существовать общего процесса, который устанавливал бы доказуемость заданной формулы $\mathfrak{U}$ функционального исчисления K, т. е. что не существует такой машины, которая, получив любую такую формулу $\mathfrak{U}$, рано или поздно скажет, является ли $\mathfrak{U}$ доказуемой.

Возможно, следует отметить, что то, что я собираюсь доказать, сильно отличается от хорошо известных результатов Гёделя. Гёдель показал, что (в рамках формализма «Оснований математики») существуют высказывания $\mathfrak{U}$ такие, что ни $\mathfrak{U}$, ни $-\mathfrak{U}$ невозможно доказать. Отсюда вытекает, что, оставаясь в рамках формализма «Оснований математики» (или K), невозможно доказать его непротиворечивость. С другой стороны, я хочу показать, что не существует общего метода, который сообщил бы, является ли доказуемой в K заданная формула $\mathfrak{U}$, или, что то же самое, является ли система, состоящая из K, с присоединенной дополнительной аксиомой $-\mathfrak{U}$ непротиворечивой.

Если бы была установлена справедливость отрицания, доказанного Гёделем, т. е. если для каждого $\mathfrak{U}$ либо $\mathfrak{U}$, либо $-\mathfrak{U}$ допускает доказательство, то мы сразу бы получили решение проблемы разрешимости. Ибо можно придумать машину K, которая будет последовательно доказывать все доказуемые формулы. Рано или поздно K дойдет либо до $\mathfrak{U}$, либо до $-\mathfrak{U}$. Если она достигла $\mathfrak{U}$, то мы знаем, что $\mathfrak{U}$ доказуемо. Если же она достигла $-\mathfrak{U}$, то, поскольку K непротиворечиво (Hilbert and Ackermann, стр. 65), мы знаем, что $\mathfrak{U}$ не доказуема.

В силу отсутствия в K целых чисел доказательства оказываются довольно длинными. Но лежащие в их основе идеи прямолинейны.

Для каждой вычислительной машины $\mathcal{M}$ мы конструируем соответствующую формулу $\mathrm{Un}(\mathcal{M})$ и показываем, что если существует общий метод уста-

новления доказуемости $\mathrm{Un}(\mathcal{M})$, то существует и общий метод установления того, что $\mathcal{M}$ когда-нибудь напечатает 0.

Возникающие пропозициональные функции интерпретируются следующим образом.

$R_S(x, y)$ интерпретируется как «в полной конфигурации x (машины $\mathcal{M}$) в ячейке y находится символ S». $I(x, y)$ интерпретируется как «в полной конфигурации x ячейка y сканируемая». $K_{q_m}(x)$ интерпретируется как «в полной конфигурации x m-конфигурация равна q_m». $F(x, y)$ интерпретируется как «y непосредственно следует за x». <…>

7 Предлагаемая автоматическая вычислительная машина (1937)

Говард Хатауэй Эйкен

XX век давно наступил, и на столе любого физика и инженера лежали таблицы математических функций, например десятизначные таблицы логарифмов. Логарифмические линейки были полезным инструментом, но их точность была ограничена. Электромеханические настольные калькуляторы были важным подспорьем в бизнесе и в науке, но труд их операторов (по преимуществу женщин, которых называли «вычислителями») был крайне утомителен.

Говард Хатауэй Эйкен (1900–1973) был профессором физики в Гарварде, дослужившимся до звания коммандера в резерве ВМС США. Он поступил в аспирантуру в Гарварде в 33 года, поработав до этого инженером в компании Westinghouse и других электропромышленных компаниях. Приближенно решая уравнения, необходимые для своей диссертации, Эйкен устал от вычислений с помощью имевшихся механических калькуляторов и числовых таблиц. Попавшиеся ему на глаза шестеренки и колесики Бэббиджа стали источником вдохновения, правда, уже после того как он приступил к проектированию собственного автоматического калькулятора (Cohen 1999, стр. 67). Похоже, он был незнаком с деталями конструкции Бэббиджа и пояснениями леди Лавлейс по поводу ее программирования, и нет никаких свидетельств в пользу того, что он что-то знал о революционной математической работе Тьюринга 1936 года или о шифровальной машине, которую Тьюринг сконструировал во время Второй мировой войны.

В этой главе приводится фрагмент предложения Эйкена, направленного им для получения поддержки со стороны промышленности в деле конструирования самой большой когда-либо построенной электромеханической машины. Калькулятор с автоматически управляемой последовательностью операций, получивший впоследствии название Mark I, был разработан при поддержке IBM и введен в эксплуатацию в 1944 году. Часть его выставлена ныне в Гарварде. Оригинальная машина размером 50 футов в длину, 8 футов в высоту и 3 фута в глубину весила почти 5 тонн и представляла собой диво дивное, состоящее из 530 миль проводов, тысяч электромагнитных реле, на которых работали переключатели, десятичные циферблаты для хранения числовых постоянных и десятичные счетчики для промежуточной памяти. Входные данные набивались на перфокартах, выходные печатались на электрической пишущей машинке. В течение десятков лет я проходил мимо этой машины чуть ли не каждый день. Остановиться и изучить ее значит ломать голову над целой кучей эволюционных тупиков, от системы счисления до прокладки проводов. Части соединены прочным сигнальным проводом, обмотанным невыцветшей изоляцией желтого, синего и красного цветов. Но вместо того чтобы собирать провода разных цветов в современные ленточные кабели, так чтобы легко было найти концы одного провода, в Mark I в отдельные пучки собраны все желтые провода, все синие и все красные. Пучки массивные, несколько дюймов толщиной, и сегодня трудно представить себе, для какой цели предназначались разные цвета.

Но самое главное – программа для Mark I набивалась на закольцованной ленте, сделанной из бумаги, которая использовалась в IBM для карт. Таким образом, машина предназначалась для выполнения повторяющихся операций, например суммирования рядов, но не могла выполнить рекурсивный алгоритм, вложенный цикл и даже – в оригинальном варианте – условное ветвление. Даже в более поздних машинах Эйкена (последней была Mark IV) не было концепции хранимой программы. Поэтому хотя Mark I, несомненно, была программируемой – в том смысле, что некоторые ученые могли подготавливать новые программы, тогда как другие использовали машину для выполнения полезной работы, – она не знала того, что сегодня мы называем программным обеспечением. Преданность Эйкена идее, впоследствии названной «гарвардской архитектурой» – данные и программы хранятся в памяти разного вида, – оставила машины Эйкена на обочине прогресса. Развитие продолжалось в других университетах и корпорациях, а Эйкен уволился из Гарварда в возрасте 60 лет.

Mark I шумел, громыхал, но работал. Сложение и вычитание 23-значных десятичных чисел занимало 0,3 с, умножение – до 6 с, деление – до 15,6 с, вычисление $\log x$, e^x или $\sin x$ – минуту или больше. Такого быстродействия было достаточно для численного решения системы из двенадцати линейных уравнений в приложении, для которого экономист из Гарварда Василий Леонтьев использовал эту машину, работая над своей теорией «затраты–выпуск». А Эйкен может смеяться последним. Большинство работающих сегодня компьютеров – встраиваемые системы, их программы прошиты в постоянной памяти и не могут быть случайно изменены во время работы машины – точно так же, как в закольцованной ленте в Mark I.

Желание сэкономить время и умственные усилия, затрачиваемые на арифметические вычисления, и заодно исключить возможность человеческой ошибки, вероятно, так же старо, как сама наука арифметика. Это желание привело к проектированию и конструированию различных вспомогательных устройств для вычислений, начиная с кучек небольших предметов, например камешков, которые сначала использовались россыпью, потом на разлинованных досках, а еще позже – в виде костяшек, нанизанных на проволочный каркас, как в абаке. Этот инструмент, видимо, был изобретен семитскими народностями, затем попал в Индию, а оттуда распространился на запад в Европу и на восток в Китай и Японию.

После изобретения абака никакого прогресса не было, пока в 1617 году Джон Непер не придумал свои счетные палочки, или палочки Непера. Появились различные виды палочек, некоторые из них были близки к механическим вычислениям, но лишь в 1642 году Блез Паскаль подарил нам первую механическую счетную машину в том смысле, в каком это слово употребляется сегодня. Его машина умела только складывать и вычитать, но в 1666 году Сэмюэль Морланд приспособил ее для умножения путем повторного сложения. [Примечание редактора: описано в работе (Morland 1673).]

Следующий шаг сделал Лейбниц, который в 1671 году придумал машину для умножения, а в 1694 году закончил ее конструирование. [Примечание редактора: на самом деле она работала уже к 1674 году и работает по сей день.] В процессе работы над своей машиной Лейбниц изобрел два важных устройства, которые до сих пор встречаются в современных калькуляторах: шаговый барабан (stepped reckoner) и штифтовое колесо.

А тем временем, вслед за изобретением логарифмов Непером, Отред, Джон Браун, Когселл, Эверард и другие придумали логарифмическую линейку. Благодаря дешевизне и простоте конструкции она получила широкое распространение среди ученых уже в 1700 году. Ее усовершенствование продолжается и в наше время, и ширится круг приложений к решению научных задач, требующих точности не ниже трех или четырех знаков после запятой, в условиях, когда общий объем вычислений не слишком велик. Поистине бесценным инструментом логарифмическая линейка оказалась в инженерных расчетах.

Хотя логарифмическая линейка очень быстро получила всеобщее признание, она стала сдерживающим фактором для разработки более точных методов механических вычислений. Поэтому имена крупнейших математиков и физиков всех времен часто встречаются в связи с созданием счетных устройств. Естественно, в своем стремлении двигать вперед науку эти люди рассматривали механические вычисления прежде всего с точки зрения собственных интересов. Заметным исключением был Паскаль, который изобрел счетную машину, чтобы помочь отцу в бухгалтерских расчетах. Несмотря на интерес со стороны широких научных масс, разработка современной вычислительной техники шла медленно до появления крупных коммерческих предприятий и возрастания сложности расчетов, вследствие чего внедрение механических вычислений стало экономической необходимостью. Таким об-

разом, идеи физиков и математиков, которые предвидели возможности и заложили фундамент, нашли приложение к отличным целям, правда, сильно отличающимся от тех, для которых изначально были предназначены.

Немного счетных машин было спроектировано специально для применения в научных исследованиях, заметным исключением стали машины Чарльза Бэббиджа и его последователей. В 1812 году Бэббидж пришел к идее счетной машины нового типа, которую предполагалось использовать для вычисления и печати таблиц математических типов. В основу был положен метод разностей, а сам механизм известен под названием разностной машины. Первый прототип был изготовлен Бэббиджем в 1822 году, а в 1823 году началось конструирование машины при финансовой поддержке Британского правительства. Конструирование продолжалось до 1833 года, когда государственное финансирование было прекращено после израсходования примерно 20 000 фунтов. В настоящее время машина находится в собрании Музея науки в Южном Кенсингтоне. <...>

Со времен Бэббиджа разработка счетных машин продолжалась в ускоренном темпе. Клавишные калькуляторы для одиночных арифметических операций – сложения, вычитания, умножения и деления – были доведены до высокой степени совершенства. Однако в больших коммерческих компаниях объем бухгалтерской работы настолько велик, что эти машины уже не удовлетворяют потребностям.

Поэтому Холлерит вернулся к идее перфокарты, впервые использованной в счетных механизмах Бэббиджем, и положил ее в основу разработки табуляции, подсчета, сортировки; ныне подобные арифметические машины широко используются в промышленности. Разработки в области электрических аппаратов и методов нашли применение и в этих машинах, изготавливаемых компанией International Business Machines, и сегодня многое из того, о чем мечтал Бэббидж, рутинно выполняется в бухгалтериях промышленных предприятий по всему миру.

Как уже было сказано, все эти машины проектируются с прицелом на специальные применения в бухгалтерии. Неизменно они ориентированы на четыре основных арифметических действия, а не на операции алгебраического характера. Однако само их существование делает возможным конструирование автоматической вычислительной машины, предназначенной специально для математических наук.

7.1. Потребность в более мощных методах вычислений в математических и физических науках

Уже отмечалось, что потребность в механической помощи при вычислениях ощущалась с момента становления науки, но в настоящее время она сильнее, чем когда-либо прежде. Быстрое развитие математических и физических наук в последние годы привело в том числе к открытию многих новых полезных функций, и почти все они определяются посредством бесконечных рядов

или иных бесконечных процессов. Большая их часть плохо табулирована, чем тормозится их применение к решению научных задач.

Возросшая точность физических измерений привела к необходимости более точных вычислений в теоретической физике, а опыт показал, что небольшие расхождения между расчетными теоретическими и экспериментальными результатами могут привести к открытию новых физических явлений, иногда имеющих огромную научную и промышленную ценность.

Многие недавние научные разработки, в т. ч. такие устройства, как электровакуумные лампы с накаливаемым катодом, основаны на нелинейных эффектах. Все чаще дифференциальные уравнения, описывающие эти физические эффекты, имеют не изученную ранее форму, а потому не поддаются никаким известным методам интегрирования. В таком случае единственные способы решения – разложение в бесконечный ряд и численное интегрирование. То и другое требует гигантских объемов вычислений.

Нынешнее развитие теоретической физики в направлении волновой механики целиком основано на математических идеях и ясно показывает, что будущее физических наук опирается на математические рассуждения, направляемые экспериментом. В настоящее время существуют проблемы, которые мы не в состоянии решить не из-за теоретических затруднений, а просто в силу недостаточности средств механического вычисления.

В некоторых разделах физики, например при изучении ионосферы, математические выражения, описывающие явление, настолько длинные и сложные, что их невозможно записать в нескольких строках на печатной странице, и тем не менее численное исследование таких выражений абсолютно необходимо для изучения физики верхних слоев атмосферы; именно от результатов таких исследований зависит будущее радиосвязи и телевидения.

Это лишь несколько примеров вычислительных трудностей, с которыми сталкиваются физические и математические науки, к ним стоило бы добавить много других из астрономии, теории относительности и даже быстро развивающейся математической экономики. Все эти вычислительные трудности можно устранить, разработав подходящую автоматическую вычислительную машину.

7.2. Чем арифмометр на перфокартах отличается от вычислительной машины, необходимой в науке

Ниже перечислены особенности, которые должны присутствовать в вычислительной машине, специально разработанной для быстрой работы над научными задачами, и отсутствуют в вычислительных машинах, которые производятся для бухгалтерских расчетов.

1. Обыкновенные бухгалтерские машины предназначены почти всецело для работы с положительными числами, тогда как машины для математических целей должны уметь работать как с положительными, так и с отрицательными числами.

2. Для математических целей вычислительная машина должна уметь использовать широкий спектр трансцендентных функций, включая тригонометрические, эллиптические, бесселевы, функции вероятности и многие другие. По счастью, не все эти функции встречаются в одном вычислении, поэтому должны быть предусмотрены средства переключения с одной функции на другую и обеспечена надлежащая гибкость.
3. Большинство вычислений в математике, в т. ч. вычисление функций, представленных рядами, вычисление по формулам, решение дифференциальных уравнений методами численного интегрирования и т. д., состоят из повторяющихся процессов. После того как процесс начат, он может продолжаться неопределенно долго, пока не будет исчерпана вся область значений независимых переменных, и обычно эта область может быть исчерпана последовательными шагами равного размера. По этой причине вычислительная машина, предназначенная для применения к математическим наукам, должна работать полностью автоматически с момента начала процесса.
4. Существующие вычислительные машины способны вычислить $\phi(x)$ как функцию x по шагам. Так, если областью определения x является интервал $a < x < b$ и $\phi(x)$ получается из x с помощью последовательности арифметических операций, то существующая процедура заключается в том, чтобы вычислить шаг (1) для всех значений x в интервале $a < x < b$. Затем выполняется шаг (2) для всех значений результата шага (1) и т. д., пока не будет получено $\phi(x)$. Однако этот процесс противоположен тому, что необходимо во многих математических операциях. Вычислительная машина, предназначенная для математических наук, должна быть способна вычислять строки вместо столбцов, поскольку очень часто, как, например, при численном решении дифференциального уравнения, вычисление второго значения в таблице функции зависит от предшествующего значения или нескольких значений.

По большому счету, эти четыре особенности – все, что необходимо для превращения существующих калькуляторов на основе перфокарт типа тех, что производит компания International Business Machines, в машины, специально приспособленные для научных целей. В силу большей сложности научных задач по сравнению с бухгалтерскими количество арифметических элементов должно быть значительно увеличено.

7.3. Какие математические операции следует включить

Ниже перечислены математические операции, которые должны быть включены в автоматическую вычислительную машину:

1. Основные арифметические действия: сложение, вычитание, умножение и деление.
2. Положительные и отрицательные числа.

3. Круглые и квадратные скобки: ( ) + ( ), [( ) + ( )] _ [( ) + ( )] и т. д.
4. Степени чисел: целые, дробные.
5. Логарифмы: десятичные и по любому другому основанию.
6. Антилогарифмы, или показательные функции: по основанию 10 и другим основаниям.
7. Тригонометрические функции.
8. Обратные тригонометрические функции.
9. Гиперболические функции.
10. Обратные гиперболические функции.
11. Высшие трансцендентные функции: интегральная функция распределения, эллиптическая функция и бесселева функция. С помощью этих функций должно быть возможно выполнить следующие процессы.
12. Вычисление по формуле и состояние таблицы результатов.
13. Вычисление ряда.
14. Решение обыкновенных дифференциальных уравнений первого и второго порядков.
15. Численное интегрирование эмпирических данных.
16. Численное дифференцирование эмпирических данных.

7.4. Математические средства выполнения операций

Следующие математические процессы можно положить в основу проектирования автоматической вычислительной машины.

1. Основные арифметические действия не нуждаются в комментариях, поскольку уже доступны. Следует разве что отметить, что все остальные операции должны в конечном итоге сводиться к этим, чтобы механическое устройство можно было использовать.
2. К счастью, алгебра положительных и отрицательных знаков очень проста. В любом случае имеется всего две возможности. Ниже будет показано, что для целей механического вычисления с этими знаками можно обращаться, как с числами.
3. Использование круглых и квадратных скобок при записи формул требует, чтобы вычисление производилось по частям. Таким образом, сначала нужно получить часть результата, сохранить ее до вычисления следующей части и т. д. Это означает, что вычислительная машина должна быть оснащена средствами временного хранения чисел до того момента, как они понадобятся в вычислении. Такие средства предоставляют счетчики.
4. Целые степени чисел можно получить последовательным умножением, а дробные степени – методом итераций. Так, если требуется найти $5^{1/3}$, то следует рассмотреть функцию

$$y = f(x) = x^3 - 5$$

и выполнить итерации

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})};$$
$$x_n = x_{n-1} - \frac{x_{n-1}^3 - 5}{3x_{n-1}^2},$$

или

$$x_n = \frac{2}{3}x_{n-1} + \frac{5}{3x_{n-1}^2}.$$

Положим

$$\begin{aligned} x_0 &= 2; \\ x_1 &= \frac{4}{3} + \frac{5}{12} = \frac{21}{12}; \\ x_2 &= \frac{42}{36} + \frac{5 \times 144}{3 \times 441} = 1.166 + 0.544 \\ &= 1.710. \end{aligned} \tag{7.1}$$

Это и есть кубический корень из 5 с точностью до четырех значащих цифр. В общем случае корень степени r из θ получается итерированием выражения

$$x_n = \left(1 - \frac{1}{r}\right)x_{n-1} + \frac{\theta}{rx_{n-1}^{r-1}}.$$

Наконец, если r не целое, то следует прибегнуть к помощи таблиц логарифмов, как описано ниже. <...>

16. Численное интегрирование эмпирических данных можно выполнить методами Симпсона, Уэддля, Гаусса и др. Во всех этих методах используются суммы последовательных значений y, умноженных на те или иные числовые коэффициенты. Поэтому единственный новый механический компонент – средства задания списка чисел. Как это сделать, описывается ниже.
17. Для численного дифференцирования эмпирических данных лучше всего применить разностную формулу. Точность большинства экспериментальных наблюдений такова, что разностями пятого порядка можно пренебречь, если наблюдения расположены достаточно близко друг к другу. Но если всеми разностями выше пятого порядка можно пренебречь, то процедуру численного дифференцирования можно выполнить разностным механизмом пятого порядка, изобретенным еще Бэббиджем. Такое устройство можно, однако, собрать из стандартных машин для сложения и вычитания с незначительными изменениями. Устройство дифференцирования можно было бы применить и ко многим другим задачам. На самом деле большинство вышеупомянутых проблем при определенных обстоятельствах можно решить примене-

нием разностных формул.

7.5. Механические вычисления

В предыдущем разделе было показано, что даже сложные математические операции можно свести к повторяющемуся процессу применения основных действий арифметики. В настоящее время машины компании International Business Machines Company умеют выполнять такие операции, как

$$\begin{aligned} A + B &= F; \\ A - B &= F; \\ AB + C &= F; \\ AB + C + D &= F; \\ A + B + C &= F; \\ A - B - C &= F; \\ A + B - C &= F. \end{aligned} \tag{7.2}$$

В этих равенствах A, B, C, D – табуляции чисел, набитые на перфокартах, а результат F также выдается на перфокарте. Карты, содержащие F, могут быть затем пропущены через другую машину и напечатаны, или же их можно использовать в качестве карт A, B, ... в другом вычислении.

Чтобы заставить данную машину выполнять вместо любой из операций (7.2) любую другую, необходимо изменить соединение проводов на коммутационной панели. Опытный оператор производит такие изменения за несколько минут.

Далее не будет никаких описаний механизма работы машин IBM. Достаточно сказать, что все операции, описанные в предыдущем разделе, можно выполнить на существующих машинах, если добавить подходящие средства управления и объединить их в единую сборку в достаточном количестве. Таким образом, задача проектирования автоматической вычислительной машины, пригодной для математических операций, сводится к задаче проектирования подходящего блока управления, и даже эта задача уже решена для простых математических операций.

Основные функции специализированных средств управления – коммутация машины и замена перфокарт непрерывной перфолентой. Чтобы переключиться с текущей последовательности коммутации на любую другую возможную последовательность, сам механизм переключения должен использовать бумажную управляющую ленту, на которой математические формулы могут быть представлены нужным образом расположенными просечками.

7.6. Существующие концепции аппаратуры

В настоящее время автоматический калькулятор видится как коммутатор, в котором смонтированы различные элементы вычислительной аппаратуры. Каждая панель коммутатора предназначена для определенных математических операций.

Ниже приведен краткий обзор требуемой аппаратуры.

1. В машинах IBM применяются два электрических потенциала: переменный ток напряжением 120 В для питания мотора и постоянный ток напряжением 32 вольт для поддержания работы реле и т. д. Необходимо будет добавить панель главного источника питания, включающую управление мотором-генератором на 110 В переменного тока / 32 В постоянного тока и необходимые предохранители для защиты всех цепей.
2. Главная панель управления: цель этого элемента управления – маршрутизация потока прохождения чисел через машину и запуск операции. Выполняются следующие процессы: (a) доставка числа из позиции (x) в позицию (y) и (b) запуск операции, для которой предполагается позиция (y). Сам главный блок управления должен допускать возможность блокировки, чтобы предотвратить попытку переместить число до того, как определено его значение, или начать следующую операцию в позиции (y) раньше, чем завершена предыдущая.
   Было бы желательно иметь четыре таких главных блока управления, каждый из которых способен управлять всей машиной или любой ее частью. Тогда для решения сложных задач можно было бы задействовать все имеющиеся ресурсы, а для более простых, требующих меньше ресурсов, можно было бы одновременно решать несколько задач.
3. Приращение независимой переменной в любом вычислении должно осуществляться равными шагами, величину которых можно корректировать вручную. Простейший способ получить такую арифметическую последовательность – подать суммирующей машине первое значение x_0 и приращение Δx. Тогда последовательное прибавление Δx даст нужную последовательность.
   Должно быть четыре таких независимых устройства, чтобы (a) вычислять формулы, содержащие четыре переменные, и (b) независимо работать с четырьмя главными блоками управления.
4. Некоторые постоянные: многие математические формулы включают такие постоянные, как e, π, $\log_{10} e$ и т. д. Эти постоянные должны быть доступны в любой момент времени.
5. Математические формулы почти всегда включают постоянные величины. При вычислении формулы как функции от независимой переменной эти постоянные используются снова и снова. Поэтому машина должна быть оснащена 24 изменяемыми позициями для хранения чисел, представляющих эти постоянные.
6. При вычислении бесконечных рядов значение 24 может быть значительно превышено. Для этого случая должна быть возможность вводить специальные значения посредством перфоленты, так что каждое по-

следующее значение подается путем продвижения ленты вперед на одну позицию. Должно быть предусмотрено два таких устройства.

7. Ввод эмпирических данных для неповторяющихся операций проще всего реализовать с помощью стандартной магазинной подачи перфокарт. Должно быть предусмотрено одно такое устройство.
8. На различных этапах вычисления, включающего круглые и квадратные скобки, может оказаться необходимым сохранить частичный результат в ожидании вычисления другой части. Если результаты хранятся в вычислительных блоках, то эти блоки оказываются недоступны для выполнения последующих шагов. Поэтому необходимо изъять числа из вычислительных блоков и временно поместить их в позиции хранения. Должно быть предусмотрено двенадцать таких позиций.
9. На трех машинах можно выполнять основные арифметические действия: сложение и вычитание, умножение и деление. Должно быть предусмотрено четыре блока для каждой операции в дополнение к тем, что непосредственно связаны с трансцендентными функциями.
10. В число постоянно присутствующих математических функций должны входить: логарифмы, антилогарифмы, синусы, косинусы, арксинусы и арктангенсы.
11. Необходимо два блока для разложения других функций в ряд Маклорена.
12. Для выполнения дифференцирования и интегрирования эмпирических данных должны быть предусмотрены суммирующие и вычитающие аккумуляторы, достаточные для вычисления разностей пятого порядка.
13. Все результаты должны по желанию печататься, выводиться на перфоленты или перфокарты. Окончательные результаты печатаются. Промежуточные результаты выводятся на перфорированный носитель для дальнейших вычислений.

	Произведений в час
2 × 8	1500
3 × 8	1285
4 × 8	1125
5 × 8	1000
6 × 8	900
7 × 8	818
8 × 8	750

Рис. 7.1. Ожидаемая скорость умножения

Предполагается, что только что описанная аппаратура, управляемая автоматическим коммутатором, сможет решить большинство встречающихся задач.

7.7. Вероятная скорость вычислений

Представление о скорости, достижимой машинами IBM, можно получить из таблицы на рис. 7.1, где 2×8 означает умножение числа с 8 значащими цифрами на число с 2 значащими цифрами без учета нулей.

При вычислении логарифмов с 10 значащими цифрами средняя скорость составит приблизительно 90 результатов в час. На вычисление таких логарифмов для всех натуральных чисел от 1000 до 100 000 потребовалось бы приблизительно 1100 часов, или 50 дней, без учета времени сложения или печати. Это оправдано, потому что операции сложения и печати выполняются очень быстро и могут производиться одновременно с умножением.

7.8. Предполагаемая точность

В примерах выше использовались десятизначные числа. Если бы все числа задавались с такой точностью, то в большинстве вычислительных компонентов нужно было бы обеспечить 23 разряда: 10 слева от десятичной точки, 12 справа и еще один для знака плюс или минус. Из двенадцати разрядов справа два являются дополнительными и отбрасываются.

7.9. Простота публикации результатов

Как уже отмечалось, все результаты вычислений должны печататься в табличной форме. С помощью фотолитографии их можно печатать непосредственно, без верстки и корректуры. Это не только дает существенную экономию при публикации математических функций, но также исключает многие возможности появления ошибок.

8 Символический анализ релейных и переключательных схем

Клод Шеннон

Электрические провода, соединенные переключателями, либо проводят ток, либо нет в зависимости от структуры соединений и положений переключателей. Еще в 1886 году философ Чарльз Сандерс Пирс обратил внимание на связь между логическими функциями и параллельно-последовательными электрическими схемами. Но только Клод Шеннон (1916–2001) впервые исследовал электрическую интерпретацию законов мышления Буля. Шеннон с детства возился с электрическими схемами, когда жил в деревушке на севере штата Мичиган и смастерил проводную телеграфную линию до соседского дома. 1930-е годы – время расцвета компаний, связанных с радио и телефоном, и Шеннон начал изучать электротехнику в Мичиганском университете. С работами Буля он познакомился на курсе по философии, а в бытность свою аспирантом Массачусетского технологического института (МТИ) установил связь между обеими двоичными системами и тем самым положил начало исключительно элегантной методике проектирования сложных электрических схем.

Сегодня мы считаем само собой разумеющимся, что требуемое поведение схемы можно описать математически и что получающимися формулами можно манипулировать, применяя математические правила, а затем «откомпилировать» их, синтезировав аппаратную реализацию. Но первым это сделал Шеннон в своей магистерской диссертации. В этом разделе приво-

дится выдержка из статьи, вошедшей в состав этой диссертации; она оказала огромное влияние на проектирование электрических схем. В ней рассматриваются анализ и синтез электрических схем, некоторые из которых гораздо сложнее простых примеров, включенных в данный текст. В той части, которую мы опустили, Шеннон устанавливает верхнюю и нижнюю границы размера схем для некоторых булевых функций при определенных ограничениях на форму схемы и тем самым предвосхищает богатую результатами и важную дисциплину – сложность электрических схем.

8.1. Введение

В схемах управления и защиты сложных электрических систем часто бывают необходимы сложные соединения контактов реле и переключателей. Такие схемы используются в автоматических телефонных коммутаторах, аппаратуре управления двигателями и в большинстве схем, предназначенных для автоматизации сложных процессов. В этой статье излагается математический анализ некоторых свойств таких схем. Особое внимание уделяется проблеме синтеза схем. Пусть даны некоторые условия функционирования; требуется найти схему, реализующую эти условия. Решение задач такого рода неоднозначно; в работе исследуются методы нахождения индивидуальных схем, требующих наименьшего числа контактов реле и переключателей. Также дается описание методов нахождения схем, эквивалентных данной по всем заданным условиям функционирования. В работе показано, что некоторым хорошо известным теоремам о схемах, составленных из элементов с заданным импедансом, соответствуют аналогичные теоремы о релейных схемах. Отметим здесь теоремы о преобразованиях треугольник–звезда и звезда–многоугольник и теорему двойственности.

Метод решения поставленных проблем может быть кратко описан следующим образом. Любая схема представляется в виде системы уравнений, составленных из символов, соответствующих различным реле и переключателям схемы. Разрабатывается аппарат для преобразования этих уравнений с помощью простых математических приемов, большинство из которых подобно обычным алгебраическим алгоритмам. Показывается, что этот аппарат в точности аналогичен исчислению высказываний символической логики. Для синтеза схемы заданные условия сначала записываются в виде системы уравнений, затем уравнения преобразуются к виду, соответствующему простейшей схеме. Тогда схема может быть получена непосредственно из уравнений. Этим методом всегда можно найти простейшую параллельно-последовательную схему, а в некоторых случаях – простейшую схему, содержащую любые типы соединений.

Используемые нами обозначения взяты главным образом из символической логики. Из большого многообразия применяемых в наши дни систем выбрана та, которая представляется более простой и удобной для нашей

интерпретации. Некоторые из употребляемых нами терминов, как, например, узел, многоугольник, треугольник, звезда и т. п., заимствованы из общей теории электротехнических схем для обозначения сходных понятий в переключательных схемах.

8.2. Параллельно-последовательные двухполюсные схемы: основные определения и постулаты

Ограничим наше исследование схемами, содержащими только контакты реле и переключатели. Такие схемы между любыми двумя своими полюсами в каждый момент времени либо замкнуты (нулевое сопротивление), либо разомкнуты (бесконечное сопротивление). Сопоставим полюсам a и b символ X_{ab}, или проще X. Эту переменную функцию времени будем называть двоичным сопротивлением двухполюсной схемы $a - b$. Символом 0 (нуль) будем обозначать сопротивление замкнутой схемы, а символом 1 (единица) – сопротивление разомкнутой схемы. Следовательно, если схема $a - b$ разомкнута, то $X_{ab} = 1$, а если замкнута, то $X_{ab} = 0$. Сопротивления X_{ab} и X_{cd} называются равными, если схема $a - b$ разомкнута тогда и только тогда, когда схема $c - d$ разомкнута. Пусть теперь символ + (плюс) определяется в смысле последовательного соединения двухполюсных схем, при котором сопротивления складываются. Так, $X_{ab} + X_{cd}$ – это сопротивление схемы $a - d$, полученной в результате объединения полюсов b и c схем $a - b$ и $c - d$. Аналогично произведение двух сопротивлений $X_{ab} \cdot X_{cd}$, или короче $X_{ab}X_{cd}$, определяется как сопротивление схемы, образованной параллельным соединением схем $a - b$ и $c - d$. Контакты реле или переключатели изображаются в схемах, как показано на рис. 8.1, где буквы – это соответствующие функции сопротивления. На рис. 8.2 показана интерпретация сложения, а на рис. 8.3 – умножения. Такой выбор символики делает операции над сопротивлениями очень похожими на обычные алгебраические преобразования.

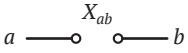


Рис. 8.1. Символическое изображение функции сопротивления

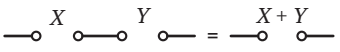


Рис. 8.2. Интерпретация сложения

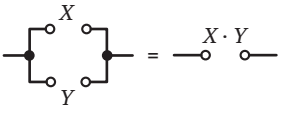


Рис. 8.3. Интерпретация умножения

Очевидно, что при приведенных выше определениях справедливы следующие постулаты.

1. *a*. $0 \cdot 0 = 0$ — Замкнутая схема, соединенная параллельно с замкнутой схемой, есть замкнутая схема

	b.	1 + 1 = 1	Разомкнутая схема, соединенная последовательно с разомкнутой схемой, есть разомкнутая схема
2.	*a.*	1 + 0 = 0 + 1 = 1	Разомкнутая схема, соединенная последовательно с замкнутой схемой в любом порядке (т. е. разомкнутая схема справа или слева от замкнутой), есть разомкнутая схема
	b.	0 · 1 = 1 · 0 = 0	Замкнутая схема, соединенная параллельно с разомкнутой схемой (в любом порядке), есть замкнутая схема
3.	*a.*	0 + 0 = 0	Замкнутая схема, соединенная последовательно с замкнутой схемой, есть замкнутая схема
	b.	1 · 1 = 1	Разомкнутая схема, соединенная параллельно с разомкнутой схемой, есть разомкнутая схема
4.			В каждый момент либо $X = 0$, либо $X = 1$

Этих постулатов достаточно, чтобы вывести все теоремы, которые будут использованы в связи со схемами, образованными только из последовательных и параллельных соединений. Постулаты расположены попарно, чтобы подчеркнуть двойственность между операциями сложения и умножения и константами 0 и 1. Так, если в каком-нибудь постулате, относящемся к типу *a*, нули заменить на единицы, умножение на сложение и наоборот, то получится соответствующий постулат типа *b*. Этот факт очень важен. Он дает для каждой теоремы двойственную; достаточно доказать одну, чтобы установить обе. Единственный из введенных нами постулатов, отличающийся от постулатов обычной алгебры, – это 1*b*. Тем не менее он дает возможность существенно упростить операции над нашими символами.

8.2.1. Теоремы. Приведем ряд теорем о способах соединения сопротивлений. Так как любая из этих теорем может быть доказана очень простым способом, приведем один пример доказательства в качестве иллюстрации. Методом доказательства является метод «полной индукции», т. е. метод проверки теоремы для всех возможных случаев. Так как в силу постулата 4 каждая переменная может принимать только значения 0 и 1, это легко сделать. Некоторые теоремы могут быть доказаны более изящно сведением к предыдущим теоремам, но метод полной индукции настолько универсален, что, вероятно, ему следует отдать предпочтение.

$$X + Y = Y + X; \tag{8.1a}$$

$$XY = YX; \tag{8.1b}$$

$$X + (Y + Z) = (X + Y + Z); \tag{8.2a}$$

$$X(YZ) = (XY)Z; \tag{8.2b}$$

$$X(Y + Z) = XY + XZ; \tag{8.3a}$$

$$X + YZ = (X + Y)(X + Z) \tag{8.3b}$$

$$1 \cdot X = X; \tag{8.4a}$$

$$0 + X = X; \tag{8.4b}$$

$$1 + X = 1; \tag{8.5a}$$

$$0 \cdot X = 0. \tag{8.5b}$$

Например, чтобы доказать теорему 8.4*a*, заметим, что X есть либо 0, либо 1. Если $X = 0$, теорема следует из постулата 2*b*; если $X = 1$, она следует из постулата 3*b*. Теорема 8.4b теперь вытекает из 8.4a по принципу двойственности в результате замены 0 на 1 и · на +.

В силу ассоциативных законов (8.2a) и (8.2b), в сумме или произведении нескольких членов скобки могут быть опущены. Символы $\sum$ и $\prod$ имеют те же значения, что и в обычной алгебре.

Дистрибутивный закон (8.3a) дает возможность «развернуть» умножение и «свернуть» сумму. Двойственная теорема (8.3b), однако, в обычной алгебре неверна.

Определим теперь новую операцию, которую назовем отрицанием. Отрицание сопротивления X обозначается через X' и определяется как переменная, равная 1, когда X равно 0, и равная 0, когда X равно 1. Если X – сопротивление замыкающего контакта реле, то X' – сопротивление размыкающего контакта того же реле. Из определения отрицания сопротивления вытекают теоремы:

$$X + X' = 1; \tag{8.6a}$$

$$XX' = 0; \tag{8.6b}$$

$$0' = 1; \tag{8.7a}$$

$$1' = 0; \tag{8.7b}$$

$$(X')' = X. \tag{8.8}$$

8.2.2. Аналогия с исчислением высказываний. Теперь можно доказать эквивалентность введенного исчисления некоторой элементарной части исчисления высказываний. Алгебра логики, введенная Джорджем Булем, является символическим методом вывода логических соотношений. Символы булевой алгебры допускают две логические интерпретации. При интерпретации в терминах классов переменные могут принимать не только два значения 0 и 1. Эта интерпретация называется алгеброй классов. Если, однако, символы рассматриваются как высказывания, то имеем исчисление высказываний, в котором переменные принимают только значения 0 и 1, как и рассмотренные выше функции сопротивления. Обычно обе интерпретации основываются на одном и том же множестве постулатов, за исключением того, что в случае исчисления высказываний к постулатам 1–4 добавляется постулат эквивалентности. Э. В. Хантингтон дает следующую систему аксиом символической логики:

1. Класс K содержит по крайней мере два различных элемента.
2. Если a и b принадлежат классу K, то $a + b$ принадлежит классу K.
3. $a + b = b + a$.
4. $(a + b) + c = a + (b + c)$.

5. $a + a = a$.
6. $ab + ab' = a$, где ab определяется как $(a' + b')'$.

Если положить, что класс K состоит из двух элементов 0 и 1, то эти постулаты будут следствиями постулатов, приведенных во введении, и наоборот, приведенные там постулаты 1, 2, 3 могут быть выведены из постулатов Хантингтона. Если добавить постулат 4 и ограничиться исчислением высказываний, то становится очевидной полная аналогия между этой ветвью символической логики и исчислением переключательных схем. Обе интерпретации символов показаны на рис. 8.4.

Символ	**Интерпретация в терминах релейных схем**	**Интерпретация в терминах исчисления высказываний**
X	Схема X	Высказывание X
0	Схема замкнута	Высказывание ложно
1	Схема разомкнута	Высказывание истинно
$X + Y$	Последовательное соединение схем X и Y	Высказывание истинно, если либо X, либо Y истинно
XY	Параллельное соединение схем X и Y	Высказывание истинно, если как X, так и Y истинно
X'	Схема, которая разомкнута, если X замкнута, и замкнута, если X разомкнута	Отрицание высказывания X
=	Схемы разомкнуты или замкнуты одновременно	Каждое из высказываний влечет другое

Рис. 8.4. Аналогия между исчислением высказываний и символическим анализом релейных схем

Благодаря этой аналогии любая теорема исчисления высказываний является также истинной теоремой, если ее интерпретировать в терминах релейных схем. Остальные теоремы данного раздела были установлены непосредственно на этой основе.

Теоремы де Моргана

$$(X + Y + Z + \ldots)' = X' \cdot Y' \cdot Z' \cdot \ldots; \tag{8.9a}$$

$$(X \cdot Y \cdot Z \cdot \ldots)' = X' + Y' + Z' + \ldots \tag{8.9b}$$

выражают отрицание суммы или произведения в терминах отрицания слагаемых или сомножителей. Они могут быть проверены для двух членов подстановкой всех возможных значений, а затем методом индукции распространены на любое число n переменных.

Функция нескольких переменных $X_1, X_2, \ldots, X_n$ – это выражение, образованное из переменных при помощи операций сложения, умножения и отрицания. Такая функция будет обозначаться $f(X_1, X_2, \ldots, X_n)$. Так, например, мы можем записать:

$$f(X, Y, Z) = XY + X'(Y' + Z').$$

В анализе бесконечно малых показано, что любую функцию (при условии что она непрерывна и все ее производные непрерывны) можно разложить в ряд Тейлора. Похожее разложение возможно и в исчислении высказываний. Чтобы вывести разложение функции в ряд, рассмотрим сначала следующие равенства [Примечание редактора: ср. Буль, стр. 73.]:

$$f(X_1, X_2, \ldots, X_n) = X_1 \cdot f(1, X_2, \ldots, X_n) + X_1' \cdot f(0, X_2, \ldots, X_n) \qquad (8.10a)$$

$$f(X_1, X_2, \ldots, X_n) = [f(0, X_2, \ldots, X_n) + X_1] \cdot [f(1, X_2, \ldots, X_n) + X_1']. \qquad (8.10a)$$

Они превращаются в тождества, если положить $X_1 = 0$ или $X_1 = 1$. В этих равенствах функция f называется разложенной по X_i. Коэффициенты при X_1 и X_1' в (8.10a) являются функциями от $n - 1$ переменных и могут быть таким же образом разложены по любой из этих переменных. Аддитивные члены в (8.10b) тоже могут быть разложены таким же образом. Разлагая по X_2, получим:

$$f(X_1, X_2, \ldots, X_n) = X_1X_2 f(1, 1, X_3, \ldots, X_n) + X_1X_2' f(1, 0, X_3, \ldots, X_n) + X_1'X_2 f(0, 1, X_3, \ldots, X_n) + X_1' X_2' f(0, 0, X_3, \ldots, X_n); \qquad (8.11a)$$

$$f(X_1, X_2, \ldots, X_n) = [X_1 + X_2 + f(0, 0, X_3, \ldots, X_n)] \cdot [X_1 + X_1' + f(0, 1, X_3, \ldots, X_n)] \cdot [X_1' + X_2 + f(1, 0, X_3, \ldots, X_n)] \cdot [X_1' + X_2' + f(1, 1, X_3, \ldots, X_n)]. \qquad (8.11b)$$

Повторяя разложение n раз, придем к полным разложениям в ряд, имеющим вид:

$$f(X_1, \ldots, X_n) = f(1, 1, 1, \ldots, 1)X_1X_2\ldots X_n + f(0, 1, 1, \ldots, 1)X_1'X_2\ldots X_n + \ldots + f(0, 0, 0, \ldots, 0)X_1' X_2'\ldots X_n'; \qquad (8.12a)$$

$$f(X_1, \ldots, X_n) = [X_1 + X_2 + \ldots X_n + f(0, 0, 0, \ldots, 0) \cdot \ldots \cdot [X_1' + X_2' + \ldots + X_n' + f(1, 1, 1, \ldots, 1)]. \qquad (8.12b)$$

Согласно (8.12a), f равна сумме произведений, полученных путем расстановки знаков отрицаний при $X_1, X_2, \ldots, X_n$ всеми возможными способами и умножения каждого произведения на коэффициент, равный значению функции, когда это произведение есть 1. Аналогично для (8.12b).

В качестве приложения такого разложения покажем, что если требуется найти схему, реализующую данную функцию, можно всегда разложить эту функцию по формуле (8.10a) или (8.10b) так, что некоторая выделенная переменная встречается не более двух раз – один раз как замыкающий и один раз как размыкающий контакт. Это показано на рис. 8.5. Согласно формулам (8.11a) и (8.11b), другая переменная встречается не более четырех раз (два раза как замыкающий контакт и два раза – как размыкающий) и т. д.

Обобщение теоремы де Моргана представляется символически следующим уравнением:

$$f(X_1, X_2, \ldots, X_n, +, \cdot)' = f(X_1', X_2', \ldots, X_n', \cdot, +). \qquad (8.13)$$

Под этим подразумевается, что отрицание любой функции может быть получено заменой каждой переменной ее отрицанием и перестановкой сим-

волов + и ·. Явные и неявные скобки остаются, конечно, на тех же местах. Например, отрицание $X + Y(Z + WX')$ будет иметь вид $X'[Y' + Z'(W' + X)]$.

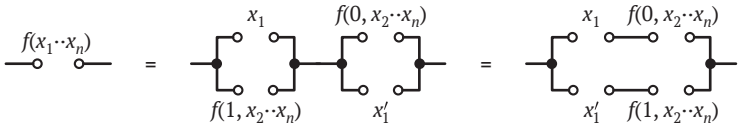


Рис. 8.5. Разложение по одной переменной

Приведем еще некоторые теоремы, используемые для упрощения формул:

$$X = X + X = X + X + X = \ldots; \tag{8.14a}$$

$$X = X \cdot X = X \cdot X \cdot X = \ldots; \tag{8.14b}$$

$$X + XY = X; \tag{8.15a}$$

$$X(X + Y) = X; \tag{8.15b}$$

$$XY + X'Z = XY + X'Z + YZ; \tag{8.16a}$$

$$(X + Y)(X' + Z) = (X + Y)(X' + Z)(Y + Z); \tag{8.16b}$$

$$Xf(X, Y, Z, \ldots) = Xf(1, Y, Z, \ldots); \tag{8.17a}$$

$$X + f(X, Y, Z, \ldots) = X + f(0, Y, Z, \ldots); \tag{8.17b}$$

$$X'f(X, Y, Z, \ldots) = X'f(0, Y, Z, \ldots); \tag{8.18a}$$

$$X' + f(X, Y, Z, \ldots) = X' + f(1, Y, Z, \ldots). \tag{8.18b}$$

Все эти теоремы могут быть доказаны методом полной индукции.

Любое выражение, образованное при помощи операций сложения, умножения и отрицания, является точным представлением схемы, содержащей только последовательные и параллельные соединения. Такая схема называется параллельно-последовательной. Каждая буква в выражении такого рода представляет замыкающий, размыкающий или переключающий контакт реле либо переключателя. Поэтому чтобы найти параллельно-последовательную схему, содержащую наименьшее число контактов, необходимо преобразовать выражение к форме, содержащей наименьшее число букв. Для этого вполне достаточно теорем, приведенных выше. Небольшой навык в оперировании символами – вот все, что требуется. К счастью, большинство из этих теорем в точности такие же, как в обычной алгебре. Автор считает, что особенно полезными для упрощения сложных выражений являются теоремы 8.3, 8.6, 8.9, 8.14, 8.15, 8.16a, 8.17 и 8.18. Часто функция может быть записана различными способами, требующими одного и того же минимального числа элементов. В таком случае может быть выбрана любая из схем, или выбор может быть продиктован иными соображениями.

В качестве примера упрощения формулы рассмотрим схему, изображенную на рис. 8.6. Функцией сопротивления X_{ab} этой схемы будет:

$$X_{ab} = W + W'(X + Y) + (X + Z)(S + W' + Z)(Z' + Y + S'V)$$
$$= W + X + Y + (X + Z)(S + 1 + Z)(Z' + Y + S'V)$$
$$= W + X + Y + Z(Z' + S'V).$$

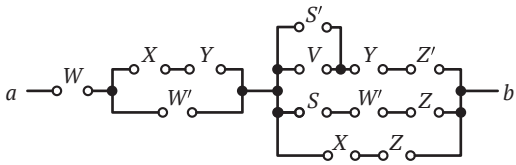


Рис. 8.6. Схема, подлежащая упрощению

Эти преобразования были произведены с помощью формулы (8.17b), где в качестве X последовательно бралось сначала W, затем X и Y. Раскрывая скобки, получим:

$$X_{ab} = W + X + Y + ZZ' + ZS'V = W + X + Y + ZS'V.$$

Схема, соответствующая этой формуле, изображена на рис. 8.7. Следует обратить внимание на большое сокращение числа элементов. <...>

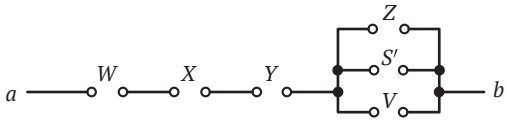


Рис. 8.7. Упрощение схемы, изображенной на рис. 8.6

9 Логическое исчисление идей, относящихся к нервной активности

Уоррен Мак-Каллок и Уолтер Питтс

Это странная и удивительная статья, запутанная технически и пророческая до высокомерия, смесь приземленной нейрофизиологии и строгой математики. Она не похожа ни на что, написанное ранее. Принятая в ней нотация сложна, а претензии экстравагантны. Провозглашенная цель – ни больше ни меньше как свести функционирование человеческого мозга к математической логике и таким образом объяснить мышление, память и разум. При всей своей нескромности и наивности, это бурлящий источник идей, питавший компьютерные науки на протяжении нескольких десятилетий.

Авторы изображают нейронную сеть, состоящую из нейронов двух типов. Одни не получают входных сигналов от других нейронов; это носители сенсорной информации, и называются они рецепторами. Другие являются переключателями и могут находиться в одном из двух состояний: возбужден и не возбужден. Система синхронизирована глобальными часами, и состояние каждого нейрона в момент $t + 1$ зависит от состояний его входов в момент t. Входные сигналы нейрону поступают либо от рецепторов, либо с выходов других нейронов (эффекторов); точки соединения, в которых входные сигналы поступают нейрону, называются синапсами. У нейрона Мак-Каллока–Питтса i имеется порог θ_i; нейрон i возбуждается, если возбуждено более θ_i его входов. Однако у нейрона имеется также тормозящий вход, и если он активен, то нейрон не возбуждается. При изображении нейронов (рис. 9.1 ниже) возбуждающими входами являются наклонные стороны треугольной диаграммы, тормозящим входом – точка слева, а выходом, или эффектором, – вертикальная сторона справа.

Короче говоря, в этой статье мозг и все его функции моделируются как цифровая система. Нейроны на рис. 9.1 – это ворота на территорию, которую сегодня мы назвали бы пороговыми логическими схемами. Цель авторов – выяснить, какие вычисления может производить такая сеть. Для этого они ассоциируют с каждым нейроном i предикат $N_i(t)$, принимающий значение истина, если в момент t нейрон i возбужден. После этого введения, но до того, как приступать к чтению статьи, имеет смысл взглянуть на некоторые диаграммы и формулы на рис. 9.1. (Одиночные точки и вертикальные двоеточия – альтернатива скобкам; они разбивают формулы на части, причем две точки сильнее, чем одна. Одна точка может также обозначать конъюнкцию. Таким образом, первая строка части (е) соответствует $N_3(t) \equiv [N_1(t-1) \vee (N_2(t-3) \wedge \sim N_2(t-2))]$, где $\sim$ означает «не», $\vee$ – «или», $\wedge$ – «и».)

Конкретным техническим достижением этой работы является доказательство того, что множество предикатов, вычислимых нейронными сетями, в точности совпадает с множеством предикатов, выразимых в некоторой очень выразительной логической системе. Особое внимание уделено сетям с обратной связью, в которых выход одного нейрона после воздействия на ряд других нейронов подается обратно на вход исходного нейрона. Авторы предполагают, что такие циклы активности объясняют процесс запоминания. Таким образом, говорят они, при наличии полного описания сети «для прогноза история никогда не необходима». Мозг – это то, что сегодня мы назвали бы детерминированным конечным автоматом: его будущее полностью определено текущим состоянием и распространяющимися в прямом направлении входными сигналами.

Приведенные в статье формулы изобилуют ошибками и неудачными обозначениями, например буква «S» обозначает функцию следования, но полужирная «$\boldsymbol{S}$» – совершенно не связанная с этим переменная, означающая «предложение» (sentence). Стивен Коул Клини упростил эту модель и в 1951 году положил ее в основу своей формализации конечных автоматов и регулярных выражений. Читатели, которым «Логическое исчисление» покажется трудным чтением, могут утешаться следующим замечанием Клини: «Настоящая статья частично является переложением результатов Мак-Каллока–Питтса; но та часть их статьи, в которой рассматриваются произвольные нейронные сети, показалась нам малопонятной…» (Kleene, 1951).

Вскоре стало очевидно, что цифровая конструкция, описанная Мак-Каллоком и Питтсом, в качестве модели мозга не годится. Открытие того, что «информация, передаваемая глазом лягушки ее мозгу» (Lettvin et al., 1959), вовсе не является растровым изображением, казалось, повергло в прах надежды Питтса упорядочить логическую картину мира. И тем менее Мак-Каллок и Питтс дали жизнь не только теории конечных автоматов, но и распростирающейся все шире области нейронных вычислений. Их дерзость окупилась сполна – правда, не так, как они рассчитывали.

«Логическое исчисление» – продукт необычного и трагического сотрудничества. Уоррен Мак-Каллок (1898–1969) был нейрофизиологом и жаждал понять, что такое разум, с научной точки зрения. Выходец из семьи успеш-

ных юристов и инженеров, Мак-Каллок скептически относился к теориям Фрейда, преобладавшим в психологии в середине XX века. Он наткнулся на книгу Уайтхэда и Рассела «Основания математики», но не мог увязать ее со своим пониманием анатомии и функционирования нервной системы. Уолтер Питтс (1923–1969), сын грубого рабочего из Детройта, юношей нашел себе прибежище в публичной библиотеке. Он занимался самостоятельно и, в частности, в 12 лет прочитал «Основания» и вступил в переписку с Бертраном Расселом. Несколькими годами позже, узнав, что Рассел читает лекции в Чикагском университете, он сбежал из дома, чтобы уже никогда не возвращаться. Он бродил вокруг университета, где повстречал Мак-Каллока (тут было бы уместно подумать о провидении). И Мак-Каллок, 42-летний профессор, и Питтс, бездомный 18-летний беглец, читали Лейбница и были решительно настроены разработать лейбницево исчисление мышления на прочном математическом и нейроанатомическом фундаменте. Эта статья стала конечным результатом их усилий. Впервые, как впоследствии заявлял Мак-Каллок, «мы знаем, как мы знаем». В последнем разделе высказывается предположение, что после того как получило объяснение функционирование мозга, в будущем умственные расстройства будут интерпретированы как конкретные сбои нейронной сети.

Увы, сам Питтс стал жертвой душевной болезни. И он, и Мак-Каллок закончили свои дни, работая в МТИ – Питтс, никогда не ходивший в школу, аспирантом основоположника кибернетики Норберта Винера (автора главы 19). Трещина в личных отношениях троих мужчин повергла Питтса в бездну депрессии и алкоголизма, от которых он скончался в возрасте 46 лет. Мак-Каллок, бывший на 25 лет старше, пережил его всего на несколько месяцев (Gefter 2015).

9.0. Краткое описание

Поскольку нервная активность подчиняется закону «все или ничего», то нейронные события и соотношения между ними можно изучать средствами логики высказываний. Оказывается, что поведение любой сети может быть описано в этих терминах с привлечением более сложных логических средств для сетей, содержащих петли. Для всякого логического выражения, удовлетворяющего некоторым условиям, можно найти сеть, имеющую описываемое этим выражением поведение. Показано, что различные выборы возможных нейрофизиологических предположений эквивалентны в том смысле, что для сети, действующей согласно одному предположению, существует другая сеть, действующая согласно другому и дающая те же результаты, хотя, быть может, не за то же самое время. Обсуждаются различные приложения исчисления.

9.1. Введение

Теоретическая нейрофизиология базируется на нескольких основных предпосылках. Нервная система является сетью нейронов, каждый из которых имеет тело и аксон. Места контакта нейронов, или синапсы, находятся всегда между аксоном одного и телом другого нейрона. В каждый момент нейрон имеет известный порог, который должно превзойти возбуждение, чтобы вызвать нервный импульс. Все это, если не считать самого факта и момента появления импульса, определено нейроном, а не возбуждением. От точки возбуждения импульс распространяется по всему нейрону. Скорость распространения импульса по аксону пропорциональна диаметру аксона и варьируется от менее 1 м/с в тонких аксонах, которые бывают обычно короткими, до более чем 150 м/с в толстых аксонах, обычно длинных. Время распространения импульса по аксону не играет, следовательно, большой роли при определении времени прибытия импульсов в точки, удаленные от одного и того же источника на разные расстояния. Возбуждение передается через синапсы преимущественно от окончания аксона к телу. Остается спорным вопрос, обусловлено ли это необратимостью отдельных синапсов или же преобладанием некоторых анатомических конфигураций. Последнее предположение не требует гипотез *ad hoc* и объясняет известные исключения; однако любое предположение о причине такого явления совместимо с нижеследующим исчислением. Не известно ни одного случая, когда возбуждение, приходящее через один синапс, вызвало бы импульс в каком-либо нейроне, тогда как любой нейрон может быть возбужден импульсами, приходящими через достаточное число соседних синапсов в течение латентного периода, который продолжается менее четверти миллисекунды. Наблюдаемое временное суммирование импульсов в более длительных интервалах времени для отдельных нейронов невозможно и эмпирически зависит от структурных свойств сети. Между прибытием импульсов к нейрону и распространением собственного импульса нейрона имеется синаптическая задержка, большая половины миллисекунды. В начале нервного импульса нейрон абсолютно индифферентен к любому возбуждению. Затем его возбудимость быстро восстанавливается, достигая в некоторых случаях сверхнормального уровня, после чего снова становится несколько ниже нормальной, а затем медленно возвращается к нормальному уровню. Частые возбуждения усиливают понижение порога возбудимости ниже нормального уровня. Эти особенности связаны со временем и местом появления нервных импульсов и не связаны ни с какими другими особенностями действий нейронов. Из последних одно лишь явление торможения приводилось в качестве серьезного довода против этого тезиса. Торможение есть прекращение или предотвращение активности одной группы нейронов посредством одновременной или предшествующей активности другой группы. До последнего времени это могло быть объяснено предположением, что предшествующая активность нейронов второй группы может настолько повысить пороги вставочных нейронов, что последние будут не в состоянии возбудиться от нейронов первой группы, тогда как импульсы нейронов первой группы должны складываться с импульсами вставочных нейронов для возбуждения нейронов, теперь тор-

мозимых. В настоящее время показано, что в некоторых случаях торможение происходит менее чем за одну миллисекунду. Это исключает участие вставочных нейронов и требует существования синапсов, импульсы через которые тормозят нейрон, возбуждаемый импульсами через другие синапсы. До сих пор эксперимент не обнаружил, является ли эта невозбудимость относительной или абсолютной. Мы предположим последнее и покажем, что разница несущественна для наших рассуждений. Каждый из видов невозбудимости может быть объяснен любым из двух способов: «тормозящий синапс» может обладать способностью вырабатывать вещество, повышающее порог нейрона, или же этот синанс может быть расположен таким образом, что локальное нарушение, производимое его возбуждением, препятствует изменению, вызываемому синапсами, которые в противном случае возбуждали бы нейрон. Поскольку в случае электрического возбуждения уже известны условия, при которых подобный эффект имеет место, то первая гипотеза должна быть отвергнута, если и до тех пор пока не будет подтверждена, ибо вторая возможность не предполагает никаких новых гипотез. Мы имеем тогда два объяснения торможения, основанных на одной и той же общей предпосылке и отличающихся только в отношении рассматриваемых нервных сетей и, следовательно, в отношении времени, потребного на торможение. В дальнейшем мы будем говорить о таких нервных сетях как об эквивалентных *в широком смысле*. Поскольку мы рассматриваем свойства сетей, инвариантные относительно этой эквивалентности, мы можем делать физические допущения, наиболее удобные для нашего исчисления.

Много лет назад один из нас, из соображений, не относящихся к этому рассуждению, пришел к выводу, что реакция любого нейрона фактически эквивалентна высказыванию, описывающему адекватный стимул. Вследствие этого он попытался описать поведение сложных сетей в терминах символической логики высказываний. Закона нервной активности «все или ничего» достаточно для того, чтобы возбуждение любого нейрона могло быть представлено как некоторое высказывание. Физиологические связи, существующие между нервными активностями, соответствуют, конечно, связям между этими высказываниями. Полезность такого представления зависит от тождественности физиологических связей и связей логики высказываний. Каждой реакции любого нейрона соответствует утверждение некоторого простого высказывания. В свою очередь, оно влечет или некоторое другое простое высказывание, или дизъюнкцию, или конъюнкцию, с отрицанием или без отрицания, аналогичных высказываний, согласно конфигурации синапсов и порогу данного нейрона. Появляются две трудности. Первая относится к явлениям облегчения и утомления, при которых предшествующая активность временно изменяет реакцию одной и той же части сети на последующий стимул. Вторая трудность касается явления обучения, при котором одновременные активности в предшествующее время изменили сеть так, что стимул, прежде бывший недостаточным, становится теперь достаточным. Однако мы можем сети, подверженные обоим изменениям, заменить эквивалентными фиктивными сетями, составленными из нейронов с неизменяемыми связями и порогами. Но одно должно быть абсолютно ясным: мы не смешиваем формальную эквивалентность с фактическим истолкованием.

Per contra! – мы рассматриваем облегчение и утомление как зависящие от непрерывных изменений порога, связанных с электрическими и химическими переменными, такими как следовые потенциалы и концентрации ионов; обучение же мы рассматриваем как длительное изменение, могущее перенести сон, анестезию, конвульсии и кому. Значение формальной эквивалентности состоит в следующем: изменения, фактически лежащие в основе облегчения, утомления и обучения, никоим образом не затрагивают выводов, следующих из формальной трактовки активности нервных сетей, и связи соответствующих высказываний остаются связями логики высказываний.

Нервная система содержит много циклических путей. Их активность так регенерирует возбуждение всех участвующих в них нейронов, что связь с прошлым становится неопределенной, хотя при этом все же предполагается, что афферентная активность со временем реализовала один из классов конфигураций. Точная спецификация этих зависимостей посредством рекурсивных функций и определение тех из них, которые могут быть воплощены в активности нервных сетей, завершают теорию.

9.2. Теория: сети без петель

Примем следующие физические допущения для нашего исчисления.

1. Активность нейрона удовлетворяет принципу «все или ничего».
2. Возбуждению нейрона в какой-либо момент времени должен предшествовать латентный период накопления возбуждений определенного фиксированного числа синапсов. Это число не зависит от предыдущей активности и от расположения синапсов на нейроне.
3. Единственным запаздыванием в нервной системе, имеющим значение, является синаптическая задержка.
4. Активность какого-либо тормозящего синапса абсолютно исключает возбуждение данного нейрона в рассматриваемый момент времени.
5. С течением времени структура сети не изменяется.

Наиболее подходящим символизмом для изложения нашей теории является карнаповский «Язык II» [1937], дополненный различными обозначениями из «Оснований» [1910] Рассела и Уайтхеда (включая соглашение об употреблении точек). Для обозначения квантора существования мы используем неперевернутое E, для обозначения импликации – стрелку $\rightarrow$. Мы используем также синтаксические обозначения Карнапа; при этом, однако, готический шрифт заменен полужирным. Мы вводим функтор S, значение которого для некоторого свойства P определяется как такое свойство чисел, которое имеет место для некоторого числа, при условии что P имеет место для предшествующего числа. Функтор S определен соотношением $S(P)(t) \mathbf{.\equiv.} P(x) \mathbf{.}\ t = x'$ [Примечание редактора: в оригинале исправлена ошибка.]; скобки в его аргументе часто будем опускать – в таком случае под аргументом понимается ближайшее справа предикатное выражение [$\boldsymbol{Pr}$]. Кроме того, мы пишем $S^2\boldsymbol{Pr}$ вместо $S(S(\boldsymbol{Pr}))$ и т. д.

Нейроны данной сети $\mathcal{N}$ можно обозначить c_1, c_2, ..., c_n, а свойство чисел «нейрон c_i возбуждается в некоторый момент» (равный числу синаптических задержек от начала отсчета времени) – N с индексом i, так что $N_i(0)$ означает утверждение «c_i возбужден в момент t». Назовем N_i *действием* c_i. Иногда мы будем рассматривать индексированное N, как если бы оно принадлежало объектному языку и заменяло функторный аргумент. Тогда его можно было бы заменить числовой переменной [z] и квантифицировать; это позволит нам сокращать длинные (но конечные) дизъюнкции и конъюнкции посредством оператора. Мы будем применять такие выражения всюду, где встречаются последовательности из ***Pr***; этого можно достигнуть формально с помощью очевидного дизъюнктивного определения. Предикаты N_1, N_2, ... образуют синтаксический класс ***N***.

Определим рецепторы сети $\mathcal{N}$ как такие нейроны из $\mathcal{N}$, которые не имеют на себе аксонов. Обозначим действия этих нейронов $\boldsymbol{N}_1$, ..., $\boldsymbol{N}_p$, действия же остальных нейронов – $\boldsymbol{N}_{p+1}$, $\boldsymbol{N}_{p+2}$, ..., $\boldsymbol{N}_n$. Тогда *решением* сети $\mathcal{N}$ будет класс высказываний вида $\boldsymbol{S}_i$: $N_{p+1}(z_1)$ **.** $\equiv$ **.** $\boldsymbol{Pr}_i$ ($\boldsymbol{N}_1$, $\boldsymbol{N}_2$, ..., $\boldsymbol{N}_p$, z_1), где $\boldsymbol{Pr}_i$ не содержит свободных переменных, кроме z_1, и описательных символов, кроме ***N***, в аргументе [***Arg***] и, возможно, содержит еще постоянные высказывания [***sa***], причем каждое $\boldsymbol{S}_i$ верно для $\mathcal{N}$. Обратно, пусть дано некоторое высказывание $\boldsymbol{Pr}_1({}^1p_1^1, {}^1p_2^1, ..., {}^1p_p^1, z, s)$, не содержащее свободных переменных, за исключением тех, что находятся в аргументе ***Arg***. Будем говорить, что оно *реализуемо в узком смысле*, если имеется такая сеть $\mathcal{N}$ и в ней такая последовательность $\boldsymbol{N}_i$, что $N_1(z_1)$ **.** $\equiv$ **.** $\boldsymbol{Pr}_i$ ($\boldsymbol{N}_1$, $\boldsymbol{N}_2$, ..., z_1, $\boldsymbol{sa}_1$)) истинно, где $\boldsymbol{sa}_1$ имеет вид $\boldsymbol{N}(0)$. Назовем такое высказывание *реализуемым в широком смысле*, или просто *реализуемым*, если для некоторого n высказывание $S^n(\boldsymbol{Pr}_1)(p_1, ..., p_p, z_1, \boldsymbol{s})$ реализуемо в вышеуказанном смысле. Нейрон c_{pi} является тогда реализующим нейроном. Будем говорить, что два закона нервного возбуждения эквивалентны в узком или широком смысле, если каждое ***S***, реализуемое в каком-либо смысле при допущениях одного закона, реализуемо в соответствующем смысле другой сетью при допущениях другого закона.

В приводимых далее теоремах под реализацией понимается реализация в широком смысле. В некоторых случаях можно получить более точные теоремы о реализации в узком смысле. Однако, помимо того что это привело бы к усложнению формулировок, получение таких теорем не имело бы большой практической ценности, ибо уровень современной нейрофизиологии позволяет определить закон возбуждения с точностью до эквивалентности в широком смысле, а более точные теоремы принимают различную форму в зависимости от сделанных допущений. А наши менее точные теоремы инвариантны по отношению к эквивалентности и достаточны для всех целей, в которых точное время прохождения импульса через всю сеть несущественно.

Мы можем теперь точно сформулировать центральные проблемы: во-первых, найти эффективный метод получения тех ***S***, которые образуют решение заданной сети; во-вторых, охарактеризовать эффективным образом класс реализуемых ***S***. Содержательно проблемы заключаются в определении поведения произвольных сетей и в нахождении сети, имеющей предписанное поведение, если таковая существует.

Назовем сеть циклической, если она содержит некоторую петлю: т. е. если в ней существует цепочка c_i, c_{i+1}, ... нейронов, каждый член которой имеет аксоны на следующем по порядку нейроне и начало которой совпадает с концом. Если система нейронов $c_1, c_2, \ldots, c_p$ такова, что ее удаление превращает $\mathcal{N}$ в сеть без петель, и если никакая меньшая система нейронов этим свойством не обладает, то эта *система* называется циклической, а число нейронов в ней называется *порядком* $\mathcal{N}$. Как мы увидим далее, порядок сети является в некотором смысле показателем сложности ее поведения. В частности, сети порядка нуль имеют особенно простые свойства; мы рассмотрим их в первую очередь.

Определим *временное пропозициональное выражение* (в.п.в.), обозначающее временную пропозициональную функцию (в.п.ф.), путем следующей рекурсии:

1) ${}^1\boldsymbol{p}^1[z_1]$ есть в.п.в., где $\boldsymbol{p}_1$ – предикатная переменная;
2) если $\boldsymbol{S}_1$ и $\boldsymbol{S}_2$ – в.п.в., содержащие одни и те же свободные индивидуальные переменные, то в.п.в. будут также выражения $S\boldsymbol{S}_1$, $\boldsymbol{S}_1 \vee \boldsymbol{S}_2$, $\boldsymbol{S}_1 \,.\, \boldsymbol{S}_2$ и $\boldsymbol{S}_1 \,.\, \sim\boldsymbol{S}_2$;
3) ничто другое не является в.п.в.

Теорема 1. *Каждая сеть порядка 0 может быть решена в терминах временных пропозициональных выражений.*
Пусть c_i – произвольный нейрон сети $\mathcal{N}$ с порогом $\theta_i > 0$; пусть $c_{i1}, c_{i2}, \ldots, c_{ip}$ имеют на нем соответственно $n_{i1}, n_{i2}, \ldots, n_{ip}$ возбуждающих синапсов и пусть $c_{j1}, c_{j2}, \ldots, c_{jq}$ имеют на нем тормозящие синапсы. Пусть κ_i – система из $\{n_{i1}, n_{i2}, \ldots, n_{ip}\}$ подклассов такая, что сумма их членов превосходит θ_i. Тогда в соответствии с вышеупомянутыми допущениями мы можем написать

$$\boldsymbol{N}_i(\boldsymbol{z}_1) \,.\equiv.\, S\left\{\prod_{m=1}^{q} \sim \boldsymbol{N}_{jm}(\boldsymbol{z}_1) \,.\, \sum_{\alpha\in\kappa_i} \prod_{s\in\alpha} \boldsymbol{N}_{is}(\boldsymbol{z}_1)\right\}, \tag{9.1}$$

где $\sum$ и $\prod$ – синтаксические символы для конечных дизъюнкций и конъюнкций. Так как выражение такого вида можно написать для каждого c_i, не являющегося рецептором, то, подставляя в (9.1) соответствующие выражения для всех $\boldsymbol{N}_{jm}$ или $\boldsymbol{N}_{is}$, нейроны которых не есть рецепторы, и повторяя этот процесс, мы придем, наконец, к выражению для $\boldsymbol{N}_i$ полностью в терминах рецепторов $\boldsymbol{N}$, ибо сеть $\mathcal{N}$ не содержит петель. Кроме того, это выражение будет в.п.в., так как (9.1) есть, очевидно, в.п.в. и так как из нашего определения немедленно вытекает, что результат подстановки в.п.в. вместо $\boldsymbol{p}(z)$ в в.п.в. является в.п.в.

Теорема 2. *Каждое в.п.в. реализуемо сетью порядка 0.*
Функтор S, очевидно, перестановочен с дизъюнкцией, конъюнкцией и отрицанием. Очевидно также, что результат подстановки всякого $\boldsymbol{S}_i$, реализуемого в узком смысле (в у.с.), вместо $\boldsymbol{p}(z)$ в реализуемое выражение $\boldsymbol{S}_1$ является выражением, реализуемым в у.с. Реализующая сеть строится путем замены рецепторов сети для $\boldsymbol{S}_1$ реализующими нейронами сетей для $\boldsymbol{S}_i$; сеть из одного нейрона реализует $\boldsymbol{p}_1(z_1)$ в у.с.; на рис. 9.1а изображена сеть, реализующая $S\boldsymbol{p}_1(z_1)$ и, следовательно, $S\boldsymbol{S}_2$ в у.с., если $\boldsymbol{S}_2$ может быть реали-

зована в у.с. Далее, если $\boldsymbol{S}_2$ и $\boldsymbol{S}_3$ реализуемы, то $S^m\boldsymbol{S}_2$ и $S^n\boldsymbol{S}_3$ реализуемы в у.с. при подходящих m и n. Этим же свойством обладают $S^{m+n}\boldsymbol{S}_2$ и $S^{m+n}\boldsymbol{S}_3$. Сети на рис. 9.1b–d реализуют соответственно $S(\boldsymbol{p}_1(z_1) \vee \boldsymbol{p}_2(z_1))$, $S(\boldsymbol{p}_1(z_1) \,.\, \boldsymbol{p}_2(z_1))$ и $S(\boldsymbol{p}_1(z_1) \,.\, \sim\boldsymbol{p}_2(z_1))$ в у.с. Значит, $S^{m+n+1}(\boldsymbol{S}_1 \vee \boldsymbol{S}_2)$, $S^{m+n+1}(\boldsymbol{S}_1 \,.\, \boldsymbol{S}_2)$ и $S^{m+n+1}(\boldsymbol{S}_1 \,.\, \sim\boldsymbol{S}_2)$ реализуемы в у.с., а следовательно, $\boldsymbol{S}_1 \vee \boldsymbol{S}_2$, $\boldsymbol{S}_1 \,.\, \boldsymbol{S}_2$ и $\boldsymbol{S}_1 \,.\, \sim\boldsymbol{S}_2$ реализуемы, если реализуемы $\boldsymbol{S}_1$ и $\boldsymbol{S}_2$. Полная индукция приводит к заключению, что всякое в.п.в. реализуемо. Таким образом, можно считать, что каждая сеть строится из основных элементов на рис. 9.1a–d в точности так же, как соответствующее в.п.в. порождается операциями предшествования, дизъюнкции, конъюнкции и отрицания с конъюнкцией. В частности, для всякого описания состояния, т. е. распределения значений истинности и ложности действий всех нейронов некоторой сети, за исключением тех, которые делают всех ложными, можно построить некоторый нейрон, возбуждение которого необходимо и достаточно для справедливости этого описания. Кроме того, всегда существует бесконечно много топологически различных сетей, реализующих произвольное в.п.в. <...>

Кажется, что феномены обучения, имеющие устойчивый характер при большинстве физиологических изменений в нервной активности, требуют возможности постоянного изменения структуры сети. Простейшее изменение такого рода – образование новых синапсов или эквивалентное ему понижение порога. Предположим, что некоторые окончания аксонов не могут сначала возбудить следующий нейрон, но если в какой-нибудь момент нейрон возбуждается одновременно с этими окончаниями, то они превращаются в обычные синапсы, способные в дальнейшем возбуждать нейрон. Устранение тормозящего синапса дает совершенно эквивалентный результат. Тогда имеем следующую теорему.

Теорема 7. *Изменяемые синапсы могут быть заменены петлями.*

Это выполняется с помощью метода рис. 9.1i. Следует также отметить, что нейрон, который становится и остается активным самопроизвольно, может быть аналогичным образом заменен петлей, приводимой в активность посредством одного рецептора в начале активности и тормозимой другим при ее прекращении. [Примечание редактора: на рис. 9.1 выражение для верхней части (f) было исправлено, а выражение для верхней части (i) в оригинале отсутствует. Часть (g) загадочна – верхняя диаграмма представляет собой вариант (d), но выражение для (g) не соответствует никакой части диаграммы (g).]

(a) $N_2(t) \,.\equiv.\, N_1(t-1)$
(b) $N_3(t) \,.\equiv.\, N_1(t-1) \vee N_2(t-1)$
(c) $N_3(t) \,.\equiv.\, N_1(t-1) \,.\, N_2(t-1)$
(d) $N_3(t) \,.\equiv.\, N_1(t-1) \,.\, \sim N_2(t-1)$
(e) $N_3(t) :\equiv: N_1(t-1) \,.\vee.\, N_2(t-3) \,.\, \sim N_2(t-2)$
$N_4(t) \,.\equiv.\, N_2(t-2) \,.\, N_2(t-1)$
(f) $N_4(t) :\equiv: \sim N_1(t-1) \,.\, N_2(t-1) \vee N_3(t-1)$
$N_4(t) :\equiv: \sim N_1(t-2) \,.\, N_2(t-2) \vee N_3(t-2) \,.\vee.\, N_1(t-2) \,.\, N_2(t-2) \,.\, N_3(t-2)$
(g) $N_3(t) \,.\equiv.\, N_2(t-2) \,.\, \sim N_1(t-3)$
(h) $N_2(t) \,.\equiv.\, N_1(t-1) \,.\, N_1(t-2)$
(i) $N_3(t) :\equiv: N_2(t-1) \,.\vee.\, N_1(t-1) \,.\, (Ex)\; t-1 \,.\, N_1(x) \,.\, N_2(x)$

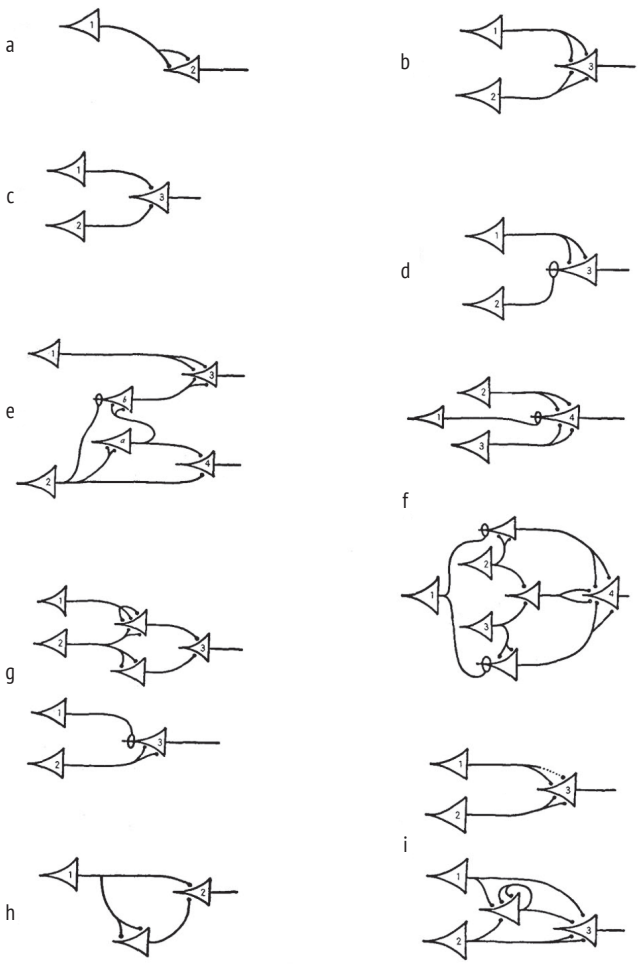


Рис. 9.1. Нейрон c_i всегда обозначается числом i над телом, а соответствующее действие обозначается N_i, как в тексте

9.3. Теория: сети с петлями

Рассмотрение сетей, не удовлетворяющих нашему предыдущему предположению об отсутствии петель, является гораздо более трудным. В значительной мере это связано с возможностью неограниченной по времени циркуляции активности, сообщенной петле, так что реализуемый предикат ***Pr*** может относиться к событиям неограниченно далекого прошлого. <...> [Примечание редактора: выражение $(Ex)\, t-1 \,.\, N_1(x) \,.\, N_2(x)$ (в части (i) рис. 9.1) означает, что в прошлом существовал момент x не позднее $t-1$, когда $N_1(x)$ и $N_2(x)$ одновременно были истинны. Непомеченный нейрон – который получает обратную связь от самого себя – останется в возбужденном состоянии неопределенно долго.]

В заключение следует отметить еще одно обстоятельство. Легко показать, что, во-первых, каждая сеть, снабженная лентой, считывающими устройствами, связанными с рецепторами, и подходящими эффекторами для выполнения необходимых моторных операций, может вычислять лишь такие числа, которые вычисляет машина Тьюринга; во-вторых, что каждое из таких чисел может быть вычислено такой сетью и что сети (без считывающего устройства и ленты) с петлями могут вычислять некоторые вычислимые числа и никакие другие, но не все из них. Это представляет интерес с точки зрения психологического оправдания тьюринговского определения вычислимости и его эквивалентов, λ-определимости Чёрча и примитивной рекурсивности Клини: если какое-либо число может быть вычислено организмом, то оно вычислимо также по этим определениям, и обратно.

9.4. Следствия

Причинность, требующая описания состояний и закона необходимой связи между ними, проявляется в различных формах во многих науках, но нигде, за исключением статистики, она не является столь необратимой, как в теории нервной активности. Задание в произвольный момент времени рецепторных возбуждений и активности (удовлетворяющей принципу «все или ничего») всех составляющих нейронов определяет состояние. Задание нервной сети определяет закон необходимой связи, с помощью которого по описанию любого состояния можно вычислить следующее за ним состояние, однако включение дизъюнктивных соотношений не позволяет полностью определить предшествующее состояние. Кроме того, восстанавливающая активность входящих в сеть петель делает связь с прошлым неопределенной. Так, наше знание мира, включая нас самих, неполно в пространственном отношении и неопределенно в отношении времени. Это незнание, касающееся неявно всех наших умственных способностей, является обратной стороной абстракции, делающей наше знание полезным. Роль умственных способностей при определении эпистемологической связи наших теорий с нашими наблюдениями и наших наблюдений с фактами совершенно ясна, ибо очевидно, что каждая идея и каждое ощущение реализуются активностью внутри нервной сети и что действительные возбуждения рецепторов не определены полностью никакой такой активностью.

Не имеется никакой теории или наблюдения, которые могли бы сохранить нечто большее, кроме их старого неполного отношения к фактам, если сеть изменяется. Появляются звон в ушах, мурашки, галлюцинации, иллюзии, смешение ощущений и дезориентация. Опыт, следовательно, подтверждает, что если наши сети не определены, то неопределенными являются и наши факты, и «реальности» мы не можем приписать ничего большего, чем одно качество или «форму». С определением сети непознаваемый объект знания, «вещь в себе», перестает быть непознаваемым.

В психологии, как бы она ни определялась, описание сети дало бы все, что может быть достигнуто в этой области, даже если анализ дошел бы до конеч-

ных психических единиц, или «психонов», ибо психон не может быть ничем меньшим, чем активностью отдельного нейрона. Так как эта активность по своей природе пропозициональна, то все психические события имеют преднамеренный, или «семиотический», характер. Закон «все или ничего» этой активности и соответствие ее связей связям логики высказываний обеспечивают то, что связи психонов суть связи двузначной логики высказываний. Следовательно, в психологии, интроспективной, бихевиористской или физиологической, основными являются связи двузначной логики.

Отсюда возникают конструктивные решения холистических проблем, включающих в себя дифференцированный континуум чувственных ощущений и нормативные, совершенствующие и разрушающие свойства восприятия и исполнения. Из необратимости причинности следует, что даже если сеть известна, то хотя по настоящей активности мы можем предсказать будущее, но не можем определить ни афферентное по центральному, ни центральное по эфферентному, ни прошлое по настоящей активности. Эти заключения подкрепляются фактами существования противоречивых свидетельств очевидцев, трудностью дифференцированного диагностирования органических больных, истерики и симуляции и сравнением памяти и воспоминаний с записями. Более того, системы, которые реагируют на различие между афферентами регенеративной сети и некоторой активностью в этой сети так, что уменьшают это различие, обнаруживают целевое поведение. Известно, что организмы обладают многими такими системами, обслуживающими гомеостазис, желание и внимание. Таким образом, как формальный, так и конечный аспекты этой активности, которую мы обычно называем умственной, строго выводимы из современной нейрофизиологии. Психиатр может найти утешение в очевидном заключении, касающемся причинности, именно что для прогноза история никогда не необходима. Он может извлечь немногое из того равным образом справедливого вывода, что наблюдаемое им объяснимо лишь в терминах нервной активности, бывшей до последнего времени вне его кругозора. Центральным моментом этого незнания является неоднозначность перехода от произвольного образца видимого поведения к нервным сетям, тогда как из воображаемых сетей существует фактически только одна, и она может в любой момент обнаружить непредвиденную активность. Разумеется, для психиатра большее значение имеет то обстоятельство, что в таких системах «разум» уже не бродит «более призрачно, чем призрак». Напротив, расстройство ума можно изучать, без потери общности или строгости, в научных терминах нейрофизиологии. В нейрологии теория нервных сетей заостряет различие между сетями, необходимыми или только достаточными для заданной активности, выясняя, таким образом, соотношения между нарушенной структурой и нарушенным функционированием. В собственной области этой теории различие между эквивалентными сетями и сетями, эквивалентными в узком смысле, указывает на подходящее использование и на важность изучения нервной деятельности во времени. Математической биофизике эта теория доставляет некоторый способ строгой символической трактовки известных сетей и легкий метод конструирования гипотетических сетей с требуемыми свойствами.

10 Первая редакция отчета о EDVAC (1945)

Джон фон Нейман

«Архитектура фон Неймана», описанная, но не названная так в этом отчете, имеет ту же логическую структуру, что и «универсальная вычислительная машина», описанная Тьюрингом (1936, здесь глава 6), хотя сходство, по-видимому, является чем-то случайным (см. стр. 18). Данные, которыми оперирует программа, хранятся в той же памяти, что и сама программа (стр. 133, 145). Алан Тьюринг ссылается на этот отчет, но не на собственную теоретическую работу, в своем плане Автоматической вычислительной машины (Turing, 1945).

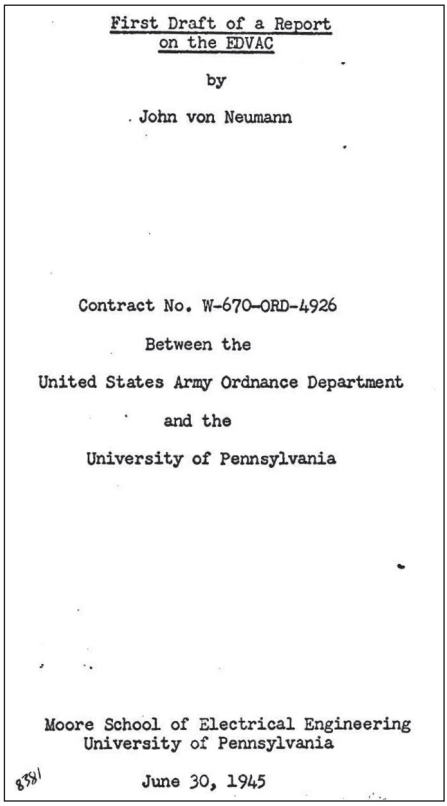
В 1944 году Артур Бёркс (1915–2008) и Герман Голдстайн (1913–2004) входили в группу под руководством Преспера Эккерта и Джона Мочли, которая занималась проектированием электронного компьютера, известного под названием EDVAC, в Электротехнической школе Мура при Пенсильванском университете. Это был преемник машины ENIAC Эккера–Мочли, которая уже трудилась над баллистическими расчетами в интересах армии США. После того как Голдстайн случайно встретился со знаменитым математиком Джоном фон Нейманом (1903–1957) и рассказал ему о проекте EDVAC, фон Нейман присоединился к группе. В этом меморандуме Джон фон Нейман описал проект; Голдстайн набрал и опубликовал его в 1945 году, указав в качестве единственного автора фон Неймана. Это достойно сожаления: документ принадлежал перу фон Неймана, но проект был заслугой всей группы, а хранимая программа уже была частью ENIAC (хотя в той машине программа размещалась в постоянной памяти, доступной только для чтения).

На рис. 10.1 показана титульная страница машинописной копии, очень плохо исполненной. Всюду, где возможно, мы добавили перекрестные ссылки на разделы, которые в оригинале были опущены с намерением вставить позже. Исправленный и дополненный отчет (Burks et al. 1947) вышел в 1946 году

под названием «Предварительное обсуждение логического проектирования электронного вычислительного прибора», и в качестве соавторов были добавлены Голдстайн и Бёркс.

Эти отчеты задали направление проектирования компьютеров, которое в конечном итоге привело к возникновению индустрии программного обеспечения, поскольку программы стало возможно обрабатывать и загружать в память, как любые другие данные. В отчетах анализируется разделение устройства на память и блоки управления («органы») с использованием двоичной нотации, а в более позднем «Предварительном обсуждении» добавлены объяснения адресов памяти («номера ячеек»), разного вида регистров, представления чисел с плавающей точкой и использование первых шести букв латиницы в качестве шестнадцатеричных цифр. Но в краткосрочной перспективе непризнание в отчете вклада других членов привело к разладу среди основоположников группы – в особенности был обижен Мочли, являвшийся автором предложения, на основе которого финансировались и разрабатывались EDVAC и его предшественник ENIAC. После войны Мочли и Эккерт отделились, основали свой компьютерный бизнес, а в 1947-м подали патентную заявку, которая в конечном итоге была отклонена из-за того, что этот отчет был опубликован раньше. (Последующая история компании Eckert–Mauchly Computer Corporation кратко описана на стр. 226.) Бёркс и Голдстайн продолжили академическую карьеру и присоединились к фон Нейману в Принстоне.

К моменту, когда фон Нейман присоединился к группе EDVAC, он уже был одним из самых авторитетных математиков в мире и внес вклад в различные области математики, математической экономики и квантовой механики. Родившись и получив образование в Венгрии, он в 1930 году переехал в Берлин, где вошел в ближний круг Давида Гильберта (см. главу 5). Перед началом Второй мировой войны он эмигрировал в США и внес важный вклад в Манхэттенский проект, т. е. в проектирование атомной бомбы. Он видел, как тысячи людей-вычислителей, пользуясь индивидуальными механическими калькуляторами, трудятся над решением выведенных им уравнений ядерной бомбы, и, хотя не мог обсуждать секретную работу с группами коллег в школе Мура или в Принстоне, это послужило сильным стимулом для разработки быстродействующих автоматических компьютеров.


First Draft of a Report
on the EDVAC

by

John von Neumann

Contract No. W-670-ORD-4926

Between the

United States Army Ordnance Department

and the

University of Pennsylvania

Moore School of Electrical Engineering
University of Pennsylvania

June 30, 1945

Рис. 10.1. Титульная страница машинописной копии «Первой редакции»

Терминология, употребляемая в «Первой редакции», иногда зависит от реализации, и это сбивает с толку. Память представляла собой линию задержки, организованную в виде того, что сегодня мы назвали бы 32-битовыми словами, – фон Нейман называет их «малыми циклами». Один из 32 бит был отведен под флаг, показывающий, представляет слово число или инструкцию («команду»). В числах еще один бит был отведен под знак, а остальные представляли двоичную дробь от –1 до 1. 32 малых цикла составляли «большой цикл», или орган линии задержки (delay line organ – DLA). Вся память состояла из 256 DLA, поэтому для задания адреса числа в памяти нужно было 8 бит, чтобы определить DLA, и еще 5 бит для выделения малого цикла.

Для достижения нужного уровня абстракции в «Первой редакции» упоминается бистабильный «элемент», который может быть физически реализован по-разному. В § 10.4.2 фон Нейман приводит довольно длинные цитаты из Мак-Каллока и Питтса, но ни разу не упоминает Тьюринга. В опущенном разделе отчета отмечается, что «элемент, который возбуждает себя, будет сохранять стимул неопределенно долго» – точно так в «Логическом исчислении» сказано о нейронах (стр. 124). А использование фон Нейманом схем, синхронизированных задатчиком времени, напоминает дискретную временную переменную, которую Мак-Каллок и Питтс применяли к нейронным схемам, чтобы подвергнуть их поведение логическому анализу. На самом деле аналогия между вакуумной лампой и нейроном мало используется в проекте, хотя для фон Неймана она представляла интерес в его размышлениях о вычислительных способностях мозга, а программа EDVAC хранилась в быстрой памяти, а не на картах или ленте, как в Mark I Эйкена, просто для того, чтобы к соседним инструкциям можно было обращаться со скоростью, характерной для электронного оборудования.

Обсуждение нейронных вычислений, имеющееся в «Первой редакции», было опущено в «Предварительном обсуждении», но фон Нейман никогда не утрачивал интереса к идее мозга как компьютера и продолжал разрабатывать ее, умирая от рака в возрасте 53 лет. Книга «Компьютер и мозг» (von Neumann, 2000) была издана посмертно.

10.1. Определения

10.1.1. Все обсуждаемое ниже относится к конструкции *высокоскоростной автоматической цифровой вычислительной системы* и, в частности, к ее *логическому управлению*. Прежде чем переходить к конкретным деталям, уместно дать общие пояснения относительно этих концепций.

10.1.2. *Автоматическая вычислительная система* – это (обычно состоящее из большого числа компонентов) устройство, которое может выполнять инструкции для осуществления вычислений произвольной сложности – например, численное решение нелинейного дифференциального уравнения с двумя или тремя независимыми переменными.

Инструкции, управляющие этой работой, должны быть даны устройству в исчерпывающей форме. Они включают всю числовую информацию, необходимую для решения рассматриваемой задачи: начальные и граничные значения зависимых переменных, значения фиксированных параметров (констант), таблицы фиксированных функций, встречающихся в постановке задачи. Эти инструкции должны быть поданы в форме, воспринимаемой устройством: пробиты на перфокартах или на телетайпной ленте, записаны на намагниченной стальной ленте или проволоке, фотографически засняты на кинопленку, реализованы с помощью проводов на одной или более фиксированной или сменной коммутационной панели – этот список ни в коем случае не является полным. Все эти процедуры нуждаются в использовании какого-то кода для выражения логического и алгебраического определения рассматриваемой задачи, а равно необходимого числового материала (см. выше).

После того как эти инструкции переданы устройству, оно должно иметь возможность выполнить их полностью, без какого-либо вмешательства со стороны человека. По завершении затребованных операций устройство должно сохранить результаты в одной из вышеупомянутых форм. Результаты являются числовыми данными; это специфицированная часть числового материала, порождаемого устройством в процессе выполнения вышеупомянутых команд.

10.1.3. Следует, однако, отметить, что в общем случае устройство производит гораздо больше числового материала (чтобы достичь результатов), чем вышеупомянутые (конечные) результаты. Поэтому лишь малую долю выходных числовых результатов необходимо записывать, как указано в § 10.1.2, остальные будут циркулировать только внутри устройства и не выводиться в виде, доступном восприятию человека. Этот момент будет далее рассмотрен более подробно. <…>

10.1.4. Замечания в § 10.1.2 о желательности автоматического функционирования устройства, естественно, предполагают, что оно функционирует без ошибок. Однако вероятность отказа любого устройства всегда конечна, а в случае сложного устройства и длинной последовательности операций может оказаться невозможным сделать эту вероятность пренебрежимо малой. Любая ошибка опорочит весь вывод устройства. Чтобы распознать и исправить такие ошибки, в общем случае может оказаться необходимым вмешательство человека.

Однако до некоторой степени таких явлений можно избежать. Устройство может распознать наиболее частые отказы, индицировать их наличие и место видимыми сигналами, а затем остановиться. При некоторых условиях оно даже могло бы автоматически выполнить необходимые корректирующие действия и продолжить работу (см. § 10.3.3).

10.2. Основные части системы

10.2.1. В ходе анализа функционирования сложного устройства на ум сразу приходят некоторые классифицирующие различия.

10.2.2. Первое: поскольку устройство является в первую очередь компьютером, оно будет чаще всего выполнять элементарные арифметические операции. Это сложение, вычитание, умножение и деление: +, –, ×, ÷. Поэтому разумно предположить, что оно должно содержать специализированные органы для таких операций.

Следует, однако, заметить, что хотя этот общий принцип, вероятно, справедлив, конкретный способ его воплощения нуждается в более пристальном изучении. Даже приведенный выше список операций – +, –, ×, ÷ – нельзя считать бесспорным. Его можно расширить, включив такие операции, как $\sqrt{}$, $\sqrt[3]{}$, sgn, | |, а также $\log_{10}$, $\log_2$, ln, sin, обратные к ним и т. д. Можно также рассмотреть возможность его ограничения, например исключения операции ÷ и даже ×. Возможны и более гибкие модификации. Для некоторых операций можно допустить совершенно другие процедуры, например использование методов последовательных приближений или таблиц функций. <…> В любом случае, вероятно, должна существовать *центральная арифметическая* часть устройства, составляющая его *первую специальную часть*: *ЦА*.

10.2.3. Второе: логическое управление устройством, т. е. правильное упорядочение его операций, наиболее эффективно может быть выполнено центральным органом управления. Если устройство должно быть гибким, т. е. настолько *универсальным*, насколько возможно, то следует проводить различие между конкретными инструкциями, которые подаются ему и определяют конкретную задачу, и общими органами управления, отвечающими за то, чтобы эти инструкции – что бы они собой ни представляли – выполнялись. Первые должны как-то храниться – как это делается в существующих устройствах, описано в § 10.1.2, – последние представлены теми или иными рабочими частями устройства. Под *центральным управлением* мы понимаем только последнюю функцию; реализующие ее органы составляют *вторую специальную часть: ЦУ*.

10.2.4. Третье: любое устройство, предназначенное для выполнения длинных и сложных последовательностей операций (а именно вычислений), должно иметь объемную память. Память необходима по крайней мере на следующих четырех этапах его работы:

(a) даже в процессе выполнения умножения или деления необходимо запоминать ряд промежуточных (частичных) результатов. В меньшей степени это относится к сложению и вычитанию (когда цифру переноса следует переносить из одного разряда в другой), а в большей – к операциям $\sqrt{}$, $\sqrt[3]{}$, если они желательны <…>;

(b) инструкции, управляющие решением сложной задачи, могут составлять значительный материал, особенно если код детальный (как это чаще всего и бывает). Этот материал необходимо запоминать;

(c) во многих задачах существенную роль играют конкретные функции. Обычно они задаются в форме таблиц. Иногда именно в таком виде они и определяются в ходе эксперимента (например, уравнение состояния вещества во многих гидродинамических задачах), а иногда, несмотря на наличие аналитического выражения, проще и быстрее получать их значения из фиксированных таблиц, чем каждый раз вычислять за-

ново (пользуясь аналитическим определением). Обычно удобно иметь таблицы умеренного размера (100–200 элементов) и применять интерполяцию. Линейной и даже квадратичной интерполяции в большинстве случаев недостаточно, поэтому лучше полагаться на стандартную кубическую или биквадратную (а то и более высокого порядка) интерполяцию. <...> Некоторые функции, упомянутые в § 10.2.2, могут обрабатываться таким способом: $\log_{10}$, $\log_2$, ln, sin, обратные к ним и, возможно, также $\sqrt{}$, $\sqrt[3]{}$. Так можно даже вычислять обратное число, сведя тем самым ÷ к ×;

(d) для дифференциальных уравнений в частных производных начальные и граничные условия могут составлять объемный числовой материал, который необходимо помнить на всем протяжении решения;

(e) для дифференциальных уравнений в частных производных гиперболического или параболического типа, интегрируемых по переменной t, (промежуточные) результаты на цикле t необходимо запоминать для вычисления цикла $t + dt$. Этот материал во многом того же типа, что в п. (d), с тем отличием, что он не вводится в устройство оператором-человеком, а порождается (и, скорее всего, впоследствии удаляется и замещается соответствующими данными для $t + dt$) самим устройством в процессе автоматического функционирования;

(f) к уравнениям в полных дифференциалах пп. (d), (e) также применимы, но требуется память меньшего объема. Дополнительные требования к памяти типа (d) предъявляются в задачах, в которых используются заданные константы, фиксированные параметры и т. д.;

(g) задачи, решаемые методом последовательных приближений (например, дифференциальные уравнения в частных производных эллиптического типа, решаемые релаксационными методами), требуют памяти типа (e): промежуточные результаты после каждого приближения должны храниться, пока вычисляется следующее приближение;

(h) в задачах сортировки и некоторых статистических экспериментах (для которых высокоскоростное устройство предлагает интересную возможность) требуется память для обрабатываемого материала.

10.2.5. Подводя итоги третьему замечанию: устройство требует значительного объема памяти. Хотя могло сложиться впечатление, что различные части этой памяти должны выполнять функции, немного различающиеся по своей природе и значительно по назначению, тем менее очень хотелось бы рассматривать всю память как один орган, части которого настолько взаимозаменяемы, насколько возможно, в использовании для различных перечисленных выше функций. Этот момент будет рассмотрен более подробно. <...>

Так или иначе, вся *память* составляет *третью специальную часть устройства: П*.

10.2.6. Все три специальные части, ЦА, ЦУ (вместе Ц) и П, соответствуют ассоциативным нейронам в нервной системе человека. Остается обсудить эквиваленты *сенсорных*, или *рецепторных*, нейронов и *двигательных*, или *эффекторных*, нейронов. Это органы ввода и вывода устройства, и сейчас мы их кратко рассмотрим.

Иными словами: любая передача числовой (или иной) информации между частями Ц и П устройства должна осуществляться посредством механизмов, содержащихся в этих частях. Остается, однако, необходимость получать исходную определяющую информацию из внешнего мира, а также передавать конечную информацию, результаты, из устройства вовне.

Под внешним миром мы понимаем носитель описанного в § 10.1.2 типа: здесь информация может порождаться более-менее непосредственно в результате действий человека (печати, перфорации, фотографирования, световых импульсов, производимых клавишами одного и того же типа, намагничивания металлической ленты или проволоки и т. д.), она может статически храниться и, наконец, более или менее непосредственно восприниматься человеческими органами чувств.

Устройство должно быть наделено способностью поддерживать контакт (сенсорный и двигательный) с некоторым носителем этого вида для ввода и вывода (см. § 10.1.2): этот носитель будем называть *внешним носителем информации устройства*: Н. Итак, имеем:

10.2.7. Четвертое: устройство должно иметь органы для передачи (числовой и иной) информации из Н в свои специальные части, Ц и П. Эти органы образуют его *вводящую специальную часть*: *ВВ*. Ниже будет показано, что лучше всего организовать все передачи из Н (посредством ВВ) в П, а не прямо в Ц (см. § 10.14.1).

10.2.8. Пятое: устройство должно иметь органы для передачи (предположительно только числовой информации) из своих специальных частей Ц и П в Н. Эти органы образуют его *выводящую специальную часть*: *ВЫ*. Ниже будет показано, что и на этот раз лучше всего организовать все передачи в Н из П (посредством ВЫ), а не прямо из Ц (см. § 10.14.1).

10.2.9. Выводимая информация, поступающая в Н, естественно, представляет собой конечные результаты работы устройства над решаемой задачей. Их следует отличать от промежуточных результатов, обсуждаемых, например, в § 10.2.4, (e)–(g), которые остаются в П. Теперь возникает важный вопрос: помимо большей или меньшей прямой доступности человеческим действиям и восприятию, Н обладает также свойствами памяти. Действительно, это естественная среда для долговременного хранения всей информации, полученной автоматическим устройством при решении различных задач. Почему же тогда необходимо обеспечивать другой вид памяти внутри устройства, П? Нельзя ли все или, по крайней мере, некоторые функции П – предпочтительно те, которые связаны с большими объемами информации, – позаимствовать от Н?

Пристальное рассмотрение типичных функций П, перечисленных в § 10.2.4, (a)–(h), показывает следующее: было бы удобно вынести (a) (краткосрочную память, необходимую на время выполнения арифметической операции) за пределы устройства, т. е. из П в Н. (Фактически (a) будет внутри устройства, но в ЦА, а не в П. <…>) Все существующие устройства, даже настольные счетные машины в этой точке, используют эквивалент П. Однако (b) (логические инструкции) должны поступать извне, т. е. из Н посредством

ВВ, и то же самое относится к (c) (таблицы функций) и (e), (g) (промежуточные результаты). Последние могут также передаваться в Н посредством ВЫ, когда их порождает устройство, и считываться с Н посредством ВВ, когда в них возникает необходимость. Это до некоторой степени справедливо и в отношении (d) (начальные условия и параметры), и, быть может, даже (f) (промежуточные результаты решения уравнения в полных дифференциалах). Что касается (h) (сортировка и статистика), то ситуация не вполне однозначна: во многих случаях возможность использовать П значительно ускоряет работу, но подходящую комбинацию использования П совместно с долговременным хранением на Н, возможно, удастся реализовать без серьезной потери скорости, и это существенно увеличит объем доступного для обработки материала.

Действительно, все существующие (полностью или частично автоматические) вычислительные устройства используют Н – например, колоды перфокарт или отрезки телетайпной ленты – для всех этих целей (за исключением (a), как было отмечено выше). Тем не менее представляется, что полезность по-настоящему высокоскоростного устройства была бы крайне ограничена, если бы оно не могло опираться на П, а не на Н для всех целей, перечисленных в § 10.2.4, (a)–(h), с некоторыми ограничениями в случаях (e), (g), (h). <...>

10.3. Процедура обсуждения

10.3.1. Разобравшись с классификацией в § 10.2, мы теперь можем взять все пять специальных частей, составляющих устройство, и обсудить их по очереди. Такое обсуждение может выявить функциональность каждой части по отдельности, а также их взаимосвязи. Следует также определить конкретные процедуры, необходимые для работы с числами с точки зрения устройства, для выполнения арифметических операций и для обеспечения общего логического управления. В рамках этого рассмотрения должны быть разрешены все вопросы синхронизации и быстродействия, а также определена сравнительная важность различных факторов.

10.3.2. В идеале хотелось бы взять все пять специальных частей в каком-то определенном порядке, исчерпывающим образом рассмотреть каждую и переходить к следующей только после того, как разрешены все вопросы с предыдущей. Однако такой план вряд ли осуществим. Желательные функции частей и принимаемые на этой основе решения становятся ясны только после обсуждения, далекого от прямолинейного. Поэтому приходится брать одну часть, переходить после ее неполного обсуждения ко второй, возвращаться после столь же неполного обсуждения последней к первой части, уже имея объединенные результаты, продолжать обсуждение первой части, не надеясь на его окончательность, затем, возможно, переходить к третьей части и т. д. Более того, эти обсуждения специальных частей будут перемежаться обсуждениями общих принципов, арифметических процедур, используемых элементов и т. д.

В ходе такого обсуждения постепенно кристаллизуются желательные функции и наиболее подходящая им организация, и в конечном итоге устройство и управление им примут более-менее определенную форму. Как подчеркивалось выше, это относится как к физическому устройству, так и к компоновке арифметических и логических элементов, определяющей его функционирование.

10.3.3. В процессе обсуждения должны быть также рассмотрены поднятые в § 10.1.4 вопросы, касающиеся обнаружения, определения места возникновения и, при некоторых условиях, даже исправления ошибок. То есть следует уделить внимание средствам контроля ошибок. Мы не сможем в полной мере отдать должное этой чрезвычайно важной теме, но постараемся хотя бы бегло касаться ее, когда это будет казаться существенным. <...>

10.4. Элементы, синхронизация, аналогия с нейронами

10.4.1. Начнем с обсуждения некоторых общих положений.

Любое цифровое вычислительное устройство содержит какие-то *элементы* типа реле с дискретными положениями равновесия. Такой элемент имеет два или более состояний, в которых может находиться неограниченно долго. Это могут быть состояния идеального равновесия, в которых элемент пребывает без внешней поддержки, а подходящий внешний импульс перемещает его из одного состояния равновесия в другое. Или же может иметься два состояния, одно из которых существует, когда нет никакой внешней поддержки, а для существования другого необходим внешний стимул. Действие реле проявляется в испускании стимула со стороны элемента, когда он сам получил стимул описанного выше типа. Испускаемый стимул должен быть того же типа, что полученный, т. е. он должен быть способен стимулировать другие элементы. Однако не должно быть никакой энергетической связи между полученным и испущенным стимулами, т. е. элемент, получивший стимул, должен быть способен испустить несколько стимулов такой же интенсивности. Иными словами: будучи реле, элемент должен получать свой запас энергии от какого-то источника, не совпадающего с входящим стимулом.

В существующих цифровых вычислительных устройствах в качестве элементов используются различные механические или электрические устройства: колеса, которые могут фиксироваться в любой из десяти (или более) значимых позиций и при перемещении из одной позиции в другую передают электрические импульсы, способные приводить в движение другие подобные колеса; одиночные или комбинированные телеграфные реле, приводимые в действие электромагнитом и предназначенные для замыкания и размыкания электрических цепей; комбинации обоих элементов; и, наконец, существует вероятная и соблазнительная возможность использовать вакуумные лампы, сетка которых играет роль затвора в катодно-анодной цепи.

В последнем случае сетку можно также заменить отклоняющими органами, т. е. вакуумную лампу – катодно-лучевой лампой, но весьма вероятно, что в недалеком будущем большая доступность и многообразные электрические преимущества настоящих вакуумных ламп выдвинут первую процедуру на передний план.

Любое такое устройство может автономно хронометрировать себя посредством последовательных времен отклика своих элементов. В этом случае все стимулы должны в конечном итоге происходить от входного сигнала. Альтернативно отсчет времени может задаваться фиксированным задатчиком времени, который подает стимулы, необходимые для функционирования в определенные периодически повторяющиеся моменты времени. В роли такого задатчика может выступать вращающаяся ось в механическом или электромеханическом устройстве или же электрический осциллятор (возможно, управляемый кристаллом) в чисто электрическом устройстве. Если работа устройства должна зависеть от синхронизации нескольких различных одновременно выполняемых последовательностей операций, то задатчик времени, очевидно, предпочтительнее. Мы будем использовать термин *элемент* в определенном выше техническом смысле и называть устройство *синхронным* или *асинхронным* в зависимости от того, определяется его хронометраж задатчиком времени или автономно.

10.4.2. Стоит отметить, что нейроны высших животных, безусловно, являются элементами в определенном выше смысле. Все они имеют тип «все или ничего», т. е. могут находиться в одном из двух состояний: пассивный или возбужденный. Они отвечают требованиям в § 10.4.1 и при этом демонстрируют интересный вариант: возбужденный нейрон испускает стандартный стимул по нескольким линиям (аксонам). Однако такая линия может быть соединена со следующим нейроном двумя разными способами. Первый: *возбуждающим синапсом*, так что стимул вызывает возбуждение нейрона. Второй: *тормозящим синапсом*, так что стимул полностью предотвращает возбуждение нейрона по любому другому (возбуждающему) синапсу. У нейрона также имеется определенное время реакции между получением стимула и испусканием вызванного им стимула, *синаптическая задержка*.

Следуя работе Мак-Каллока и Питтса (1943, здесь глава 9), мы игнорируем более сложные аспекты функционирования нейрона: пороги, временную суммацию, относительное торможение, изменения порога в результате эффекта последействия стимулирования за пределами синаптической задержки и т. д. Удобно, однако, иногда рассматривать нейроны с фиксированными порогами 2 и 3, т. е. нейроны, которые могут возбуждаться (одновременными) стимулами на двух или трех возбуждающих синапсах (при отсутствии стимула на тормозящем синапсе). <…>

Легко видеть, что такие упрощенные функции нейронов можно имитировать с помощью телеграфного реле или вакуумных трубок. Хотя нервная система предположительно асинхронная (из-за синаптических задержек), точные синаптические задержки можно получить, используя синхронную конфигурацию. <…>

10.4.3. Ясно, что высокоскоростное вычислительное устройство в идеале

должно иметь элементы на вакуумных лампах. Агрегаты из вакуумных ламп, например счетчики и преобразователи, уже использовались и продемонстрировали надежную работу при времени реакции (синаптической задержке) порядка одной микросекунды (= 10^{-6} с), такая производительность даже отдаленно недоступна никакому другому устройству. Действительно, механические устройства можно отбросить сразу, а практическое время реакции телеграфного реле составляет порядка 10 миллисекунд (= 10^{-2} с) и более. Интересно отметить, что синаптическая задержка человеческого нейрона имеет порядок одной миллисекунды (= 10^{-3} с).

Поэтому далее мы будем предполагать, что элементами устройства являются вакуумные лампы. Мы также попытаемся оценить количество необходимых ламп, хронометраж и т. д., исходя из предположения, что используются стандартные, коммерчески доступные типы ламп. То есть не предполагается использование ламп особой сложности или с принципиально новыми функциями. Можно ли использовать новые типы ламп, станет яснее после тщательного анализа стандартных типов (или каких-то эквивалентных элементов <...>).

Наконец, будет показано, что у синхронного устройства имеются значительные преимущества. <...>

10.5. Принципы, определяющие арифметические операции

10.5.1. Рассмотрим теперь некоторые функции первой специальной части: центрального арифметического блока, ЦА.

Элемент в смысле § 10.4.3, вакуумная лампа, используемая в качестве затвора или *вентиля*, является устройством типа «все или ничего» или, по крайней мере, приближается к нему: в зависимости от того, выше порога напряжение на сетке или ниже, ток пропускается или нет. Верно, что для поддержания любого состояния необходимы определенные потенциалы на всех электродах, но существуют комбинации вакуумных ламп, достигающие идеального равновесия: несколько состояний, в каждом из которых комбинация может оставаться неопределенно долго без внешней поддержки, и при этом подходящие внешние стимулы (электрические импульсы) будут переводить комбинацию из одного равновесного состояния в другое. Существуют так называемые триггерные схемы, самая простая из которых имеет два равновесных состояния и содержит два триода или один пентод. Триггерные схемы с большим числом равновесных состояний непропорционально сложнее.

Таким образом, не важно, используются ли лампы в качестве вентилей или триггеров, конфигурации типа «все или ничего», с двумя равновесными состояниями, самые простые. Поскольку такие конфигурации ламп должны обрабатывать числа на основе их цифр, естественно использовать систему счисления, в которой цифры принимают всего два значения. Это наводит на

мысль о двоичной системе.

Аналоги человеческих нейронов, обсуждавшихся в § 10.4.2–10.4.3, точно так же являются элементами типа «все или ничего». Представляется, что они чрезвычайно полезны для всех предварительных, прикидочных рассмотрений систем на вакуумных лампах (см. § 10.6.1–10.6.2). Поэтому вызывает удовлетворение тот факт, что и здесь естественной системой счисления оказывается двоичная.

10.5.2. Последовательное использование двоичной системы, вероятно, также значительно упростит операции умножения и деления. Конкретно, оно позволит уйти от таблицы десятичного умножения или альтернативной ей процедуры удвоения, которая сначала строит кратные каждой цифре множителя или частного путем сложения, а затем объединяет их (в соответствии с позицией) с помощью второй последовательности сложений или вычитаний. Иными словами, двоичная арифметика имеет более простую и цельную логическую структуру, чем любая другая, особенно десятичная.

Конечно, не стоит забывать, что числовой материал, которым непосредственно пользуется человек, будет, вероятно, выражен в десятичной системе. Тем не менее в ЦА предпочтительнее использовать только двоичные процедуры, и в таком же виде выражать любой числовой материал, поступающий в центральный блок управления, ЦУ. Следовательно, в П должен храниться только двоичный материал.

Это делает необходимым включение в ВВ и ВЫ средств преобразования из десятичной системы в двоичную и обратно. Поскольку эти преобразования подразумевают большой объем арифметических операций, экономически наиболее целесообразно использовать ЦА, а значит, и ЦУ для координации, в связке с ВВ и ВЫ. Однако использование ЦА означает, что вся арифметика в обоих преобразованиях должна быть строго двоичной. <…>

10.5.3. Здесь возникает еще один принципиальный вопрос. Во всех существующих устройствах, где элементом является не вакуумная лампа, время реакции элемента настолько велико, что желательно как-то совместить шаги сложения, вычитания и, в еще большей степени, умножения и деления. Для примера рассмотрим двоичное умножение. Для многих задач, сводящихся к дифференциальным уравнениям, разумным считается оставлять 8 значащих десятичных цифр, так чтобы относительная ошибка округления не превышала 10^{-8}. Это соответствует 2^{-27} в двоичной системе, т. е. оставлению 27 значащих двоичных цифр. Поэтому умножение включает вычисление произведений всех пар, включающих одну из 27 двоичных цифр (0 или 1) множимого и одну из 27 цифр множителя, с последующим комбинированием и помещением их в нужные позиции. Получается $27^2 = 729$ шагов, а операции сбора и комбинирования могут приблизительно удвоить это число. Стало быть, нужно 1000–1500 шагов.

Естественно заметить, что в десятичной системе число шагов значительно меньше: $8^2 = 64$, а после удвоения примерно 100 шагов. Однако за это уменьшение приходится расплачиваться использованием таблицы умножения либо иным увеличением количества или сложности оборудования. Но

такой ценой можно сократить и процедуру двоичного умножения, применив уловки, о которых будет сказано ниже. По этой причине представляется необязательным отдельно обсуждать десятичную процедуру.

10.5.4. Как уже было сказано, при 1000–1500 последовательных шагах любое устройство, не основанное на вакуумных лампах, оказывается недопустимо медленным. У всех таких устройств, если не считать нескольких новейших реле специального назначения, время реакции превышает 10 миллисекунд, а эти новейшие реле (со временем реакции, сокращенным до 5 миллисекунд) эксплуатируются еще недостаточно долго. Получается минимум 10–15 секунд на одно умножение (8-значных десятичных чисел), тогда как у быстрых современных настольных арифмометров это время равно 10 секунд, а у стандартных элементов перемножения IBM – 6 секунд. (О важности этих временных показателей, а также об аналогичных показателях для потенциальных устройств на вакуумных лампах применительно к типичным задачам см. <...>.)

Логическая процедура, позволяющая избежать таких долгих задержек, заключается в *совмещении операций*, т. е. одновременном выполнении максимально возможного их числа. Сложности переноса не позволяют одновременно выполнить даже такие простые операции, как сложение и умножение. В случае деления вычисление следующей цифры даже нельзя начать, пока не станут известны все цифры слева от нее. И все же значительное совмещение возможно: при сложении или вычитании все пары соответственных цифр можно объединить одновременно, все первые разряды переноса можно применить вместе на следующем шаге и т. д. В случае умножения все частичные произведения вида (множимое) × (цифра множителя) можно вычислить и разместить одновременно – в двоичной системе такое частичное произведение равно нулю или множимому, поэтому остается только размещение. И для сложения, и для умножения можно использовать вышеупомянутые формы сложения и вычитания. Кроме того, при умножении частичные суммы можно быстро сложить, если складывать первую пару одновременно со второй, третью с четвертой и т. д., а затем сложить первую пару сумм одновременно со второй парой и т. д. И так, пока не будут собраны все члены. (Поскольку $27 < 2^5$, это позволяет собрать 27 частичных сумм – в предположении, что множитель состоит из 27 двоичных цифр, – за время, необходимое для пяти сложений. Эта схема предложена Г. Эйкеном.)

Такое ускоряющее совмещение применяется во всех существующих устройствах. (Использование десятичной системы, с совмещением или без, также относится к этому типу, как отмечено в конце § 10.5.3. Но фактически оно несколько менее эффективно, чем чисто диадические процедуры. Аргументы, приведенные в § 10.5.1–10.5.2, делают излишним рассмотрение здесь этого вопроса.) Однако такие процедуры экономят время ровно в той степени, в какой усложняют оборудование, т. е. количество элементов в устройстве. Очевидно, что если длительность уменьшается вполовину путем систематического выполнения сразу двух сложений, то требуется удвоить количество сумматоров (даже если считать, что их можно использовать без непропорционального усложнения средств управления и на сто процен-

тов эффективно) и т. д.

Описанный способ экономии времени за счет дополнительного оборудования вполне оправдан в устройствах, не основанных на вакуумных лампах, где выигрыш во времени – насущная необходимость, и имеется обширный инженерный опыт по части организации сложных устройств, содержащих много элементов. По-настоящему универсальная автоматическая цифровая вычислительная система, сконструированная по этому принципу с учетом всего имеющегося опыта, содержит свыше 10 000 элементов.

10.5.5. С другой стороны, для устройства на вакуумных лампах более многообещающей представляется противоположная процедура.

Как отмечалось в § 10.4.3, время реакции не слишком сложного устройства на вакуумных лампах может быть доведено до одной микросекунды. При такой скорости даже безо всяких уловок время умножения, выведенное в § 10.5.3, приемлемо: для 1000–1500 реакций нужно 1–1,5 миллисекунды, а это настолько быстрее любого мыслимого устройства на иной элементной базе, что на самом деле возникает серьезная проблема поддержания загруженности устройства – необходимо, чтобы наблюдение за вводом и выводом со стороны человека не отставало от темпа работы. <…>

Относительно других арифметических операций можно сказать следующее: сложение и вычитание, очевидно, гораздо быстрее умножения. При наличии 27 двоичных цифр (см. § 10.5.3) и принимая во внимание переносы, обе операции требуют не более чем два раза по 27 шагов, т. е. примерно 30–50 шагов, или времен реакции. Это 0,03–0,05 миллисекунды. При той же схеме, когда не предпринимается никаких уловок для уменьшения количества и совмещения операций и используется двоичная система, деление требует примерно такого же количества шагов, как умножение. <…> Извлечение квадратного корня по такой же схеме обычно не намного медленнее деления.

10.5.6. Поэтому ускорение этих арифметических операций не кажется необходимым – по крайней мере, до тех пор, пока мы не накопим обширный практический опыт использования высокоскоростных устройств такого рода и, достигнув надлежащего понимания, не начнем использовать совершенно новые возможности решения сложных задач, которые они открывают. Кроме того, кажется сомнительным, достигнет ли вообще в этой ситуации своей цели метод ускорения путем совмещения процессов ценой кратного увеличения количества элементов: чем сложнее оборудование на вакуумных лампах, т. е. чем больше необходимое количество элементов, тем больше должны быть допуски. Следовательно, любое увеличение в этом направлении потребует также иметь дело с большим временем реакции, чем вышеупомянутая одна микросекунда. Точные последствия этого фактора трудно оценить количественно в общем случае – но, безусловно, они гораздо важнее для элементов на вакуумных лампах, чем на телеграфных реле.

Таким образом, представляется разумным рассмотреть следующую точку зрения: устройство должно быть максимально простым, т. е. содержать как можно меньше элементов. Этого можно добиться, никогда не выполняя две

операции одновременно, если это ведет к значительному увеличению количества необходимых элементов. В результате устройство будет работать более надежно, а время реакции вакуумных ламп можно будет сделать меньше, чем в противном случае.

10.5.7. До какого предела можно распространить этот принцип, не жертвуя пользой, разумеется, зависит от физических характеристик имеющихся элементов на вакуумных лампах. Возможно, оптимум достигается не при 100%-ном его применении и будет найден какой-то оптимальный компромисс. Однако это всегда будет зависеть от текущего состояния технологии вакуумных ламп; очевидно, что чем быстрее лампы, надежно работающие в этой ситуации, тем больше причин для бескомпромиссного применения этого принципа. Создается впечатление, что даже при имеющихся на данный момент технических возможностях оптимум находится довольно близко к бескомпромиссному решению.

Следует также подчеркнуть, что до настоящего времени все размышления о высокоскоростных цифровых вычислительных устройствах были устремлены в противоположном направлении: ускорения путем совмещения процессов ценой кратного увеличения количества потребных элементов. Поэтому представляется тем более поучительным как можно тщательнее продумать противную точку зрения: полностью воздержаться от вышеупомянутой процедуры, т. е. последовательно придерживаться принципа, сформулированного в § 10.5.6.

Поэтому мы продолжим движение в этом направлении. <...>

10.6. E-элементы

10.6.1. Соображения, изложенные в § 10.5, легли в основу главных принципов работы с ЦА.

Теперь, опираясь на этот фундамент, мы перейдем к более конкретным техническим деталям. Для этого нам потребуется использовать схематическое изображение функционирования стандартного элемента устройства; действительно, решения, относящиеся к арифметическим процедурам и логическому управлению устройством, равно как и к другим его функциям, можно принять только на основе каких-то допущений о функционировании элементов.

В идеале хотелось бы рассматривать элементы как то, чем мы собираемся их сделать: вакуумные лампы. Однако это вынудит нас опуститься на уровень детального анализа конкретных вопросов радиотехники уже на этой ранней стадии обсуждения, когда имеется еще слишком много альтернатив, которые следует изучить подробно и исчерпывающим образом. Кроме того, многочисленные альтернативные возможности организации арифметических процедур, логического управления и т. д. наложатся на столь же многочисленные возможности выбора типоразмеров вакуумных ламп и других элементов с точки зрения практической производительности и т. д. Все это создаст запутанную и непрозрачную ситуацию, в которой предварительная

ориентировка, которая нас сейчас интересует, будет вряд ли возможна.

Чтобы избежать этого, мы положим в основу рассмотрения гипотетический элемент, который по существу работает как вакуумная лампа – например, как триод с подходящей резистивно-индуктивно-емкостной цепью, – но который можно обсуждать как изолированный объект, не вдаваясь в детали радиочастотного электромагнитного поля. Подчеркнем еще раз: это упрощение лишь временное – промежуточная точка, необходимая, чтобы сделать возможным предварительное обсуждение. Придя к определенным предварительным выводам, мы должны будем вернуться к рассмотрению истинной электромагнитной природы элементов. Но к этому моменту уже будут ясны итоги предварительного обсуждения и исключены некоторые альтернативы.

10.6.2. Аналоги человеческих нейронов, которые обсуждались в § 10.4.2–10.4.3 и были еще раз упомянуты в конце § 10.5.1, похоже, дают нам элемент как раз того вида, о котором мы говорили в конце § 10.6.1. Их мы и предлагаем использовать для описанной выше цели: как составные элементы устройства на время предварительного обсуждения. Поэтому необходимо точно описать постулируемые свойства этих элементов.

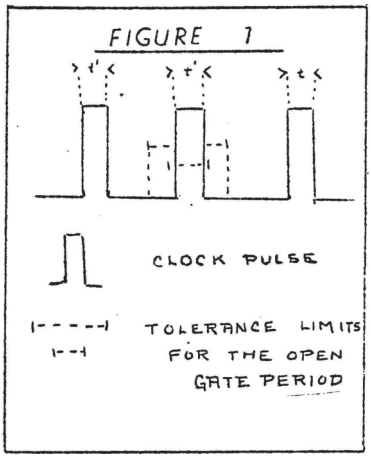


Рис. 10.2. Тактовые импульсы

Элемент, который мы будем обсуждать и назовем *E-элементом*, будет представлен кружочком O; он получает возбуждающие и тормозящие стимулы и испускает собственные стимулы по присоединенной к нему линии: O—. Эта линия может ветвиться: O—<, O—<. Испускаемый по ней стимул следует за исходным стимулом с синаптической задержкой, которую мы можем считать фиксированной, одинаковой для всех E-элементов, обозначим ее *t*. Мы предлагаем пренебречь всеми задержками (обусловленными распространением стимула вдоль линий), кроме *t*. Будем отмечать наличие задержки *t* стрелкой на линии: O—>, O—>. Это также позволит идентифицировать начало и направление линии.

10.6.3. Сейчас необходимо сделать следующее наблюдение. В нервной системе человека время прохождения сигнала по линиям (аксонам) может превышать синаптические задержки, поэтому наше предложение игнорировать все задержки, кроме *t*, было бы неоправданным. Однако для подразумеваемой интерпретации вакуумных ламп это предложение обосновано: *t* будет приближенно равно одной микросекунде, за это время электромагнитный импульс проходит расстояние 300 м, а поскольку линии, вероятно, будут гораздо короче, то временем прохождения по ним действительно можно пренебречь. (Потребовалось бы ультравысокочастотное устройство – $t = 10^{-8}$ или меньше – чтобы этот аргумент утратил силу.)

Еще одна точка существенного расхождения между нервной системой че-

ловека и нашим предполагаемым применением – наше использование точно определенной, не имеющей разброса синаптической задержки t, общей для всех Е-элементов. (Ключевой момент здесь – отсутствие разброса. Мы на самом деле будем использовать Е-элементы с синаптической задержкой $2t$, ….) Мы предлагаем использовать задержки t как абсолютные единицы времени, на которые можно полагаться для синхронизации функций различных частей устройства. Преимущества такой организации сразу же внушают доверие, конкретные технические причины будут приведены в [Примечание редактора: здесь предложение обрывается].

Чтобы добиться этого, необходимо рассматривать устройства как синхронное в смысле § 10.4.1. Центральный задатчик времени лучше всего представлять себе как электрический осциллятор, испускающий с периодом t короткий стандартный импульс продолжительностью t', равной приблизительно $\frac{1}{5}t - \frac{1}{2}t$. Стимулы, номинально испускаемые Е-элементом, на самом деле являются импульсами задатчика, для которого этот импульс действует как вентиль. Очевидно, имеется широкий допуск для периода, в течение которого вентиль должен быть открыт, чтобы тактовый импульс проходил без искажений. См. рис. 10.2. Таким образом, открытием вентиля можно управлять с помощью любого электрического устройства задержки со средним временем задержки t и достаточно большой допустимой дисперсией. Тем не менее эффективная синаптическая задержка будет считаться равной t в предположении идеальной точности задатчика времени, а стимул формируется полностью заново и синхронизируется после каждого шага. <…>

10.12. Емкость памяти П, общие принципы

10.12.1. Далее рассмотрим третью специальную часть: память П. <…>

10.12.2. Однако до этого обсуждения мы должны поговорить о том, какую емкость П хотим иметь. Это количество стимулов, которые может запомнить орган, или, точнее, количество случаев, для которых он может запомнить, был стимул или не было. Наличие либо отсутствие стимула (в заданном случае, т. е. на заданной линии в заданный момент) можно использовать, чтобы выразить значение 1 или 0 двоичной цифры (в заданной позиции). Поэтому емкость памяти – это количество двоичных цифр (их значений), которое можно в ней сохранить. Иными словами:

Единица (емкости) памяти – это способность сохранять значение одной двоичной цифры.

Теперь мы можем выразить «стоимость» хранения различных типов информации в этих единицах памяти.

Сначала рассмотрим емкость памяти, необходимую для хранения стандартного (вещественного) числа. Как было отмечено <…>, мы зафиксируем размер такого числа на уровне 30 двоичных цифр (по крайней мере, для большинства применений <…>). При этом относительные ошибки округления будут меньше 2^{-30}, т. е. 10^{-9}, что соответствует оставлению 9 значащих

десятичных цифр. Таким образом, стандартное число занимает 30 единиц памяти. К этому нужно добавить одну единицу на знак числа <...> и рекомендуется добавить еще одну единицу, которая будет характеризовать его как число (в отличие от команды, см. § 10.14). Так мы приходим к значению $32 = 2^6$ единиц памяти на число.

Тот факт, что число требует 32 единиц памяти, наводит на мысль разделить всю память следующим образом. Прежде всего, очевидно, на единицы, затем на группы по 32 единицы, которые будем называть *малыми циклами*. <...> Соответственно, каждое стандартное (вещественное) число занимает один малый цикл. Организация всей памяти упростится, а вместе с тем станет проще решать сопутствующие разнообразные проблемы синхронизации устройства, если все прочие константы памяти также будут приспособлены к этому делению на малые циклы. <...>

10.14. ЦУ и П

10.14.1. Наша следующая цель – углубиться в анализ ЦУ. Такой анализ, однако, зависит от знания системы команд, используемых для управления устройством, т. к. функция ЦУ заключается в том, чтобы получать эти команды, интерпретировать их, а затем либо выполнять, либо надлежащим образом стимулировать органы, которые будут их выполнять. Поэтому прежде всего перед нами стоит задача предоставить список команд, управляющих устройством, т. е. описать код, который будет использоваться устройством, и определить математический и логический смысл и операционное значение слов этого кода.

Прежде чем мы сможем сформулировать этот код, следует рассмотреть несколько общих вопросов, касающихся функций ЦУ и его связи с П.

Команды, которые получает ЦУ, поступают из П, т. е. из того же места, где хранится числовой материал. (См. § 10.2.4. <...>) Содержимое П состоит из малых циклов <...>, поэтому, в силу сказанного выше, каждый малый цикл должен включать отметку, показывающую, содержит он стандартное число или команду.

Команды, получаемые ЦУ, можно естественно отнести к следующим четырем классам: (a) команды, заставляющие ЦУ инструктировать ЦА о необходимости выполнить одну из десяти его конкретных операций <...>; (b) команды, заставляющие ЦУ инициировать перемещение стандартного числа из одного места в другое; (c) команды, заставляющие ЦУ переместить свою собственную точку связи с П в другую точку П с целью получить оттуда следующую команду; (d) команды, управляющие вводом и выводом устройства (т. е. ВВ из § 10.2.7 и ВЫ из § 10.2.8).

Рассмотрим классы (a)–(d) по отдельности. В данный момент мы не можем ничего добавить к утверждениям <...>, касающимся (a) <...>. Обсуждение (d) также пока лучше отложить. <...> А вот (b) и (c) мы предлагаем обсудить сейчас.

10.14.2. К (b). Эти перемещения могут происходить внутри П, внутри ЦА

или между П и ЦА. Операцию первого рода всегда можно заменить двумя операциями третьего рода, т. е. маршрутизировать все перемещения внутри П через ЦА. Мы предлагаем так и поступать, потому что это согласуется с общим принципом из § 10.5.6 <...>, и поэтому исключим из рассмотрения все перемещения первого рода. Перемещения второго рода, очевидно, обрабатываются средствами управления работой самого ЦА. Остаются только перемещения третьего рода. Они, очевидно, распадаются на два класса: перемещения из П в ЦА и из ЦА в П. Поэтому мы можем разбить (b) на два подкласса (b′) и (b″), соответствующих этим двум операциям.

10.14.3. К (c). В принципе, ЦУ следует инструктировать после каждой команды, где искать следующую команду, подлежащую выполнению. Мы видели, однако, что в таком виде это нежелательно и должно быть зарезервировано для исключительных случаев, тогда как в ходе нормальной работы ЦУ должно исполнять команды в том порядке, в котором они естественно появляются на выходе органа линии задержки (DLA), к которому ЦУ подключено (см. соответствующее обсуждение памяти иконоскопа <...>). Должны, однако, существовать команды, которые можно использовать в исключительных случаях, когда требуется сказать ЦУ, чтобы оно переместило точку связи в другое место П. В основном речь идет о перемещении этой связи на другой орган DLA. Однако так как нужная связь должна быть задана определенным малым циклом, рассматриваемая команда должна состоять из двух инструкций: во-первых, переместить точку связи ЦУ на определенный орган DLA, а во-вторых, ждать, пока на выходе этой DLA появится определенный r-период – тот, в котором находится нужный малый цикл, – и только в этот момент ЦУ может принять команду.

Дополнительно команда перемещения должна обеспечить, чтобы после приема и выполнения команды в указанном малом цикле ЦУ вернуло точку связи на тот орган DLA, который содержит малый цикл, следующий за тем, который содержит команду передачи, дождалось появления этого малого цикла на выходе, а затем продолжило принимать команды с этой точки в их естественном порядке следования. Альтернативно, после приема и выполнения команды в указанном малом цикле ЦУ могло бы продолжить работу с этой точки и принимать команды оттуда в их естественном порядке. Удобно называть перемещение первого типа *временным*, а перемещение второго типа – *постоянным*.

Ясно, что постоянные перемещения нужны часто, поэтому второй тип, безусловно, необходим. Временные перемещения, несомненно, требуются в связи с перемещением стандартных чисел (команды (*c′*) и (*c″*), см. конец § 10.14.2 <...>). Кажется весьма сомнительным, так ли они необходимы в настоящих командах, в особенности потому, что такие команды составляют лишь малую часть содержимого П <...>, и потому, что команды временного перемещения всегда можно выразить двумя командами постоянного перемещения. Поэтому мы сделаем все перемещения постоянными, за исключением тех, что связаны с перемещением стандартных чисел, как отмечено выше. <...>

10.15. Код

10.15.1. Соображения, изложенные в § 10.14, закладывают основу для полной классификации содержимого П, т. е. систему последовательных дизъюнкций, которые вместе составляют эту классификацию. Наличие такой классификации позволит нам сформулировать код, который осуществляет логическое управление ЦУ, а значит, и всем устройством.

Поэтому повторим относящиеся к делу определения и дизъюнкции.

Содержимым П являются единицы памяти, каждая из которых характеризуется наличием или отсутствием стимула. Ее можно использовать для представления двоичной цифры 1 или 0, и ничто не мешает нам обозначать содержимое той двоичной цифрой $i = 1$ или 0, с которой установлено описанное соответствие. <...> Эти единицы группируются в малые циклы, состоящие из 32 единиц, и эти малые циклы и являются сущностями, которые приобретают непосредственную значимость в том коде, что мы собираемся представить. <...> Обозначим двоичные цифры, образующие 32 единицы малого цикла и следующие друг за другом в своем естественном порядке, i_0, i_1, i_2, ..., i_{31}. Малые циклы с такими единицами можно записать в виде $I = (i_0, i_1, i_2, \ldots, i_{31}) = (i_\nu)$.

Малые циклы распадаются на два класса: стандартные числа и команды. (См. <...> § 10.14.1.) Эти две категории различаются по первой единице <...>, т. е. по значению i_0. Условимся, что $i_0 = 0$ означает стандартное число, а $i_0 = 1$ – команду.

10.15.2. Оставшаяся 31 единица стандартного числа выражает его двоичные цифры и знак. Согласно природе всех арифметических операций и прежде всего из-за переноса разрядов двоичные цифры числа должны подаваться справа налево, т. е. цифры в позициях с меньшими номерами – сначала. (Это связано с тем, что цифры появляются поочередно, а не одновременно. <...> Детали наиболее выпукло проявляются при обсуждении сумматора. <...>) Знак выступает в роли самой левой цифры, т. е. цифры в позиции с наибольшим номером. <...> Поэтому он подается последним, т. е. $i_{31} = 0$ означает знак +, а $i_{31} = 1$ – знак –. Наконец, <...> двоичная точка расположена сразу после знакового разряда, а представленное таким образом число ξ должно быть сдвинуто по модулю 2 в интервал –1, 1. То есть $\xi = i_{31}i_{30}i_{29} \ldots i_1 = \sum_{\nu=1}^{31} i_\nu 2^{\nu-31} \pmod 2$, $-1 \leqslant \xi < 1$.

10.15.3. С другой стороны, оставшаяся 31 единица команды должна выражать природу этой команды. В § 10.14.1 было описано разделение команды на четыре класса (a)–(d), которые, в свою очередь, делятся на более мелкие. <...> [Примечание редактора: на рис. 10.3 приведены команды в табличном виде.]

Table.

(I) Type.	(II) Meaning.	(III) Short Symbol	(IV) Code Symbol
			Minor cycle $I = (i_\nu) = (i_0 i_1 i_2 \dots i_{31})$
Standard number or Order (γ)	Storage for the number defined by $\xi = i_{31}$. $i_{30}\ i_{29} \dots i_1 = \sum_{\nu}^{31} i_\nu 2^{31-\nu} \pmod 2_1 - 1 \leqq \xi < 1$. i_{31} is the sign: 0 for +, 1 for −. If CC is connected to this minor cycle, then it operates as an order, causing the transfer of into I_{ca}. This does not apply however if this minor cycle follows immediately upon an order w → A or wh → A.	$N\xi$	$i_0 = 0$
Order (α) +(∂)	Order to carry out the operation w in CA and to dispose of the result. w is from the list of 11.4. These are the operations of 11.4, with their current numbers w and their symbols w:	w →up or wh →up	$i_1 = 1$
Order (α) +(ε)		w →f or wh →f	
Order (α) +(ϑ)		w →A or wh →A	
Order (α)	h means that the result is to be held in 0_{ca}. →μp means, that the result is to be transferred into the minor cycle ρ in the major cycle u; →f, that it is to be transferred into the minor cycle immediately following upon the order; →A, that it is to be transferred into I_{ca}; no →, that no disposal is wanted (apart from h).	wh	
Order (β)	Order to transfer the number in the minor cycle ρ in the major cycle u into I_{ca}.	A←μp	
Order	Order to connect CC with the minor cycle ρ in the major cycle μ.	C←μp	

w, decimal	w, binary	w	w, decimal	w, binary	w
0	0000	+	5	0101	i
1	0001	−	6	0110	j
2	0010	x	7	0111	s
3	0011	/	8	1000	db
4	0100	√	9	1001	bd

Рис. 10.3. «Команды» (инструкции) в первой редакции EDVAC

11 Как мы можем мыслить (1945)

Ванневар Буш

Изучая технические дисциплины в МТИ, Ванневар Буш (1890–1974) построил аналоговый компьютер для решения дифференциальных уравнений, который назвал дифференциальным анализатором. По современным стандартам это было громоздкое устройство, преимущественно механическое, с несколькими электрическими компонентами, но оно работало. В 1936 году Буш нанял Клода Шеннона в качестве лаборанта для работы на этом устройстве. Этот опыт привел Шеннона к систематическим размышлениям о коммутационных схемах и в конечном итоге к его знаменитой магистерской диссертации (глава 8). Буш же начал раздумывать о хранении и поиске информации. Под влиянием недавно получивших распространение микрофильмов («Нью-Йорк таймс» начала публиковать свои номера в виде микроформ в 1935 году) Буш вообразил устройство, которое описывает в этой провидческой статье 1945 года, под названием *memex*. Микрофильм точно так же воодушевил писателя-фантаста Г. Г. Уэллса (Wells 1938). «Теперь нет никаких практических препятствий, – писал он, – к созданию эффективного указателя ко всем знаниям, идеям и достижениям человеком, т. е. к созданию полной планетарной памяти всего человечества. <…> Вся память человечества может и, вероятно, в скором времени будет сделана доступной любому индивидууму».

Буш, напротив, писал как инженер, и не просто рядовой инженер. Он занимал должность директора управления научных исследований и разработок во время Второй мировой войны, направляя усилия американского научного сообщества на оказание помощи в военных действиях. Он, например, хорошо знал о компьютере Mark I Эйкена, созданном в Гарварде, и о трудных вычислениях, проделанных в Манхэттенском проекте при проектировании атомного оружия. Эта статья появилась примерно в то же время, когда Буш вручил президенту Трумэну доклад «Наука: бесконечные рубежи» (Bush 1945c),

посвященный будущему науки, который привел к созданию Национального научного фонда. Буш опубликовал этот труд в журнале «Atlantic», – а вскоре вслед за тем и в журнале «Life» в сокращенной и богато иллюстрированной форме (Bush 1945b) – как часть своих усилий по расширению общественной поддержки научных исследований.

Здесь Буш задается вопросом, какой вклад наука может внести в мирное будущее человечества, и предлагает в ответ свое видение мгновенного, всепроникающего, ассоциативного поиска информации. Имевшиеся в то время носители не могли обеспечить такое достижение – статья интересна просто как обзор тогдашних технологий вычислений и хранения и проекция их совершенствования на будущее, – но его провидение функциональности стало источником вдохновения для многих позднейших электронных систем.

Это не была война ученых, это была война, в которой каждый принял участие. Ученые, похоронившие свои старые профессиональные разногласия, в угоду общему делу многим поделились и многому научились. Работать эффективно и быть при этом партнерами оказалось на удивление весело. Сейчас для многих это подходит к концу. Что ученые будут делать дальше? Для биологов и частично для ученых-медиков для нерешительности почти нет оснований, потому что война едва ли потребовала от них сойти со старой тропы. На самом деле многие смогут продолжать начатые во время войны исследования в своих хорошо знакомых лабораториях мирного времени. Их цели почти не изменились.

А вот физики были наиболее жестоко сбиты со своего пути, оставили академические карьеры во имя создания странных разрушительных приспособлений, принуждены были разрабатывать новые методы для непредвиденных задач. Они сделали свою часть работы, изготовив устройства, которые сделали возможным поражение врага. Они работали, объединив усилия с физиками наших союзников. Они ощутили жажду свершений. Они были частью великой команды. Сейчас, когда мир на подходе, мы задаем себе вопрос, где они найдут задачи себе по плечу.

11.1

Какую не сиюминутную пользу принесли человеку достижения науки и новые инструменты, созданные в результате его исследований? Во-первых, эти инструменты усилили контроль человека над материальным миром. Улучшилась одежда человека, его еда, его убежища; эти инструменты обезопасили человека, частично освободили его от оков борьбы за выживание. Они расширили его знания о своих собственных биологических процессах, так что он постепенно освобождается от болезней, продолжительность его жизни увеличивается. Они делают понятнее взаимоотношения между психологической и физиологической функциями, обещая тем самым улучшение психического здоровья. Наука ускорила коммуникацию между людьми; дала

возможность записывать идеи, возможность манипулировать этой информацией и извлекать из записей нужное; таким образом, знание развивается и сохраняется в течение жизни всей расы, а не индивидуума. Гора исследований растет. Но все больше свидетельств в пользу того, что мы погружаемся в болото вследствие растущей специализации. Исследователь застывает в нерешительности среди сотен чужих находок и выводов – выводов, которые у него не хватает времени осознать, не говоря уже о том, чтобы запомнить. Однако специализация становится все более необходимой для прогресса, а усилия, направленные на наведение мостов между дисциплинами, становятся все менее основательными.

Специалисту известно, что наши методы передачи и анализа результатов исследований возникли несколько поколений назад и ныне совершенно не отвечают своему назначению. Если совокупное время, потраченное на написание научных трудов и на их прочтение, можно было бы вычислить, то разница между этими величинами времени поразила бы воображение. Те, кто честно будет пытаться идти в ногу с течением мысли, даже в ограниченной области, сосредоточенно и непрерывно читая, скорее всего, в ужасе отшатнутся, узнав, сколько усилий, затраченных в прошлом месяце, можно было бы сэкономить, просто сделав один звонок. Законы генетики Менделя целое поколение были неизвестны миру, потому что его публикации не дошли до тех немногих, кто мог бы понять и развить их; и такого рода катастрофы, несомненно, происходят вокруг нас вновь и вновь – что-то по-настоящему ценное теряется в потоке несущественного.

Проблема видится не столько в том, что мы публикуем чрезмерно много, – ведь широта и многообразие интересов в сегодняшнем мире действительно велики, – а в том, что количество публикаций вышло далеко за пределы нашей способности использовать их во благо. Совокупность человеческого опыта растет с неимоверной скоростью, а средства, которыми мы пользуемся для отыскания в этом лабиринте того, что важно в данный момент, те же, что во времена парусников. Но есть и признаки перемен, поскольку на свет появляются новые мощные инструменты. Фотоэлементы, способные «видеть» в физическом смысле; продвинутая техника фотографии, с помощью которой можно сфотографировать видимое и невидимое; электронные лампы с накаливаемым катодом, способные контролировать могущественные силы, потребляя энергии меньше, чем требуется комару на трепетание крылышками; электронно-лучевые трубки, делающие видимыми события настолько быстротечные, что в сравнении с ними микросекунда – это долго; релейные схемы, которые могут управлять сложными последовательностями действий надежнее любого человека и в тысячу раз быстрее, – есть множество технических решений, способных совершить переворот в регистрации научных результатов.

Два столетия назад Лейбниц придумал счетную машину, обладавшую большинством существенных функций недавно созданных устройств с клавиатурой, но тогда ее не удалось ввести в обиход. Против нее играла экономическая ситуация: трудовые затраты на ее изготовление во времена, когда еще не было массового производства, превышают трудовые затраты, которые она позволила бы сэкономить, поскольку все, чего удалось бы достичь с ее

помощью, можно было бы повторить, пользуясь лишь ручкой и бумагой, если затратить достаточно времени. Более того, она бы часто ломалась, поэтому мы не могли бы на нее положиться, поскольку и тогда, и еще много времени спустя сложность и ненадежность были синонимами.

Бэббидж, даже с невероятно щедрой для своего времени поддержкой, не смог построить свою большую арифметическую машину. Его идея звучала убедительно, но затраты на создание и обслуживание в те времена были неподъемными. Если бы у фараона были все необходимые для постройки автомобиля чертежи и даже если бы он смог разобраться в них, потребовались бы ресурсы всего царства египетского, чтобы изготовить тысячи деталей одной-единственной машины, и эта машина наверняка сломалась бы в первой же поездке в Гизу.

Машины со взаимозаменяемыми частями сейчас строятся гораздо экономнее. Несмотря на сложность, они надежно работают. Доказательства тому: печатная машинка, кинокамера, автомобиль. Электрические контакты перестали вставлять нам палки в колеса, как только стали понятными. Напомню про автоматизированные телефонные коммутаторы, в которых таких контактов сотни тысяч, и тем не менее они надежно работают. Тонюсенькая металлическая проволока, заключенная в тонкий стеклянный контейнер и нагретая до ослепительного сияния, короче говоря, электронная лампа радиоприемника – такие лампы изготавливают сотнями миллионов, упаковывают в коробки, которые грузчики швыряют туда-сюда, а потом вставляют в гнезда – и они работают! Тончайшие элементы, при изготовлении которых требовалась невероятная точность и над которыми лучшие мастера гильдии трудились бы месяцами, сейчас продаются за тридцать центов. Мир вступил в эпоху дешевых и сложных, но при этом очень надежных устройств; и что-то непременно должно за этим последовать.

11.2

Регистрация научных результатов, если мы хотим добиться от нее пользы, должна непрерывно продолжаться, их запись должна быть сохранена, и, главное, должна быть возможность к ней обращаться. На данный момент мы традиционно ведем записи с помощью письма и фотографии, с последующей печатью; кроме того, мы записываем на кинопленку, на восковые диски и на магнитную проволоку. Даже если никаких новых способов не появится, существующие, несомненно, будут модифицироваться и развиваться.

Действительно, прогресс в фотографии не собирается останавливаться.

В скором времени появятся более быстрые материалы и линзы, более автоматизированные камеры, мелкозернистые светочувствительные составы, позволяющие развить идею мини-камеры. Попробуем предсказать логический, а пожалуй, и неизбежный итог этой тенденции. Фотолюбитель будущего носит на лбу бугорок размером чуть больше грецкого ореха. Это устройство делает снимки размером 3×3 миллиметра, которые потом можно спроецировать или увеличить, что лишь в 10 раз превосходит наши нынеш-

ние возможности – не так уж много. Объектив имеет фиксированную наводку на фокус, во всем диапазоне аккомодации невооруженного глаза, просто потому что он короткофокусный. В орехе имеется встроенный фотоэлемент, такой же присутствует уже сейчас как минимум в одной автоматической камере и позволяет автоматически подстраивать экспозицию в широком диапазоне освещенности. Пленки в орехе хватит на сотни кадров, а пружина, что двигает затвор и прокручивает пленку, заводится всего один раз, когда мы вставляем кассету. Это устройство выдает полноцветные фотографии. Оно вполне может быть стереоскопическим и записывать с помощью разнесенных в пространстве стеклянных глаз, поскольку поразительные успехи в стереоскопической технике уже на подходе.

Шнур, который спускает затвор, может крепиться к рукаву, так чтобы человеку было легко им воспользоваться. Короткое нажатие – и снимок готов. На обычные очки нанесен квадрат из тонких линий почти у верхнего края линз, за пределами привычного поля зрения, где мы не замечаем этих линий. Когда объект оказывается в этом квадрате, линии обозначают границы будущего изображения. Когда ученый будущего, в лаборатории или в экспедиции, увидит что-то стоящее, ему достаточно нажать на спуск затвора – и все будет сделано бесшумно. Фантастика? Фантастика только в том, как проявить так много фотографий.

Появится ли сухая фотография? Она уже существует в двух формах. Когда Брэди делал свои фотографии Гражданской войны, фотопластинка должна была быть влажной в момент экспозиции. Сейчас пленка должна быть влажной только во время проявления. В будущем, возможно, ее вообще не понадобится увлажнять. Уже довольно давно существует пленка, пропитанная диазокрасителем, при использовании которой изображение появляется без проявления, т. е. сразу после использования камеры. Обработка парами аммиака разрушает неэкспонированный краситель, после чего фотографию можно извлечь на свет и посмотреть. Пока что этот процесс медленный, но кто-нибудь придумает, как его ускорить, и он лишен проблем зернистости, которые сейчас так беспокоят исследователей в области фотографии. Часто бывает очень полезно щелкнуть камерой и сразу же взглянуть на фотографию.

Ныне используется и другой процесс, тоже медленный и громоздкий. На протяжении пятидесяти лет используется бумага, темнеющая в том месте, где ее коснется электрический контакт, благодаря химической реакции с йодистым составом, которым она пропитана. Эта технология может быть использована для записи, с помощью указателя, оставляющего за собой след. Электрический потенциал указателя может меняться, делая линию светлее или темнее.

Эта схема сейчас используется при передаче факсимильных изображений. Указатель рисует на бумаге набор близко расположенных линий через весь лист, одну за другой. Во время движения электрический потенциал указателя изменяется в зависимости от переменного тока, передаваемого по проводу с удаленной станции, где эти изменения генерирует фотоэлемент, точно так же сканирующий изображение. В каждый момент времени яркость рисуемой линии такая же, как яркость точки изображения, сканируемого фотоэлемен-

том. Когда все изображение будет перенесено, его копия появится на принимающей стороне. <…>

Микрофотографии, как и сухой фотографии, предстоит пройти еще долгий путь. <…> Предположим, что в будущем будет достигнуто стократное уменьшение. Представьте пленку толщиной с бумагу, хотя, несомненно, появятся и более тонкие пленки. Даже при таких условиях объем обычной книги будет больше ее копии на микропленке в 10 000 раз. Британская энциклопедия уместится в спичечном коробке, библиотека, насчитывающая миллионы томов, – на одном краю рабочего стола. Если с момента изобретения наборного шрифта человечество произвело – в виде журналов, газет, книг, брошюр, рекламных буклетов, корреспонденции – объем, сопоставимый с миллиардом книг, то все это, должным образом скомпонованное и сжатое, спокойно поместится в передвижной фургончик. Одного сжатия, конечно же, недостаточно; требуется не только создать и хранить запись, но и обращаться к ней, этот аспект мы рассмотрим позже. Даже к самой лучшей современной библиотеке редко обращаются, лишь немногие пользуются ее благами.

Однако сжатие важно, когда речь заходит о стоимости. Материал для микрофильма, содержащего Британскую энциклопедию, обойдется в 5 центов, а за его доставку почтой придется заплатить около цента. Во сколько обойдется печать миллиона копий? Печать газетного листа при большом тираже стоит долю цента. Весь материал Британской энциклопедии в микрофильме поместится на листе размером восемь с половиной на одиннадцать дюймов. Как только это станет возможным благодаря будущим методам фотографического воспроизведения, крупнотиражное копирование, вероятно, будет стоить меньше цента за копию, не считая стоимости материалов. Подготовка оригинала к копированию? Тут мы переходим к обсуждению следующего аспекта.

11.3

Чтобы сделать запись, мы берем ручку или стучим по клавишам печатной машинки. Далее наступает черед корректуры и правки текста, а затем сложный процесс верстки, печати и распространения готовой продукции. Что касается первого этапа: быть может, будущий писатель перестанет писать или печатать, а будет непосредственно проговаривать текст? Сейчас его действия опосредованы, он диктует стенографистке или наговаривает на восковой цилиндр; но все элементы, необходимые для непосредственного преобразования речи в печатный текст, уже существуют. Нужно лишь воспользоваться имеющимися механизмами и изменить язык.

На прошедшей Всемирной выставке демонстрировалась машина Voder. Девушка нажимала на клавиши, и машина воспроизводила вполне распознаваемую речь. В тот момент человеческая речь в процедуре не участвовала; клавиши просто объединяли электрические колебания и передавали их громкоговорителю. В компании Bell Laboratories есть машина, делающая прямо противоположное, Vocoder. В ней громкоговоритель заменен микрофоном,

который воспринимает звук. Говоришь в него, и соответствующие клавиши двигаются. Это может стать одним из элементов вышеупомянутой системы.

Другой элемент мы находим в стенографической машине – том приводящем в замешательство устройстве, что встречается обычно на массовых мероприятиях. Девушка лениво стучит по своим клавишам, обводя тревожным взглядом зал, а иногда и лектора. Из машины ползет полоса напечатанного текста, на которой фонетически упрощенным языком записано все, что предположительно произносил оратор. Позже эта полоска перепечатывается на обычном языке, поскольку в исходном виде она понятна лишь посвященным. Объедините эти два элемента, пусть Vocoder приводит в действие стенографическую машину, и вот вам результат – машина, печатающая то, что произносится.

Существующие языки плохо приспособлены для такого рода механизации, это правда. Странно, что изобретатели универсальных языков до сих пор не наткнулись на идею создания такого языка, который был бы пригоден для технологий передачи и записи речи. Механизация, возможно, еще заставит решить этот вопрос, особенно в научной среде, где научный жаргон становится все менее понятным обывателю.

Давайте представим себе будущего исследователя в его лаборатории. Его руки свободны, сам он не привязан к месту. Перемещаясь и наблюдая, он одновременно фотографирует и оставляет комментарии. Время автоматически фиксируется, чтобы можно было связать последовательные записи. Находясь в экспедиции, он может связываться с записывающим устройством по радио. Разбирая свои заметки вечером, он точно так же может наговаривать комментарии к записанному. Распечатанные записи, равно как и фотографии, миниатюрны, поэтому для просмотра он использует проектор.

Но много чего должно произойти между сбором данных и наблюдений, извлечением смежного материала из существующих записей и окончательной вставкой нового материала в тело существующей записи. Тщательно обдуманной мысли нет механической замены. Но творческое мышление и мышление по существу рутинное – совершенно разные вещи. Для последнего вполне возможны и уже существуют механические вспомогательные средства.

Сложение столбца чисел – рутинный мыслительный процесс, и он давным-давно был поручен машине. Да, правда, машина иногда управляется с клавиатуры, а для чтения данных и нажатия нужных клавиш какое-никакое обдумывание необходимо, но даже этого можно избежать. Уже созданы машины, которые будут считывать напечатанные цифры с помощью фотоэлементов и нажимать соответствующие клавиши; это сочетание фотоэлементов для сканирования, электрических схем для сортировки получающихся в результате вариаций и релейных схем для перевода результатов в движение соленоидов, толкающих клавиши.

Все эти сложности необходимы из-за выученного нами неуклюжего способа записи цифр. Если бы мы записывали их позиционно, разными наборами точек на карточках, то автоматический механизм считывания оказался бы сравнительно простым. На самом деле если точки заменить дырочками, то мы получим перфокарточную машину, давным-давно построенную Холле-

ритом для переписи населения и до сих пор используемую в бизнесе сплошь и рядом. Некоторые крупные компании вряд ли могли бы работать без таких машин.

Сложение – лишь одна из операций. В арифметических вычислениях участвуют также вычитание, умножение и деление, а кроме того, какой-то метод временного хранения результатов, извлечения этих результатов из памяти для последующих вычислений и вывода итоговых результатов на печать. Сейчас есть два типа машин для решения этих задач: клавишные машины для бухгалтерских расчетов и схожих операций, с ручным контролем ввода данных и, как правило, со средствами автоматизации последовательности операций; и перфокарточные машины, в которых разные операции обычно делегированы нескольким машинам, а перфокарты физически передаются от одной машины к другой. Оба типа машин очень полезны, но с точки зрения сложных вычислений оба пока находятся в зачаточном состоянии.

Быстрый, основанный на электричестве подсчет появился вскоре после того, как физики сочли его необходимым для измерения космического излучения. Для своих собственных нужд физики оперативно создали оборудование на основе ламп с накаливаемым катодом, способное подсчитывать электрические импульсы частотой до 100 000 в секунду. Передовые арифметические машины будущего будут электрическими в своей основе и работать в сто и более раз быстрее существующих.

Более того, эти машины будут гораздо более гибкими, чем существующие коммерческие машины, так что их можно будет легко настраивать для решения широкого круга задач. Они будут управляться контрольной картой или пленкой, они будут выбирать данные самостоятельно и обрабатывать их в соответствии с полученными инструкциями, они будут производить сложные арифметические вычисления с чрезвычайно высокой скоростью и записывать результаты в форме, удобной для распространения или последующих манипуляций. Эти машины будут обладать чудовищным аппетитом. Даже одна из них будет способна принимать инструкции и данные, подаваемые целым залом девушек-операторов, вооруженных простыми клавишными перфораторами, и выдавать листы с результатами расчетов каждые несколько минут. Всегда найдется, что посчитать, когда миллионы людей заняты сложным делом.

11.4

Однако рутинные мыслительные процессы не ограничиваются арифметикой и статистикой. На самом деле всякий раз, как человек объединяет и записывает факты, следуя общепринятым логическим процессам, творческий акт имеет место только на стадии отбора данных, а последующий процесс и манипулирование по природе своей рутинны и потому вполне подходят для передачи машине. Если не считать арифметики, то в этом направлении сделано не так уж много, как могло бы быть сделано, в основном по экономическим причинам. Потребности бизнеса и стоящий наготове обширный

рынок привели к массовому производству арифмометров, как только методы производства получили достаточное развитие.

В случае машин для продвинутого анализа такой ситуации не было, ибо не было и нет никакого обширного рынка; пользователи продвинутых методов обработки данных составляют очень малую часть населения. Существуют, однако, машины для решения дифференциальных уравнений, да и функциональных и интегральных тоже, если на то пошло. Есть много специальных машин, например гармонический синтезатор для предсказания приливов. Будет и много других, но поначалу они, конечно, окажутся в руках ученых и в небольших количествах.

Если бы научные рассуждения были ограничены логическими процессами арифметики, то мы не продвинулись бы так далеко в понимании физического мира. Можно с таким же успехом попытаться освоить покер, пользуясь только математической теорией вероятностей. Абак со своими бусинами, нанизанными на параллельные проволоки, привел арабов к идее позиционной системы счисления и понятию нуля на много столетий раньше всего остального мира; и это был полезный инструмент – настолько полезный, что существует по сей день.

От абака до современного клавишного арифмометра длинный путь. И такой же путь отделяет нас от арифметической машины будущего. Но даже эта новая машина не сможет доставить ученого туда, куда ему нужно. Если люди желают использовать свои мозги для чего-то большего, чем рутинные преобразования по точно сформулированным правилам, то нужно освободить их и от трудоемкого дотошного применения высшей математики. Математик – это не тот, кто может без труда жонглировать числами, зачастую он этого как раз и не умеет. Это даже не тот, кто легко проделывает преобразования уравнений, применяя математический анализ. Это прежде всего человек, поднаторевший в искусном использовании символической логики, а главное, способный на интуитивный выбор процессов манипулирования фактами.

Что до всего остального, то у него должна быть возможность препоручить это механизму с такой же уверенностью, с какой он поручает запуск двигателя своего автомобиля сложному механизму, находящемуся под капотом. Только тогда математика сможет эффективно применить растущий объем знаний в области атомистики к решению трудных задач химии, металлургии и биологии. По этой причине появятся новые машины, которые будут выполнять сложные математические операции в интересах ученого. Некоторые из них окажутся достаточно необычными, чтобы удовлетворить самого разборчивого ценителя современных артефактов цивилизации.

11.5

Ученый, однако, не единственный, кто работает с данными и исследует окружающий мир с помощью логических процессов, хотя иногда они готовы создать такое впечатление, принимая в свою касту всякого, кто демонстрирует логическое мышление, точно так же, как лидера британских лейбористов

возводят в рыцарское звание. Всякий раз, как имеет место логический процесс – т. е. всякий раз, как мысль следует по проторенной дорожке, – есть возможность использовать машину. Формальная логика всегда была острым инструментом в руках учителя, испытывающего души своих учеников. Вполне возможно построить машину, которая будет манипулировать исходными предположениями в согласии с формальной логикой, просто благодаря изобретательному применению релейных схем.

Подайте набор исходных предположений такому устройству, нажмите на рычаг – и оно будет услужливо выдавать один вывод за другим, в полном соответствии с законами логики, ошибаясь не чаще, чем следует ожидать от клавишного арифмометра. [Примечание редактора: сравните со стр. 34.]

Логика может оказаться неимоверно трудной, и, без сомнения, было бы неплохо питать больше доверия к правильности ее применения. Машины для высшего анализа обычно создавались для решения уравнений. Начинают появляться идеи насчет преобразователей уравнений, которые будут изменять форму связи, выраженной уравнением, в соответствии со строгой и довольно сложной логикой. Прогресс тормозится на редкость неуклюжим способом выражения связей, применяемым математиками. Они используют символизм, который разросся, как Топси[1], и растерял внутреннюю согласованность – странное дело в этой самой логической из всех областей знания.

Разработка нового символизма, по всей видимости позиционного, очевидно, должна предшествовать сведению математических преобразований к машинным процессам. А далее, за границами строгой логики математики, лежит применение логики к повседневным материям. Быть может, когда-нибудь мы сможем вводить аргументы в машину с такой же уверенностью, с какой сегодня рассчитываемся с помощью кассового аппарата. Но машина логики не будет похожа на кассовый аппарат, даже самой обтекаемой формы.

Ну и хватит о манипулировании идеями и об их вставке в записи. Пока что все выглядит еще хуже, чем прежде, – ибо мы можем расширить записи до невероятных размеров, а между тем даже при нынешних размерах испытываем трудности с обращением к ним. Этот вопрос куда серьезнее, чем просто извлечение данных для целей научного исследования; он включает весь процесс извлечения человеком пользы из ранее накопленных и унаследованных им знаний. В этом процессе основное действие – отбор, и тут мы упираемся в стену. Возможно, существуют миллионы тончайших мыслей и повествований об опыте, на котором они основаны, и все это заключено в каменные стены приятной архитектурной формы; но если ученый способен добраться всего до одной в результате недели напряженных поисков, то его обобщения вряд ли будут поспевать за текущей ситуацией.

Отбор, в этом широком смысле, – это каменное тесло в руках столяра-краснодеревщика. Да, в узком смысле и в других областях какие-то действия, связанные с отбором, уже выполнялись механически. Сотрудник отдела кадров на фабрике кладет стопку из нескольких тысяч учетных карточек работников в сортировальную машину, выставляет код, отвечающий принятому согла-

---

[1] Топси – слониха, выступавшая в цирке американского антрепренёра Адама Форпо. – *Прим. перев.*

шению, и очень скоро получает список всех работников, которые проживают в Трентоне и знают испанский. Но даже такие устройства работают слишком медленно, когда нужно, например, найти один набор отпечатков пальцев среди пяти миллионов хранящихся в картотеке. Скорость работы таких сортировальных машин, которая сейчас составляет несколько сотен карточек в минуту, скоро будет увеличена. Благодаря использованию фотоэлементов и микропленки они смогут просматривать тысячу объектов в секунду и печатать копии отобранных.

Однако этот процесс – не более чем простой выбор: большой набор объектов просматривается по одному, и выбираются те, которые обладают заданными характеристиками. Существует другой вид выбора, который лучше проиллюстрировать на примере автоматического телефонного коммутатора. Вы набираете номер, а машина выбирает одного из миллионов абонентов и устанавливает соединение с ним. Она не перебирает всех подряд. Она смотрит только на класс, определяемый первой цифрой, затем только на подкласс, определяемый второй цифрой, и т. д., поэтому до выбранного абонента добирается быстро и почти без ошибок. На выбор уходит несколько секунд, хотя процесс можно было бы ускорить, если бы это было экономически оправдано. При необходимости его можно было бы сделать чрезвычайно быстрым, заменив механическую коммутацию коммутацией с применением вакуумных ламп с накаливаемым катодом, тогда выбор можно было бы выполнить за одну сотую долю секунды. Никто не хочет тратить деньги на внесение такого изменения в телефонные системы, но общая идея применима и в других случаях.

Возьмем прозаическую проблему большого универмага. Каждый раз, когда производится продажа в кредит, нужно сделать несколько вещей. Необходимо уменьшить количество товара на складе, продавец должен одобрить продажу, в главную бухгалтерскую книгу нужно внести запись и, самое главное, нужно списать средства с покупателя. Разработано центральное записывающее устройство, которое позволяет удобно делать большую часть этой работы. Продавец кладет на стойку идентификационную карточку покупателя, свою собственную карточку и карточку, снятую с проданного товара, – все это перфокарты. Когда он нажимает на рычаг, контакты проходят сквозь отверстия, машина в центральном офисе проделывает необходимые вычисления, и печатается чек, который продавец вручает покупателю.

Но с магазином может вести дела десять тысяч человек, покупающих в кредит, и для завершения операции кто-то должен выбрать правильную карточку и ввести ее в центральном офисе. Механизм быстрого выбора мог бы вставить нужную карточку в приемное устройство за секунду-другую, а потом вернуть ее на место. Но возникает другая трудность. Кто-то должен прочитать итог на карточке, чтобы машина могла прибавить к нему вычисленную стоимость покупки. Можно представить себе, что карточки изготовлены с помощью описанного выше сухого фотографического процесса. Тогда прежний итог может быть считан фотоэлементом, а новый введен с помощью электронного луча.

Карточки могут быть миниатюрными и не занимать много места. Они должны допускать быстрое перемещение – не обязательно далеко, просто

в нужную позицию, где с ними могут работать фотоэлемент и рекордер. Данные могут обозначаться позиционными точками. В конце месяца машина может прочитать их и напечатать обычный счет. Если для выбора используются электронные лампы, не содержащие никаких механических частей, то подготовка нужной карточки к операции не займет много времени – на всю операцию хватит и секунды. При желании запись на карточке может быть целиком закодирована магнитными точками на стальной пластине, а не оптически видимыми точками; для этого можно воспользоваться схемой, которую Поулсен давным-давно предложил для записи речи на магнитную проволоку. Преимущества этого метода – простота и легкость стирания. Зато фотография позволяет проецировать увеличенную запись на некоторое расстояние с помощью процесса, применяемого в телевизионном оборудовании.

Применение такой быстрой выборки и дистанционного проецирования можно рассмотреть и для других целей. Способность выбирать один листок из миллиона за одну-две секунды и затем что-то добавлять в него наводит на многие мысли. Ее можно было бы использовать даже в библиотеках, но это уже другая история. В любом случае, открывается много интересных возможностей. Можно было бы, к примеру, говорить в микрофон, как было описано выше в связи с управляемой голосом пишущей машинкой, и таким образом делать выборку. Уж, конечно, это было бы эффективнее обычного конторского служащего.

11.6

Но истинная проблема выбора лежит глубже, чем задержка во внедрении механизмов библиотеками или отсутствие инициативы по разработке устройств, необходимых для их использования. Наша неспособность добраться до нужной записи в значительной степени обусловлена искусственностью систем индексирования. Когда данные любого вида помещаются в хранилище, они расставляются в алфавитном или числовом порядке, а для нахождения информации (если она вообще имеется) переходят от одного подкласса к другому. Нужная информация может находиться только в одном месте (при отсутствии дубликатов); необходимы правила относительно выбора пути, на котором ее искать, а эти правила громоздкие. Более того, отыскав один объект, человек должен выйти из системы и заново ввести следующий путь.

Мозг человека работает не так. Он действует по ассоциации. Завладев одним объектом, он мгновенно переходит к следующему, подсказанному мысленной ассоциацией, руководствуясь запутанной паутиной тропинок, переносимых клетками мозга. Конечно, этот процесс имеет иные характеристики; тропинки, по которым ходят нечасто, зарастают, объекты не хранятся постоянно, память преходяща. И тем не менее сложность тропинок, детальность мысленных образов приводят в восхищение, ничто в природе не может с этим сравниться.

Нет надежды, что человек сможет воспроизвести этот мыслительный процесс искусственно, но он, безусловно, должен извлечь из него уроки. В чем-то его даже можно усовершенствовать, потому что наши записи относительно постоянны. Но первая идея, вытекающая из этой аналогии, касается выбора. Выбор по ассоциации, а не путем индексирования, еще предстоит механизировать. Нельзя рассчитывать, что по скорости и гибкости мы сравняемся с мозгом, скачущим по тропинкам ассоциаций, но должна быть возможность уверенно обогнать мозг в части постоянства и отчетливости объектов, извлекаемых из хранилища.

Рассмотрим будущее устройство для индивидуального пользования, своего рода механизированную личную картотеку и библиотеку. Его нужно как-то назвать, и слово «мемэкс» (memex)[1] кажется подходящим. Мемэкс позволяет человеку хранить все свои книги, записи и корреспонденцию и механизирован таким образом, что к нему можно обращаться с огромной скоростью и гибкостью. Это персональное расширение памяти владельца.

Он представляет собой конторку и, хотя предположительно допускает дистанционную работу, прежде всего встраивается в мебель. Сверху расположены наклонные светящиеся экраны, на которые можно проецировать материал для удобства чтения. Имеется клавиатура, а также кнопки и рычаги. В остальном он выглядит как обычное рабочее место.

В одном конце хранятся материалы. Вопрос объема решен благодаря улучшенной технологии микрофильмирования. Лишь небольшая часть внутреннего пространства мемэкса отведена под хранилище, все остальное – механизм. И тем не менее если пользователь будет добавлять по 5000 страниц материалов в день, на заполнение хранилища до отказа понадобятся сотни лет, так что он может не экономить и вводить материалы, ни в чем себя не ограничивая.

Большая часть содержимого мемэкса приобретается в виде микрофильмов, готовых для ввода. Книги всех сортов, текущая периодика, газеты – все это покупается и укладывается на место. Туда же отправляется деловая переписка. И есть возможность прямого ввода. В верхней части мемэкса расположена прозрачная пластина. На нее можно класть обычные заметки, фотографии, памятные записки и прочее. Положив предмет, нужно нажать рычаг – и фотография предмета окажется в первой свободной позиции определенного участка пленки мемэкса, при этом используется сухой фотографический процесс. Разумеется, оставлена возможность обратиться к записи при помощи обычной схемы индексирования. Если пользователь захочет обратиться к определенной книге, он введет ее код на клавиатуре, и перед его глазами быстро появится титульная страница книги, спроецированная в одну из позиций для просмотра. Часто используемые коды запоминаются, поэтому ему редко приходится заглядывать в кодовую книгу, но если все-таки такая необходимость возникла, то она вызывается одним нажатием клавиши. В распоряжении пользователя есть и дополнительные рычаги. Отклоняя один из них вправо, он просматривает открытую перед ним книгу – страницы переворачиваются с такой скоростью, чтобы как раз

[1] Memory extender – расширитель памяти. – *Прим. перев.*

хватило времени понять, то ли это, что нужно. Если отклонить рычаг вправо сильнее, то книга будет перелистываться порциями по 10 страниц, если еще сильнее – то по 100. Отклонение рычага влево перелистывает книгу в обратном направлении.

Специальная кнопка переносит пользователя сразу на первую страницу индекса. Так он сможет найти и обратиться к любой книге из своей библиотеки гораздо быстрее и удобнее, чем если бы пришлось снимать ее с полки. Поскольку положений для проецирования несколько, пользователь может, не убирая одну книгу, просматривать другую. Он может оставлять заметки на полях, пользуясь преимуществами одного из видов сухой фотографии. Возможно, даже организовать процесс, так чтобы можно было пользоваться стилосом, как это сейчас делается в телеавтографах, которые можно увидеть в залах ожидания на железной дороге; тогда пользователь будет работать так, будто перед ним находится физическая страница.

11.7

Все это известные вещи, мы просто спроецировали на будущее механизмы и приспособления, применяемые уже сегодня. Однако это первый шаг к ассоциативному индексированию, основная идея которого – сделать так, чтобы любой объект позволял по желанию немедленно и автоматически выбирать следующий. В этом и состоит главная особенность мемэкса. Процесс связывания двух объектов – очень важная вещь.

Прокладывая тропинку, пользователь дает ей имя, вставляет имя в свою кодовую книгу и набирает его на клавиатуре. Перед ним в соседних позициях просмотра находятся два объекта, подлежащих связыванию. В нижней части каждого имеется несколько пустых мест для ввода кода, и указатель отмечает одно из них в каждом объекте. Пользователь нажимает всего одну клавишу, и объекты связываются постоянной связью. В месте для кода появляется кодовое слово. Вне поля зрения, но тоже в место для кода вставляются точки для распознавания фотоэлементом; в каждом объекте расположение этих точек идентифицирует номер другого объекта в индексе.

Впоследствии всякий раз, как один из этих объектов выводится для просмотра, можно будет мгновенно вызвать другой, нажав кнопку под соответствующим местом для кода. Более того, если таким образом было связано несколько объектов, образующих тропинку, их можно будет просматривать по очереди, быстро или медленно, отклоняя в сторону рычаг, который используется и для перелистывания книги. Все выглядит так, будто несколько физических объектов было собрано из территориально разнесенных источников и переплетено в один том. И даже лучше, потому что один и тот же объект может участвовать в нескольких тропинках.

Допустим, владельца мемэкса интересует происхождение и свойства лука и стрел. Точнее, он хочет знать, почему короткий турецкий лук превосходил длинный английский во время крестовых походов. В его мемэксе десятки книг и статей, которые могут относиться к этой теме. Сначала он просматри-

вает энциклопедию, находит интересную, но поверхностную статью и оставляет ее на экране. Затем, в разделе истории, он находит еще одну статью на эту тему и связывает оба объекта. Продолжая, он выстраивает тропинку, содержащую много объектов. Время от времени он вставляет собственные комментарии – либо включая их в основную тропинку, либо помещая в отдельную тропинку, отходящую от некоторого объекта. Когда становится понятно, что упругие свойства доступных материалов имеют самое непосредственное отношение к луку, он создает боковую тропинку, которая проведет по учебникам по теории упругости и таблицам физических постоянных. Он вставляет рукописную страницу с собственным анализом вопроса. Так он строит в лабиринте доступных материалов тропинку, представляющую для него интерес.

И его тропинки не зарастают. Прошло несколько лет – и в разговоре с приятелем всплыла тема о том, как люди противятся новшествам, даже представляющим жизненный интерес. И у него наготове пример – как разъяренные европейцы так и не приняли турецкий лук. У него даже есть тропинка на эту тему. Одним касанием он вызывает на экран кодовую книгу. Еще несколько ударов по клавишам – и на экране появляется начало цепочки. Рычаг позволяет пройти по ней, останавливаясь в интересных местах и исследуя боковые ответвления. Очень интересная тропинка, относящаяся к теме обсуждения. Поэтому он запускает дубликатор, фотографирует всю тропинку и передает ее приятелю, чтобы тот вставил ее в собственный мемэкс, включив в более общую тропинку.

11.8

Появятся совершенно новые виды энциклопедий, пронизанные сетью ассоциативных тропинок, которые легко вставить в мемэкс и там дополнить. В распоряжении юриста имеются мнения и решения по делам за все время его практики, а также мнение и решения его коллег и органов власти. Патентный поверенный может обратиться к миллионам выданных патентов, имея знакомые тропинки к каждому пункту, представляющему интерес для его клиента. Врач, озадаченный реакциями пациента, находит тропинку, созданную при исследовании похожего случая, и быстро просматривает аналогичные истории болезни с отсылками к классикам по относящимся к делу вопросам анатомии и гистологии. Химик, работающий над синтезом органического соединения, имеет перед собой в лаборатории всю литературу по химии, испещренную тропинками, посвященными аналогичным соединениям с боковыми ответвлениями на тему их физического и химического поведения.

Историк, располагающий обширными хронологическими сведениями о народах, проводит параллели с краткой тропинкой, останавливающейся только на значительных событиях, и в любой момент может проследовать по тропинкам, охватывающим то же время, что даст ему обзор всей цивилизации в конкретную эпоху. Появится новая профессия пролагателей тропинок, кото-

рые находят высшую радость в установлении полезных тропинок в огромном массиве общедоступных записей. Наследием мастера становится не только его вклад в мировую копилку; для его последователей интерес представляют и леса, с помощью которых было возведено созданное им здание.

Таким образом, наука может воплотить в жизнь те способы, которыми человек производит, хранит и обращается к опыту всего человечества. Было бы крайне интересно дать более впечатляющий общий обзор инструментальных средств будущего, а не рабски следовать за уже известными методами и элементами, которые претерпевают быстрое развитие, как было сделано в этой статье. Конечно, были оставлены за бортом технические трудности разного рода, но также проигнорированы средства, доселе неизвестные, которые могут появиться в любой день и ускорить технический прогресс так резко, как это случилось после появления электронных ламп с накаливаемым катодом. Чтобы картина не казалась совсем уж обыденной, привязанной к реалиям нынешнего дня, стоит упомянуть одну такую возможность, не пророчествовать, а просто предположить, поскольку пророчество, базирующееся на продолжении известного, имеет под собой основание, тогда как пророчества, базирующиеся на неизвестном, – всего лишь догадка в квадрате.

Все наши шаги по созданию или потреблению сущности записи подразумевают использование одного из органов чувств – осязания, когда мы касаемся клавиш; слуха, когда мы говорим или слушаем; зрения, когда мы читаем. Но нельзя ли представить себе, что когда-нибудь может быть установлен более прямой путь? Мы знаем, что когда глаз смотрит, вся последующая информация передается в мозг посредством электрических колебаний в канале зрительного нерва. Это точная аналогия с электрическими колебаниями в телевизионном кабеле: они передают информацию от фотоэлементов, которые видят картинку, к радиопередатчику, который ее вещает. Мы также знаем, что если приблизиться к кабелю с подходящими инструментами, то к нему даже не нужно прикасаться; мы можем снять колебания с помощью электрической индукции и таким образом получить и воспроизвести передаваемую сцену – точно так же, как можно прослушать сообщение, передаваемое по телефонному проводу. Импульсы, циркулирующие в нервах руки машинистки, передают к ее пальцам информацию, поступающую через глаза или уши, чтобы пальцы могли нажать на соответствующие клавиши. Нельзя ли перехватить эти токи, либо в исходной форме, в которой информация поступает мозгу, либо в чудесным образом преобразованной, в которой они передаются руке?

Пользуясь костной проводимостью, мы уже умеем вводить звуки в нервные каналы глухого, чтобы он мог слышать. Нельзя ли научиться вводить их не таким громоздким способом, как сейчас: без предварительного преобразования электрических колебаний в механические, которые организм человека тут же преобразует обратно в электрические? Если закрепить два электрода на черепе, то перо энцефалографа рисует кривые, имеющие отдаленную связь с электрическими явлениями, протекающими в мозге. Да, эта энцефаллограмма невнятна и показывает лишь серьезные отклонения в мозговой деятельности; но кто сейчас осмелится сказать, насколько далеко способна завести эта идея?

Во внешнем мире все виды разумной деятельности, как звуковые, так и зрительные, сведены к той или иной форме переменных токов в электрической цепи, с тем чтобы их можно было передать по проводам. Внутри человеческого тела протекают точно такие же процессы. Обязательно ли выполнять преобразование в механическую форму, чтобы перейти от одного электрического явления к другому? Эта мысль заслуживает обдумывания, но вряд ли может служить основой для предсказания, если мы не хотим утратить связь с реальностью и скоростью воплощения.

Видимо, духу человеческому надлежит возвыситься, чтобы лучше обозреть свое темное прошлое и более полно и объективно проанализировать стоящие перед ним проблемы. Человек построил цивилизацию столь сложную, что теперь нуждается в механизации своих записей, если хочет довести свой эксперимент до логического завершения, а не просто застрять на полпути, чрезмерно обременив свою ограниченную память. Его изыскания станут более приятными, если он вновь обретет право забывать многообразные вещи, которые не нужны прямо сейчас, но будет уверен, что сможет найти их, когда понадобится.

Наука позволила человеку построить хорошо обеспеченный дом и учит его вести в нем здоровый образ жизни. Она же дала возможность бросать массы людей друг против друга, снабдив их смертоносным оружием. Но, быть может, она позволит ему охватить разумом и расширить огромный опыт, накопленный всем человечеством. Возможно, он погибнет в каком-нибудь катастрофическом конфликте, прежде чем научится использовать этот опыт во благо. Но сейчас применение науки к удовлетворению потребностей и желаний человека находится на такой стадии, что было бы очень жаль прервать процесс или утратить надежду на благополучный исход.

12 Математическая теория связи (1948)

Клод Шеннон

После защиты магистерской диссертации (глава 8) Клод Шеннон разработал алгебру для вычислений в менделевской генетике и представил докторскую диссертацию на эту тему в 1940 году. Затем он перешел в компанию Bell Labs, где изобрел теорию информации, изложенную в этой статье. Преимущества двоичной системы счисления к тому времени уже были известны: группа, работавшая над EDVAC, выбрала ее для организации внутренней памяти и арифметических операций. В «Первой редакции» недвусмысленно утверждается: «Единица (емкости) памяти – это способность сохранять значение одной двоичной цифры» (стр. 143 этой книги). Шеннон придумал для этой единицы ставшее знаменитым название «бит», хотя приписывает эту честь математику Джону Тьюки. Затем Шеннон изучает вопрос о том, как закодировать длинное сообщение относительно небольшим числом битов, при условии что не все сообщения возможны или равновероятны. Эта работа легла в основу не только теории сжатия информации и определения его пределов (например, кода Хаффмана [Huffman 1952]), но и важной теории восстановления сообщений в случае искажения нескольких битов на пути от источника к приемнику (см. главу 13). Термин «энтропия», ныне также вошедший в обиход, заимствован из статистической физики и здесь представляет меру количества информации в источнике.

Позже Шеннон опубликовал свой третий шедевр, «Теория связи в секретных системах» (Shannon 1949), в котором сформулировал некоторые принципы криптографии с теоретико-информационной точки зрения. Статья была основана на засекреченной работе, выполненной по контракту между Bell и Министерством обороны США. Шеннон вернулся в МТИ на должность профессора в 1958 году и еще долго оставался задорным умельцем-изобретателем. Он жонглировал, катался на уницикле, а иногда делал то и другое

одновременно. Он построил машину, которая умела жонглировать, и запрограммировал механическую мышь находить выход из лабиринта.

Но в конечном итоге его математическая плодовитость пошла на убыль, он перестал публиковаться и преподавать и в 2001 году умер от болезни Альцгеймера в возрасте 84 лет.

12.0. Введение

Недавнее развитие различных методов модуляции, таких как PCM или PPM, основанных на улучшении пропускной способности за счет уменьшения отношения «сигнал–шум», привело к возрастанию интереса к общей теории связи [Примечание редактора: импульсно-кодовая модуляция и фазово-импульсная модуляция.]. Основы такой теории заложены в важных статьях Найквиста (Nyquist 1924, 1928) и Хартли (Hartley 1928). В настоящей статье мы расширим эту теорию, учтя некоторый ряд новых факторов, в частности наличие шума в канале, а также экономию, достигаемую вследствие статистической структуры исходного сигнала и природы конечного приемника информации.

Основной задачей связи является восстановление (точное или приближенное) в данной точке сигнала, отправленного из другой точки. Часто сигнал имеет некое значение, т. е. соотносится или коррелирует с заданными состояниями некоторой системы. Однако эти семантические аспекты связи не имеют отношения к инженерной задаче, важно лишь то, что сообщение *выбирается из* некоторого набора возможных. Система связи должна одинаково хорошо работать со всеми возможными вариантами сообщения, а не только с тем, который будет выбран в действительности – на этапе разработки системы это еще неизвестно.

Если набор содержит конечное число сообщений, то само это число или любая его монотонная функция может служить мерой информации, получаемой при выборе одного из них, если мы примем все варианты равновероятными. Как было отмечено Хартли, наиболее естественно выбрать логарифмическую функцию. И хотя это определение должно быть существенно обобщено на случаи неравновероятных сообщений и их непрерывных множеств, мы во всех случаях будем использовать именно логарифмические меры.

Логарифмическая мера удобна по следующим причинам:

1. Она удобна практически. Параметры, важные в инженерных приложениях, такие как время, пропускная способность, число переключателей и т. д., обычно линейно зависят от логарифма числа возможных вариантов. К примеру, добавление одного переключателя в группу удваивает число возможных состояний переключателей, увеличивая на единицу его двоичный логарифм. Увеличение в два раза времени приводит к квадратичному росту числа сообщений, или удвоению логарифма этого числа, и т. д.

2. Она близка к нашему интуитивному представлению о правильной мере. Это тесно связано с предыдущим пунктом, так как мы интуитивно измеряем величины, линейно сравнивая их со стандартами. Так, нам кажется, что на двух перфокартах можно разместить в два раза больше информации, а по двум одинаковым каналам передать ее в два раза больше.
3. Она удобна математически. Многие предельные переходы просты в терминах логарифмов, в то время как в терминах числа вариантов требуют громоздкого переформулирования.

Выбор основания логарифма соответствует выбору единицы измерения информации. Если взять основание 2, то полученные единицы можно назвать двоичными цифрами, или *битами*, это слово было предложено Дж. У. Тьюки. Устройство с двумя устойчивыми состояниями, такое как переключатель или триггер, способно хранить один бит информации, N таких устройств – N бит, т. к. полное число состояний равно 2^N, а $\log_2 2^N = N$. При использовании же основания 10 единицы можно назвать десятичными цифрами. Так как

$$\log_2 M = \log_{10} M / \log_{10} 2 = 3.32 \log_{10} M,$$

десятичной цифре соответствует примерно $3\frac{1}{3}$ бита. Цифровое колесо арифмометра имеет 10 устойчивых состояний и потому может хранить одну десятичную цифру информации. В аналитических расчетах, при проведении дифференцирования и интегрирования, бывает полезно использовать натуральный логарифм по основанию e. Соответствующие единицы информации можно назвать натуральными. Переход от основания a к основанию b сводится к умножению на $\log_b a$.

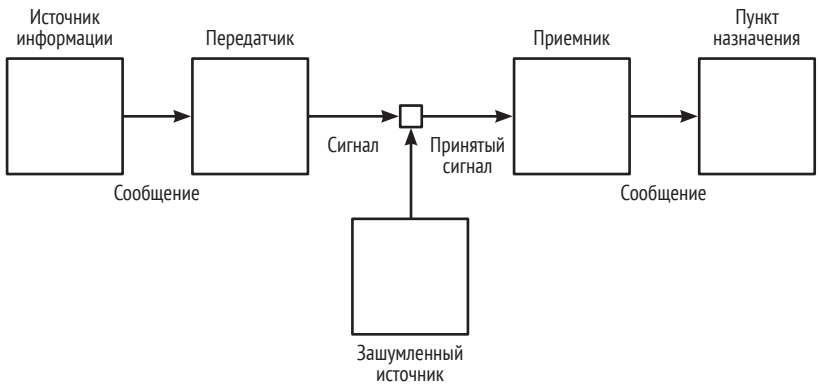


Рис. 1.2. Схематическое изображение общей системы связи

Под системой связи мы будем понимать систему типа изображенной на рис. 12.1. Такая система состоит из пяти частей:

1) *источника информации*, испускающего сообщение или серию сообщений для связи с приемной станцией. Сообщения могут быть различ-

ных типов: (а) последовательностью букв, как в телеграфной системе; (б) некоторой функцией времени $f(t)$, как в системе телефонной связи; (в) функцией времени и других переменных, как в телевизионных системах, – такое сообщение можно рассматривать как функцию $f(x, y, t)$ двух пространственных координат и времени, представляющую интенсивность излучения в точке (x, y) в момент времени t на фотокатоде; (г) двумя или более функциями времени, к примеру $f(t)$, $g(t)$, $h(t)$, как в случае передачи «трехмерного звука» или просто нескольких сигналов одновременно; (д) несколькими функциями нескольких переменных – при передаче цветного телевизионного сигнала мы имеем дело с тремя функциями $f(x, y, t)$, $g(x, y, t)$, $h(x, y, t)$, определенными на трехмерном множестве (эти три функции можно также рассматривать как компоненты трехмерного векторного поля в пространстве); (е) могут также возникать различные комбинации сигналов, например при передаче телевизионного изображения совместно со звуковым каналом;

2) *передатчика*, преобразующего неким образом сигнал и делающего его пригодным для передачи по линии связи. В телефонии это сводится главным образом к преобразованию давления звука в пропорциональный электрический сигнал. В случае телеграфа процесс кодирования сигнала преобразует его в последовательность точек, тире и пробелов в линии связи. В многоканальной PCM-системе различные речевые функции должны быть дискретизованы, сжаты, закодированы и, наконец, сведены надлежащим образом в одно сообщение. Вокодеры (устройства цифрового кодирования речи), телевидение и частотная модуляция – другие примеры сложных преобразований, которым может быть подвергнуто сообщение при формировании сигнала;
3) *канала*, представляющего собой среду, используемую для передачи сигнала от передатчика к приемнику. Это может быть пара проводов, коаксиальный кабель, радиоволны определенной частоты, луч света и т. д.;
4) *приемника*, выполняющего задачи, противоположные выполняемым передатчиком, а именно восстановление сообщения из сигнала;
5) *пункта назначения* – человека (или устройства), для которого сообщение предназначено.

Мы хотели бы рассмотреть некоторые общие задачи, имеющие отношение к системам связи. Для этого необходимо сначала представить различные вышеупомянутые элементы математическими величинами, отвлекаясь от их физической сущности. Системы связи можно грубо разделить на три категории: дискретные, непрерывные и смешанные. Под дискретной системой будем понимать такую, в которой и сообщение, и сигнал являются последовательностями дискретных символов. Типичным примером такой системы является телеграфия, в которой и сообщение, и сигнал являются последовательностями точек, тире и промежутков между ними. Непрерывная система – такая, в которой и сообщение, и сигнал рассматриваются как непрерывные функции, например в радио или телевидении. В смешанной

системе, соответственно, фигурируют как непрерывные, так и дискретные величины, как, например, в PCM-системе передачи речи.

Рассмотрим вначале дискретный случай. Он имеет приложения не только в теории связи, но также и в теории вычислительных машин и т. д. Вдобавок дискретные системы являются основой для рассмотрения непрерывных и смешанных, которые разбираются во второй половине статьи.

12.1. Дискретный канал без шума

Телетайп и телеграф – два простых примера дискретных каналов для передачи информации. В общем случае будем называть дискретным каналом систему, посредством которой последовательность, выбранная из набора элементарных символов $S_1, \ldots, S_n$, может быть передана из одной точки в другую. Предполагается, что каждый из этих символов S_i имеет некоторую протяженность во времени t_i секунд (не обязательно одинаковую для различных символов S_i, например точек и тире в телеграфии). Не требуется возможность передачи произвольной последовательности символов, вполне допустим вариант с ограниченным набором разрешенных последовательностей. Это возможные сигналы в канале. К примеру, в телеграфии такими символами являются: (1) точка, соответствующая замыканию линии на единичное время и размыканию той же длительности, (2) тире, соответствующее замыканию линии на три единицы времени и размыканию на одну, (3) промежуток между буквами, соответствующий, к примеру, размыканию линии на три единицы времени, и (4) промежуток между словами, соответствующий размыканию линии на шесть единиц времени. Мы можем ограничить возможные последовательности символов, запретив, к примеру, последовательные промежутки (т. к. два следующих друг за другом буквенных промежутка идентичны промежутку между словами). Вопрос, который мы сейчас рассмотрим, состоит в том, как определить пропускную способность такого канала.

В случае телетайпа, в котором все символы имеют одну и ту же длительность и разрешена любая комбинация 32 символов, ответ прост. Каждый символ представляет собой пять бит информации. Если система связи передает n символов в секунду, естественно сказать, что пропускная способность канала равна $5n$ бит в секунду. Это не значит, что телетайп всегда передает информацию с такой скоростью – это лишь максимально возможный темп, а будет он достигаться или нет, зависит от источника информации, подаваемой в канал (см. далее). Для более общего случая различной длины символов и ограничений на разрешенные последовательности дадим следующее определение:

Определение. Пропускная способность C дискретного канала дается выражением

$$C = \lim_{T \to \infty} \frac{\log N(T)}{T},$$

где N(T) – число возможных сигналов длительности T.

Легко видеть, что в случае телетайпа этот результат сводится к предыдущему. Можно показать, что этот предел в большинстве интересующих нас случаев существует и является конечным. Предположим, что разрешены все последовательности символов $S_1, \ldots, S_n$ длительностью $t_1, \ldots, t_n$ соответственно. Какова пропускная способность канала? Если $N(t)$ – число последовательностей длительности t, то имеем

$$N(t) = N(t - t_1) + N(t - t_2) + \ldots + N(t - t_n).$$

Полное число равно сумме чисел последовательностей, оканчивающихся на $S_1, S_2, \ldots, S_n$, т. е. $N(t - t_1), N(t - t_2), \ldots, N(t - t_n)$ соответственно. Согласно широко известному результату теории конечных разностей, $N(t)$ при больших t асимптотически стремится к X_0^t, где X_0 – наибольший действительный корень характеристического уравнения $X^{-t_1} + X^{-t_2} + \ldots + X^{-t_n} = 1$ и, следовательно, $C = \log X_0$.

При наличии ограничений на разрешенные последовательности символов мы зачастую также можем получить разностное уравнение такого типа и найти C из характеристического уравнения. В вышеупомянутом случае телеграфной системы

$$N(t) = N(t - 1) + N(t - 4) + N(t - 5) + N(t - 7) + N(t - 8) + N(t - 10),$$

что получается путем подсчета последовательностей символов с заданным последним или предпоследним символом. Следовательно, равно $-\log \mu_0$, где μ_0 – положительный корень уравнения $1 = \mu^2 + \mu^4 + \mu^5 + \mu^7 + \mu^8 + \mu^{10}$. Решая его, находим C = 0.539.

Достаточно общим видом ограничений на возможные последовательности символов может быть следующий. Рассмотрим множество возможных состояний $a_1, a_2, \ldots, a_m$. В каждом состоянии могут передаваться лишь некоторые символы из набора $S_1, \ldots, S_n$ (различные подмножества в различных состояниях). При передаче одного из этих символов состояние изменяется в зависимости как от предыдущего состояния, так и от переданного символа. Простым примером такой системы является телеграф, который может находиться в одном из двух состояний в зависимости от того, был ли последним переданным символом пробел. Если да, то может быть послана только точка или тире, и состояние изменится на противоположное. Если же нет, может быть послан произвольный символ, и состояние изменится, если посланный символ – пробел. Эта ситуация иллюстрируется линейным графом, изображенным на рис. 12.2. Вершины графа соответствуют состояниям, а ребра – разрешенным для передачи символам. В приложении 1 [Примечание редактора: опущено.] показано, что при таких ограничениях на разрешенные к передаче символы C существует и может быть вычислена в соответствии со следующим результатом.

Теорема 1. Пусть $b_{ij}^{(s)}$ – длительность s-го символа, разрешенного в состоянии i и приводящего к переходу системы в состояние j. Тогда пропускная способность канала C равняется $\log W$, *где W – наибольший действительный корень уравнения, содержащего определитель*

$$\left|\sum_s W^{-b_{ij}^{(s)}} - \delta_{ij}\right| = 0,$$

где $\delta_{ij} = 1$, *если* $i = j$, *и равно* 0 *в противном случае.*

Например, в случае телеграфа (рис. 12.2) определитель равен

$$\begin{vmatrix} -1 & (W^{-1} + W^{-4}) \\ (W^{-3} + W^{-6}) & (W^{-2} + W^{-4} - 1) \end{vmatrix} = 0,$$

что после раскрытия дает приведенное выше уравнение для данного случая.

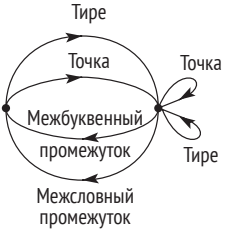


Рис. 12.2. Графическое представление ограничений на последовательности телеграфных символов

12.2. Дискретный источник информации

Мы видели, что при достаточно общих условиях логарифм числа возможных сигналов в дискретном канале линейно растет со временем. Пропускная способность канала может быть охарактеризована темпом этого роста, числом битов в секунду, необходимым для передачи данного конкретного сигнала. Рассмотрим теперь источник информации. Как он может быть описан математически и сколько битов информации в секунду он производит? Тут важно знать статистику этого источника, что позволяет понизить требуемую пропускную способность канала выбором соответствующего способа кодирования информации. В телеграфии, к примеру, передаваемые сообщения состоят из последовательностей букв. Эти последовательности, однако, не являются совершенно случайными. В общем случае они образуют предложения и имеют статистическую природу, характерную, скажем, для английского языка. Буква E встречается чаще Q, последовательность TH – чаще, чем XP. Наличие такой структуры позволяет получить выигрыш во времени передачи сообщения (или пропускной способности канала) за счет подходящей кодировки последовательностей сообщений в последовательности сигналов. Именно такой подход используется в телеграфии, где самый короткий символ, точка, применяется для наиболее часто встречающейся английской буквы E, в то время как самые редкие, Q, X, Z, представляются более длинными последовательностями точек и тире. Еще больший выигрыш во времени

передачи достигается в некоторых коммерческих системах кодирования, в которых наиболее распространенные фразы и выражения заменяются четырех- или пятибуквенными комбинациями. Использующиеся в настоящее время стандартизованные поздравительные и приветственные телеграммы также позволяют кодировать целые предложения достаточно короткими последовательностями чисел.

Можно считать, что дискретный источник формирует сообщение символ за символом, выбирая их в соответствии с некоторыми вероятностями, зависящими как от текущего символа, так и от выбранных ранее. Физическая система, или же математическая модель системы, генерирующей такую последовательность символов в соответствии с набором вероятностей, представляет собой стохастический процесс (см., например, Chandrasekhar [1943]). Таким образом, мы можем рассматривать дискретный источник как стохастический процесс. И наоборот, любой стохастический процесс, генерирующий дискретную последовательность символов из ограниченного множества, можно считать дискретным источником. Это включает в себя:

1) естественные языки, такие как английский, немецкий и китайский;
2) непрерывные источники информации, которая может быть дискретизована в результате некоторой операции, например квантованная речь в PCM-передатчике, или квантованный телевизионный сигнал;
3) чисто математические случаи, когда мы абстрактно определяем некоторый стохастический процесс, генерирующий последовательность символов. Приведем несколько примеров этого типа.

   (A) Пусть имеется пять букв A, B, C, D, E, которые выбираются с равной вероятностью 0.2, причем каждый следующий выбор не зависит от предыдущих. Тогда типичной последовательностью символов будет, к примеру,

   B D C B C E C C C A D C B D D A A E C E E A
   A B B D A E E C A C E E B A E E C B C E A D.

   Данная последовательность была построена с использованием таблицы случайных чисел (Kendall 1939).

   (B) Используя те же самые пять букв, но с вероятностями 0.4, 0.1, 0.2, 0.2, 0.1 соответственно, получаем следующее типичное сообщение:

   A A A C D C B D C E A A D A D A C E D A
   E A D C A B E D A D D C E C A A A A A D.

   (C) Более сложная структура может быть получена при отказе от взаимной независимости отдельных актов выбора – т. е. когда каждый последующий символ зависит от предшествующих. В простейшем случае каждый символ зависит лишь от предыдущего и не зависит от более ранних. Статистическая структура тогда может быть представлена набором вероятностей перехода $p_i(j)$, т. е. вероятностей того, что за символом i будет следовать символ j. Вторым эквивалентным методом описания такой структуры являются относительные вероятности «диграмм» (двухбуквенных комбина-

ций) $p(i, j)$. Частоты встречаемости букв $p(i)$ (вероятность буквы i), вероятности перехода $p_i(j)$ и вероятности диграмм $p(i, j)$ связаны следующими формулами:

$$p(i) = \sum_j p(i,j) = \sum_j p(j,i) = \sum p(j)p_j(i);$$
$$p(i,j) = p(i)p_i(j);$$
$$\sum_j p_i(j) = \sum_i p(i) = \sum_{i,j} p(i,j) = 1.$$

Для примера возьмем три буквы A, B, C с такими таблицами вероятностей:

$p_i(j)$		j		
		A	B	C
i	A	0	$\frac{4}{5}$	$\frac{1}{5}$
	B	$\frac{1}{2}$	$\frac{1}{2}$	0
	C	$\frac{1}{2}$	$\frac{2}{5}$	$\frac{1}{10}$

i	$p(i)$
A	$\frac{9}{27}$
B	$\frac{16}{27}$
C	$\frac{2}{27}$

$p(i, j)$		j		
		A	B	C
i	A	0	$\frac{4}{15}$	$\frac{1}{15}$
	B	$\frac{8}{27}$	$\frac{8}{27}$	0
	C	$\frac{1}{27}$	$\frac{4}{135}$	$\frac{1}{135}$

Типичным сообщением от такого источника будет:

A B B A B A B A B A B A B A B B B A B B B B B A B A B A B A B A B B B A C A C A B B A B B B B A B B A B A C B B B A B A.

Следующим по сложности будет учет частот встречаемости триграмм, т. е. ситуация, при которой выбор текущего символа зависит лишь от двух предшествующих. В данной ситуации требуется знание вероятностей триграмм $p(i, j, k)$, или, что то же самое, вероятностей переходов $p_{ij}(k)$. При дальнейшем обобщении можно получить и сигналы с более сложной структурой. Так, в общем случае для задания статистической структуры требуется знание набора вероятностей $p(i_1, i_2, \ldots i_n)$ или же вероятностей перехода $p_{i_1,i_1\ldots,i_{n-1}}(i_n)$.

(D) Можно также определить стохастический процесс, который генерирует текст, являющийся последовательностью «слов». Пусть имеется пять букв A, B, C, D, E и 16 «слов» языка с соответствующими вероятностями:

0.10 A	0.16 BEBE	011 CABED	0.04 DEB
0.04 ADEB	0.04 BED	0.05 CEED	0.15 DEED
0.05 ADEE	0.02 BEED	0.08 DAB	0.01 EAB
0.01 BADD	0.05 CA	0.04 DAD	0.05 EE

Пусть эти слова выбираются независимо и разделяются пробелом. Тогда типичным сообщением будет:

DAB EE A BEBE DEED DEB ADEE ADEE EE DEB BEBE BEBE BEBE ADEE BED DEED DEED CEED ADEE A DEED DEED BEBE CABED BEBE BED DAB DEED ADEB.

> Если все слова имеют ограниченную длину, то данный процесс сводится к одному из вышеописанных, однако описание его в терминах структуры и вероятностей слов будет проще. Можно пойти еще дальше и ввести вероятности перехода между словами и т. д.

Такие искусственные языки полезны для формулировки простых задач и примеров различных возможностей. Мы можем также приближенно описать некоторый естественный язык серией простых искусственных. Нулевым приближением подобного рода будет модель с равновероятными независимыми буквами. Первое приближение получается, если считать различные буквы независимыми, но распределенными с такими же вероятностями, как в естественном языке. (Вероятности букв, диграмм и триграмм приведены в работе Pratt [1939]. Частоты встречаемости отдельных слов приведены в работе Dewey [1923]). Так, в первом приближении для английского языка буква E будет выбираться с вероятностью 0.12, а W – с вероятностью 0.02, но не будет никакого влияния букв друг на друга и не будет заметна тенденция к образованию наиболее частых буквосочетаний, таких как TH или ED. Во втором приближении необходимо учесть структуру диграмм – после выбора одной буквы следующая выбирается уже согласно соответствующей условной вероятности, что требует знания частот диграмм $p_i(j)$. На следующем шаге вводится структура триграмм – каждая буква зависит уже от двух предшествующих.

12.3. Постепенное приближение к английскому языку

Для иллюстрации того, как эта серия приближений описывает естественный язык, были построены типичные последовательности приближений для английского языка. Во всех случаях использовался 27-символьный «алфавит»: 26 букв и пробел.

1. Нулевое приближение (символы независимы и равновероятны).

   XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGHYD QPAAMKBZAACIBZLHJQD.

2. Первое приближение (символы независимы, но частоты их соответствуют английскому тексту).

   OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL.

3. Второе приближение (диграммная структура, как в английском языке).

   ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE.

4. Третье приближение (триграммная структура, как в английском языке).

   IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE.

5. Четвертое приближение. Вместо дальнейшего использования тетра-

грамм, ..., *n*-грамм проще перейти к словарной структуре. В данном приближении слова независимы, но их частоты соответствуют английскому тексту (Dewey, 1923).

REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

6. Приближение второго порядка с использованием слов. Вероятности перехода между словами правильны, но более точной структуры нет.

   THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

С каждым шагом заметно возрастает сходство с английским языком. Заметим, что приведенные примеры сохраняют достаточно похожую на оригинал структуру на отрезках текста, примерно вдвое превышающих длину, которая учитывалась в соответствующем приближении. Так, второе приближение дает приемлемый текст на уровне двухбуквенных сочетаний, но четырехбуквенные комбинации в нем также являются достаточно схожими с английским языком; фигурирующие в последнем примере комбинации четырех и более слов также вполне могут встречаться в реальных предложениях. Так, фраза из 10 слов «attack on an English writer that the character of this» не является совсем уж бессмысленной. Видно, что достаточно сложный стохастический процесс может дать вполне удовлетворительную модель дискретного источника.

Два первых примера построены с использованием книги случайных чисел и таблицы частот букв. Похожий метод можно было бы использовать и для примеров (3), (4), (5), т. к. частоты диграмм и триграмм известны, однако мы воспользовались иным, более простым подходом. Для построения второго приближения, например, мы открывали книгу случайным образом, выбирали случайную букву и записывали ее; затем мы открывали книгу на другой странице, искали первую двухбуквенную комбинацию, начинающуюся с ранее выбранной буквы, и записывали следующую букву. Потом процедура повторялась. Похожий подход использовался и для остальных примеров. Было бы интересно построить также и следующие приближения, однако это потребовало бы слишком больших затрат времени и сил.

12.4. Представление марковского процесса в виде графа

Стохастические процессы описанного выше типа в математике известны как марковские процессы и достаточно широко освещены в литературе (за подробным изложением отсылаем читателя к книге (Frechet 1938). В общем случае их можно описать следующим образом. Существует конечное число

возможных состояний системы $S_1, S_2, \ldots, S_n$. Кроме того, известен набор вероятностей перехода $p_i(j)$ системы из состояния i в состояние j. Чтобы сделать такой марковский процесс источником информации, достаточно предположить, что он выдает по одной букве в момент каждого перехода. Различные состояния системы в таком случае соответствуют «остаточному влиянию» предшествующих букв.

Эту ситуацию можно представить в виде графов, как показано на рис. 3, 4 и 5. «Состояния» здесь представлены вершинами графа, а вероятности переходов и соответствующие им буквы – соответствующими ребрами. Рисунок 12.3 соответствует примеру (B) из § 12.2, а рис. 12.4 – примеру (C). У системы, изображенной на рис. 12.3, есть лишь одно состояние, т. к. выбираемые буквы независимы. На рис. 12.4 число состояний равно числу букв; в случае учета триграммной структуры их число было бы равно n^2, что соответствует возможным парам букв, предшествующих выбираемой. На рис. 12.5 изображен граф, соответствующий структуре слов из примера (D). Здесь S обозначает символ «пробела». <...>

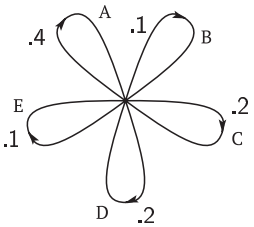


Рис. 12.3. Граф, соответствующий источнику из примера B

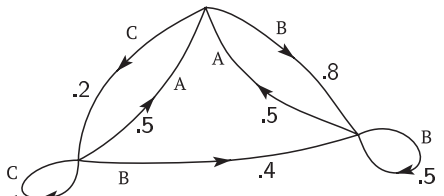


Рис. 12.4. Граф, соответствующий источнику из примера C

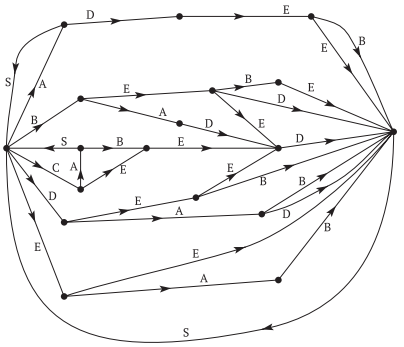


Рис. 12.5. Граф, соответствующий источнику из примера D

12.6. Выбор, неопределенность и энтропия

Мы представили дискретный источник информации марковским процессом. Можно ли определить величину, характеризующую в некотором смысле количество информации, «производимой» таким процессом, или, точнее, темп «производства» информации?

Пусть имеется набор возможных событий с вероятностями p_1, p_2, ..., p_n. Эти вероятности известны, но больше про эти события не известно ничего. Можно ли найти, сколько «произвола» заключено в выборе события, иначе говоря, меру неопределенности исхода?

Разумно потребовать от такой меры, назовем ее $H(p_1, p_2, \ldots, p_n)$, следующих свойств:

1. H должна непрерывно зависеть от p_i.
2. В случае равенства всех p_i, $p_i = 1/n$, H должна быть монотонно возрастающей функцией n. Для равновероятных событий произвол, или неопределенность, возрастает с ростом числа возможных событий.
3. Если выбор можно разбить на два последовательных действия, то исходная величина H должна быть взвешенной суммой отдельных величин H. Смысл этого иллюстрируется на рис. 12.6. Слева мы имеем три возможности $p_1 = 1/2$, $p_2 = 1/3$, $p_3 = 1/6$. Справа мы сначала выбираем между двумя возможностями с вероятностью 1/2 каждая и в случае выбора второй делаем следующий выбор с вероятностями 2/3, 1/3. Конечные результаты имеют те же вероятности, что и раньше. В данном конкретном случае требование сводится к

$$H\left(\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\right) = H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{1}{2} H\left(\frac{2}{3}, \frac{1}{3}\right).$$

Коэффициент 1/2 возникает из-за того, что второй выбор делается лишь в половине случаев.

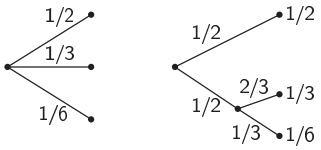


Рис. 12.6. Разложение выбора из трех возможностей

В приложении 2 [Примечание редактора: опущено.] доказывается следующий результат.

Теорема 2. Единственная H, удовлетворяющая трем вышеприведенным условиям, имеет вид:

$$H = -K\sum_{i=1}^{n} p_i \log p_i,$$

где K – положительная постоянная.

Эта теорема, как и предположения, необходимые для ее доказательства, никак не будет использоваться в дальнейшем. Она приведена главным образом для придания обоснованности нашим последующим определениям. Но настоящим обоснованием станут следствия из них.

Величины вида $H = -\sum p_i \log p_i$ (константа K отвечает за выбор единицы измерения) играют центральную роль в теории информации, являясь мерами информации, произвола и неопределенности. Такой же вид имеет выражение для энтропии в некоторых формулировках статистической механики, где p_i – вероятность найти систему в ячейке i ее фазового пространства. Тогда H – не что иное, как H в знаменитой теореме Больцмана. Назовем $H = -\sum p_i \log p_i$ энтропией набора вероятностей $p_1, p_2, \ldots, p_n$. Если x – случайная величина, то будем обозначать ее энтропию $H(x)$; здесь x – не аргумент функции, а метка числа, призванная отличать обозначение энтропии величины x от энтропии $H(y)$ случайной величины y.

Энтропия для случая двух возможностей с вероятностями p и $q = 1 - p$

$$H = -(p \log p + q \log q)$$

изображена на рис. 12.7 в виде функции p.

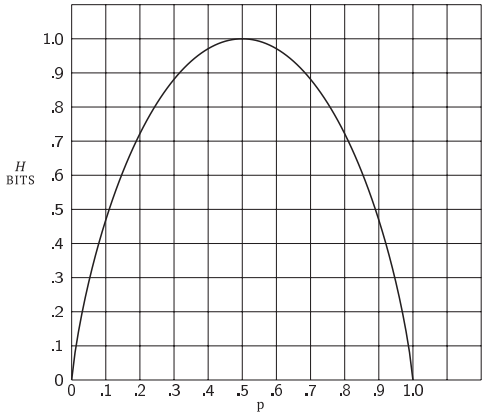


Рис. 12.7. Энтропия в случае двух возможностей с вероятностями p и $1 - p$

Величина H обладает несколькими интересными свойствами, делающими ее приемлемой мерой произвола или информации:

1. $H = 0$ тогда и только тогда, когда все вероятности p_i, за исключением одной, равны нулю, а одна – единице, т. е. величина H обращается в нуль лишь тогда, когда мы уверены в исходе. Во всех остальных случаях H положительна.
2. Для данного n H принимает максимальное значение, равное $\log n$, тогда, когда все p_i равны между собой (т. е. равны $1/n$). Интуитивно понятно, что это – наиболее неопределенная ситуация.
3. Рассмотрим два события, x и y, причем первому соответствует m, а второму – n возможностей. Обозначим $p(i, j)$ вероятность совместного появления i в первом случае и j во втором. Энтропия такого совместного события равна

$$H(x, y) = -\sum_{i,j} p(i, j) \log p(i, j),$$

тогда как

$$H(x) = -\sum_{i,j} p(i, j) \log \sum_{j} p(i, j);$$
$$H(y) = -\sum_{i,j} p(i, j) \log \sum_{j} p(i, j).$$

Легко показать, что $H(x, y) \leqslant H(x) + H(y)$, причем равенство достигается, лишь когда события независимы (т. е. $p(i, j) = p(i)p(j)$). Неопределенность совместного события меньше или равна сумме неопределенностей отдельных событий.
4. Любое изменение, направленное на уравнивание вероятностей $p_1, p_2, \ldots, p_n$, увеличивает H. Так, если $p_1 < p_2$ и мы увеличиваем p_1, уменьшая p_2 на ту же величину, так что p_1 и p_2 оказываются почти равны, то величина H возрастает. В более общем случае, если мы производим некоторую процедуру «усреднения» вида

$$p_i' = \sum_j a_{ij} p_j,$$

где $\sum_i a_{ij} = \sum_j a_{ij} = 1$ и все $a_{ij} \geqslant 0$, то величина H возрастает (за исключением специального случая, когда это преобразование сводится к перестановке; в этом случае H, естественно, остается неизменной).
5. Рассмотрим два случайных события x и y, как в пункте 3, не обязательно независимых. Для любого значения i, которое может принимать x, обозначим $p_i(j)$ условную вероятность того, что y принимает значение j. Она равна

$$p_i(j) = \frac{p(i, j)}{\sum_j p(i, j)}.$$

Определим *условную энтропию* величины y, $H_x(y)$, как среднюю энтропию y при каждом значении x, взвешенную в соответствии с вероятностью получения данного значения x. Она равна

$$H_x(y) = -\sum_{i,j} p(i,j) \log p_i(j).$$

Эта величина показывает, насколько в среднем мы не уверены в y, если знаем x. Подставляя $p_i(j)$, получаем

$$\begin{aligned} H_x(y) &= -\sum_{i,j} p(i,j) \log p(i,j) + \sum_{i,j} p(i,j) \log \sum_j p(i,j) \\ &= H(x,y) - H(x), \end{aligned}$$

или $H(x, y) = H(x) + H_x(y)$. Неопределенность (или энтропия) совместного события x, y равна сумме неопределенности x и неопределенности y при известной величине x.

6. Из 3 и 5 имеем $H(x) + H(y) \geqslant H(x, y) = H(x) + H_x(y)$. Отсюда $H(y) \geqslant H_x(y)$. Неопределенность y никогда не увеличивается при конкретизации x. Она уменьшается, если только x и y не являются независимыми событиями. В последнем же случае она остается неизменной.

12.7. Энтропия источника информации

Рассмотрим дискретный источник информации описанного выше типа. Для каждого возможного состояния i существует набор вероятностей $p_i(j)$ порождения различных возможных символов j. Поэтому каждому состоянию соответствует энтропия H_i. Определим энтропию источника как среднее всех H_i с весами, равными вероятностям соответствующих состояний:

$$\begin{aligned} H &= \sum_i P_i H_i \\ &= -\sum_{i,j} P_i p_i(j) \log p_i(j). \end{aligned}$$

Это энтропия в расчете на один символ текста. Если марковский процесс протекает в определенном темпе, то можно также определить энтропию в секунду $H' = \sum_i f_i H_i$, где f_i – средняя частота (возникновений в секунду) состояния i. Ясно, что $H' = mH$, где m – среднее число символов, порождаемых в секунду. H или H' характеризует количество информации, генерируемой источником в расчете на один символ или в одну секунду. Если логарифм двоичный, то это соответственно количество битов на символ или битов в секунду.

Если последовательные символы независимы, то H есть просто $-\sum p_i \log p_i$, где p_i – вероятность символа i. Пусть имеется достаточно длинное сообщение, содержащее N символов. С высокой вероятностью в нем будет примерно $p_1 N$ вхождений первого символа, $p_2 N$ – второго и т. д. Таким образом, вероятность данного сообщения будет примерно равна

$$p \doteq p_1^{p_1 N} p_2^{p_2 N} \dots p_n^{p_n N},$$

или

$$\log p \doteq N \sum_i p_i \log p_i;$$
$$\log p \doteq -NH;$$
$$H \doteq \frac{\log 1/p}{N}.$$

Следовательно, H – это логарифм обратной вероятности типичной длинной последовательности, поделенный на число символов в ней. Аналогичный результат имеет место для произвольного источника. <...>

13 Коды с обнаружением и исправлением ошибок (1950)

Р. У. Хэмминг

Сегодня мы считаем само собой разумеющимся, что если извлечь число, хранившееся в памяти компьютера, то оно будет точно таким же, как перед помещением туда. Если хранится значение π, то при его использовании программа всякий раз будет видеть 3.14159 и никогда 3.24159 или 3.14158. Аналогично программа, собиравшаяся обратиться к ячейке памяти 2468, никогда не обратится вместо этого к ячейке 2478. Если компьютеры и сбоят, то из-за ошибок программирования, коротких замыканий и неправильных входных данных. Но сами биты всегда правильны, даже если передавались с другой стороны земного шара или с датчика в глубоком космосе и даже если физическая среда, в которой они хранятся и передаются, непрерывная, неидеальная и подчиняется законам статистической физики. Если данные подверглись искажению, то наши компьютеры об этом скажут.

Но так было не всегда.

Поскольку релейные переключатели и шестерни раннего механического счетного оборудования были очень ненадежны, в некоторых устройствах использовались коды с контролем четности, позволявшие обнаружить ошибку в одном бите. Если, к примеру, данные содержали четыре бита, то пятый был бы равен их сумме по модулю 2, т. е. 0, когда количество единиц в четырех битах данных четно, и 1, когда оно нечетно.

Ричард Хэмминг (1915–1998) работал с Джоном фон Нейманом в Манхэттенском проекте, до того как был принят в компанию Bell Telephone Labs на должность математика в 1946 году. Там он делил кабинет с Клодом Шенноном и начал применять опыт вычислений, приобретенный в Лос-Аламосе,

к только нарождающейся науке – теории информации. Биография, сопровождающая присуждение ему премии Тьюринга, сообщает о событии, произошедшем в 1947 году и ставшем причиной открытия совершенно новой области.

> Как-то в пятницу, работая в Bell Laboratories, он настроил свои еще докомпьютерные вычислительные машины на решение сложной задачи и ожидал, что к следующему понедельнику результаты будут готовы. Но, придя на работу в понедельник, обнаружил, что в самом начале вычислений произошла ошибка, и калькуляторы на релейных схемах не смогли продолжить (ACM, 1968).

Случилась ошибка четности, и все вычисление остановилось. Хэмминг, будучи математиком, внимательно проштудировавшим «Законы мышления» Буля, понял, что если компьютер сумел понять, что ошибка произошла, то мог бы понять также и где это случилось и исправить ошибку. Так родилась идея о коде, исправляющем ошибки, который в случае четырехбитового элемента данных требует трех дополнительных битов. (На самом деле в «Первой редакции» отчета о EDVAC на стр. 131 этой книги уже было отмечено, что хотя аппаратные ошибки неизбежны, некоторые из них можно исправить автоматически.)

В этой статье Хэмминг сначала описывает конкретный код с исправлением одиночной ошибки, а затем развивает общую теорию. Он определяет расстояние между двумя битовыми векторами как число позиций, в которых они различаются; ныне эта мера известна как *расстояние Хэмминга*. Два битовых вектора, которые невозможно спутать в результате ошибки в одном бите, находятся друг от друга на расстоянии, не меньшем 2. Поэтому нахождение n-битового кода, типа простого кода с контролем четности, который мог бы обнаруживать ошибки в одном бите, эквивалентно нахождению множества точек в пространстве кодовых слов $\{0, 1\}^n$, в котором расстояние между любыми двумя точками не меньше 2. Если минимальное расстояние между кодовыми словами равно 3, то код способен исправить ошибки в одном бите, поскольку любой битовый вектор может находиться на расстоянии 1 только от одного кодового слова. Таким образом, проектирование кодов сводится к задаче упаковки непересекающихся сфер (геометрического места точек, отстоящих на постоянное расстояние от данной точки) в пространстве $\{0, 1\}^n$.

После публикации статьи Хэмминга эта область начала бурно развиваться. Физический размер битов в современных запоминающих устройствах настолько мал, а их количество так велико, что ошибки на уровне битов неизбежны. Современные компьютеры достигают макроскопической безошибочности благодаря включению тщательно продуманной избыточности на микроскопическом уровне, невидимом пользователю. А используемые для этого методы опираются на идеи, впервые развитые Хэммингом.

Ричард Хэмминг внес много других вкладов в теорию связи, а в 1976 году перешел на работу в академическое учреждение. Он ратовал за улучшенное преподавание математики вплоть до своей смерти в возрасте 82 лет.

13.1. Введение

На исследование, предпринятое в этой работе, автора навело обдумывание больших вычислительных машин, в которых большое количество операций должно быть выполнено без единой ошибки вплоть до получения конечного результата. Эта проблема «правильности» в крупном масштабе не нова; например, на центральном телефонном узле производится очень много операций, но ошибки, ведущие к неправильному соединению, успешно контролируются, хотя полностью избежать их не удается. Это было достигнуто в том числе и путем использования схем с самоконтролем. Редкие ошибки, ускользнувшие от процедур контроля, обнаруживаются пользователем, и если они повторяются, то клиент пишет жалобу. Случайная же ошибка приводит лишь к эпизодическим соединениям с неправильным номером. А весь остальной телефонный узел функционирует нормально. С другой стороны, в цифровом компьютере одиночная ошибка обычно означает полный отказ в том смысле, что если она обнаружена, то последующее вычисление невозможно, пока ошибка не будет локализована и исправлена. Если же ошибка осталась необнаруженной, то все последующие машинные операции дают неверные результаты. Иными словами, в центральном телефонном узле имеется несколько параллельных путей, более или менее независимых друг от друга, тогда как в цифровой машине обычно имеется единственный длинный путь, который проходит по одному и тому же оборудованию много, много раз, прежде чем будет получен ответ.

При передаче информации из одного места в другое цифровые машины используют коды, это просто наборы символов, которым приписан какой-то смысл, или значение. Примеров кодов, спроектированных для обнаружения изолированных ошибок, много; среди них высокоразвитые коды «2 из 5», применяемые в стандартных управляющих коммутаторах и в релейных компьютерах компании Bell Labs (Alt 1948a,b), код «3 из 5», применяемый в радиотелеграфе (Sparks and Kreer 1947, особенно стр. 417), и счетчик слов, передаваемый к конце каждой телеграммы.

В некоторых ситуациях самоконтроля недостаточно. Например, в релейных компьютерах модели 5 компании Bell Telephone Laboratories, построенных для Абердинского испытательного полигона, на раннем этапе эксплуатации наблюдалось два или три отказа реле в день, притом что всего в двух компьютерах было 8900 релейных переключателей. Это означает примерно один отказ на два или три миллиона операций. Наличие средств самоконтроля означало, что эти отказы не приводили к необнаруженным ошибкам, поскольку машины работали в необслуживаемом режиме ночью и в выходные. Однако же ошибки часто становились причиной остановки вычислений, хотя иногда их следствием оказывались новые проблемы. Сейчас развитие цифровых компьютеров движется в сторону электронных скоростей, и надежность базовых элементов в расчете на одну операцию будет выше, чем у релейных переключателей. Однако возможность возникновения изолированных отказов, даже оставшихся необнаруженными, может серьезно помешать нормальной эксплуатации таких машин. Поэтому представляется желательным исследовать следующий за обнаружением шаг – исправление ошибок.

Мы будем предполагать, что передающее оборудование обрабатывает информацию, представленную в двоичной форме, т. е. в виде последовательности нулей и единиц. Это предположение сделано для удобства математических рассуждений и потому, что двоичная система – естественная форма представления замкнутых и разомкнутых реле, триггерных схем, точек и тире, а также перфолент, применяемых во многих видах связи. Таким образом, каждый кодовый символ представляется последовательностью 0 и 1.

Коды, используемые в этой статье, называются *систематическими*. Систематический код можно определить как такой, в котором каждый кодовый символ занимает ровно n двоичных цифр, из которых m цифр несут информацию, а оставшиеся $k = n - m$ цифр служат для обнаружения и исправления ошибок. Возникающая в результате *избыточность* R определяется как отношение общего числа двоичных цифр к минимальному числу, необходимому для передачи той же самой информации, т. е.

$$R = n/m.$$

Эта величина измеряет эффективность кода с точки зрения передачи информации, а только этот аспект проблемы и обсуждается в данной статье. Можно сказать, что избыточность снижает эффективную пропускную способность канала передачи информации.

Поскольку потребность в исправлении ошибок приобрела актуальность лишь недавно, пока еще мало известно об экономической стороне вопроса. Ясно, что для таких кодов понадобится дополнительное оборудование для кодирования и исправления ошибок и что пропускная способность канала неизбежно снизится, о чем было сказано выше. Из-за этого можно ожидать, что поначалу такие коды будут применяться лишь в случае крайней необходимости. Приведем некоторые типичные примеры таких ситуаций:

a) эксплуатация в необслуживаемом режиме на протяжении длительного времени с минимальным количеством резервного оборудования;
b) очень большие и тесно взаимосвязанные системы, в которых одиночный отказ выводит из строя систему в целом;
c) передача сигналов при наличии шума, когда невозможно или экономически неэффективно уменьшать долю шума в сигнале.

Такие ситуации возникают все чаще. Первые две особенно характерны для больших цифровых вычислительных машин, а третья, в числе прочего, возникает в условиях постановки радиопомех.

В этой статье описываются принципы проектирования кодов с обнаружением и исправлением ошибок для случаев, которые, скорее всего, возникнут в первую очередь. Схемы для реализации этих принципов можно конструировать с помощью хорошо известных методов, но эта проблема здесь не обсуждается. В части I показано, как построить специальные коды с минимальной избыточностью в следующих случаях:

a) коды, обнаруживающие одиночные ошибки;
b) коды, исправляющие одиночные ошибки;

c) коды, исправляющие одиночные ошибки и обнаруживающие двойные ошибки.

В части II обсуждается общая теория таких кодов и доказывается, что при сделанных предположениях коды, описанные в части I, «наилучшие» из возможных.

ЧАСТЬ I. КОНКРЕТНЫЕ КОДЫ

13.2. Коды, обнаруживающие одиночные ошибки

Код, обнаруживающий одиночные ошибки, с n двоичными цифрами можно построить следующим образом. В первые $n-1$ позиций помещаем $n-1$ цифры информации. В n-ю позицию помещаем 0 или 1, так чтобы количество единиц во всех n позициях было четным. Очевидно, что этот код обнаруживает одиночную ошибку, поскольку любая такая ошибка при передаче сделала бы число единиц в кодовом символе нечетным.

Поскольку $m = n - 1$, избыточность такого кода равна

$$R = \frac{n}{n-1} = 1 + \frac{1}{n-1}.$$

Может показаться, что для достижения низкой избыточности следует сделать n очень большим. Однако, увеличивая n, мы увеличиваем вероятность, по крайней мере, одной ошибки в символе, а вместе с тем возрастает риск двойной ошибки, которая останется незамеченной. Например, если $p \ll 1$ – вероятность любой ошибки, то при n порядка $1/p$ вероятность получить правильный символ приближенно равна $1/e = 0.3679\ldots$, тогда как вероятность двойной ошибки равна $1/2e = 0.1839\ldots$.

Описанный выше тип контроля, позволяющий определить наличие одиночной ошибки в символе, будет использоваться на протяжении всей статьи и называться *контролем четности*. Тот способ, который описан выше, называется контролем с дополнением до четности, а если бы мы проверяли, что количество единиц нечетно, то получили бы контроль с дополнением до нечетности. Кроме того, контроль четности необязательно должен включать все позиции символа, можно проверять только избранные позиции.

13.3. Коды, исправляющие одиночные ошибки

Чтобы построить код, исправляющий одиночные ошибки, мы сначала постулируем, что m из n имеющихся позиций являются информационными. Будем считать m фиксированным, но конкретные номера позиций определим позже. Оставшиеся k позиций назовем проверочными. Значения в этих k позициях будут определены в процессе кодирования посредством контроля с дополнением до четности выбранных информационных позиций.

n	m	Соответствующее k
1	0	1
2	0	2
3	1	2
4	1	3
5	2	3
6	3	3
7	4	3
8	4	4
9	5	4
10	6	4
11	7	4
12	8	4
13	9	4
14	10	4
15	11	4
16	11	5
и т. д.		

Рис. 13.1

Представим на секунду, что мы получили кодовый символ, с ошибкой или без. k раз выполним контроль четности по порядку, и каждый раз, как результат контроля совпадает со значением, наблюдаемым в его проверочной позиции, будем записывать 0, а каждый раз, как вычисленное и наблюдаемое значения различаются, будем записывать 1. Будучи записана справа налево в ряд, эта последовательность 0 и 1 (не путать со значениями, вычисленными в процессе контроля четности) может рассматриваться как двоичное число. Назовем его проверочным числом. Мы потребуем, чтобы это проверочное число давало позицию любой одиночной ошибки, считая, что нулевое значение говорит об отсутствии ошибок в символе. Таким образом, проверочное число должно описывать $m + k + 1$ различных вещей, поэтому k должно удовлетворять условию $2^k \geqslant m + k + 1$. Поскольку $n = m + k$, находим

$$2^m \leq \frac{2^n}{n+1}.$$

Пользуясь этим неравенством, мы построили рис. 13.1, на котором показано максимальное m для заданного n, или, что то же самое, минимальное n для заданного m.

Теперь определим позиции, к которым нужно применять различные операции контроля четности. Проверочное число строится цифра за цифрой, справа налево, путем применения контроля четности по порядку и выписывания соответствующих нулей или единиц. Поскольку проверочное число должно давать позицию любой ошибки в кодовом символе, любая позиция, в двоичном представлении которой справа находится 1, должна приводить к отказу первой проверки. Рассматривая двоичные представления целых чисел, находим, что

1 = 1
3 = 11
5 = 101
7 = 111
9 = 1001
и т. д.

заканчиваются 1. Следовательно, для первого контроля четности необходимо использовать позиции 1, 3, 5, 7, 9, ….

Точно так же находим, что для второго контроля четности необходимо использовать те позиции, в двоичном представлении которых вторая цифра справа равна 1.

2 = 10
3 = 11
6 = 110
7 = 111
10 = 1010
11 = 1011
и т. д.

Для третьего контроля четности:

4 = 100
5 = 101
6 = 110
7 = 111
12 = 1100
13 = 1101
14 = 1110
15 = 1111
20 = 10100
и т. д.

Остается для каждого контроля четности решить, какие позиции должны содержать информацию, а какие – результаты проверок. Выбор позиций 1, 2, 4, 8, … в качестве проверочных, как показано в таблице ниже, хорош тем, что номера проверочных позиций не зависят друг от друга. Все остальные позиции информационные. Таким образом, получаем рис. 13.2.

В качестве иллюстрации применим развитую выше теорию к случаю семипозиционного кода. По рис. 13.1 для $n = 7$ находим, что $m = 4$ и $k = 3$. По рис. 13.2 находим, что первый контроль четности применяется к позициям 1, 3, 5, 7 и используется для определения значения в первой позиции; второй

контроль четности применяется к позициям 2, 3, 6, 7 и определяет значение во второй позиции; третий контроль четности применяется к позициям 4, 5, 6, 7 и определяет значение в четвертой позиции. Таким образом, позиции 3, 5, 6, 7 являются информационными. Результаты записи всех возможных четырехзначных двоичных чисел в позициях 3, 5, 6, 7 и последующего вычисления значений в проверочных позициях 1, 2, 4 показаны на рис. 13.3.

Проверочное число	Проверочная позиция	Проверенные позиции
1	1	1, 3, 5, 7, 9, 11, 13, 15, 17, ...
2	2	2, 3, 6, 7, 10, 11, 14, 15, 18, ...
3	4	4, 5, 6, 7, 12, 13, 14, 15, 20, ...
4	8	8, 9, 10, 11, 12, 13, 14, 15, 24, ...
.	.	.
.	.	.
.	.	.

Рис. 13.2

Позиция							Десятичное значение символа
1	2	3	4	5	6	7	
0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1
0	1	0	1	0	1	0	2
1	0	0	0	0	1	1	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	1	5
1	1	0	0	1	1	0	6
0	0	0	1	1	1	1	7
1	1	1	0	0	0	0	8
0	0	1	1	0	0	1	9
1	0	1	1	0	1	0	10
0	1	1	0	0	1	1	11
0	1	1	1	1	0	0	12
1	0	1	0	1	0	1	13
0	0	1	0	1	1	0	14
1	1	1	1	1	1	1	15

Рис. 13.3

Таким образом, семипозиционный код с исправлением одиночной ошибки допускает 16 кодовых символов. И $2^7 - 16 = 112$ символов бессмысленны. В некоторых приложениях может быть желательно исключить первый символ из кода, чтобы комбинация, состоящая только из нулей, не могла быть ни кодовым символом, ни кодовым символом с одиночной ошибкой, поскольку ее можно спутать с отсутствием сообщения. Тогда останется 15 полезных кодовых символов.

Для иллюстрации того, как «работает» этот код, возьмем символ 0 1 1 1 1 0 0, соответствующий десятичному значению 12, и заменим 1 в пятой позиции

на 0. Теперь исследуем новый символ 0 1 1 1 0 0 0 описанными в этом разделе методами и посмотрим, как локализуется ошибка. По рис. 13.2 находим, что первый контроль четности применяется к позициям 1, 3, 5, 7 и предсказывает 1 для первой позиции, тогда как в действительности мы находим в ней 0; поэтому записываем 1. Второй контроль четности применяется к позициям 2, 3, 6, 7 и правильно предсказывает вторую позицию; поэтому записываем 0 слева от 1 – получается 0 1. Третий контроль четности применяется к позициям 2, 3, 6, 7 и дает неправильное предсказание; поэтому записываем 1 слева от 0 1 – получается 1 0 1. Эта последовательность нулей и единиц, рассматриваемая как двоичное число, дает число 5; стало быть, ошибка имеет место в пятой позиции. Следовательно, для получения правильного символа нужно заменить 0 в пятой позиции на 1.

13.4. Коды с исправлением одиночных ошибок и обнаружением двойных ошибок

Для построения кода, исправляющего одиночные ошибки и обнаруживающего двойные, начнем с кода, исправляющего одиночные ошибки. К нему мы добавим еще одну позицию для проверки всех предыдущих позиций с помощью контроля с дополнением до четности. Чтобы понять, как работает этот код, нужно рассмотреть несколько случаев.

1. Ошибок нет. Все контроли четности, включая последний, проходят.
2. Одиночная ошибка. Последний контроль четности во всех таких ситуациях не проходит вне зависимости от того, произошла ошибка в информационных позициях, первоначальных проверочных позициях или в последней проверочной позиции. Первоначальное проверочное число дает позицию ошибки, но теперь нулевое значение означает последнюю проверочную позицию.
3. Две ошибки. Во всех таких ситуациях последний контроль четности проходит, а проверочное число указывает на наличие какой-то ошибки.

Для иллюстрации построим восьмипозиционный код на основе ранее рассмотренного семипозиционного. Для этого добавим восьмую позицию, выбрав ее так, чтобы во всех восьми позициях было четное число единиц. Таким образом, мы добавляем восьмой столбец к рис. 13.3 (см. рис. 13.4).

0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0	1
1	1	1	1	1	1	1	1	1
1	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	0	1	0
0	0	1	1	0	0	1	1	0
1	0	1	0	1	0	1	0	1
1	1	0	0	1	1	0	0	1
1	1	1	1	1	1	1	1	1
0	1	1	0	0	1	1	0	0
0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	1	1

Рис. 13.4

ЧАСТЬ II. ОБЩАЯ ТЕОРИЯ

13.5. Геометрическая модель

При изучении различных проблем, связанных с кодами, обнаруживающими и исправляющими ошибки, часто бывает удобно ввести геометрическую модель. Используемая здесь модель заключается в интерпретации различных последовательностей 0 и 1, являющихся символами кода, как вершин единичного n-мерного куба. Кодовые точки, обозначаемые x, y, z, ..., образуют подмножество множества всех вершин куба. В этом пространстве, состоящем из 2^n точек, мы введем *расстояние*, или, как его обычно называют, *метрику*, $D(x, y)$. Определение метрики основано на том наблюдении, что одиночная ошибка в кодовой точке изменяет одну координату, двойная ошибка – две координаты, и вообще ошибки в d позициях приводят к различию в d координатах. Поэтому определим расстояние $D(x, y)$ между точками x и y как количество координат, в которых x и y различаются. Это то же самое, что наименьшее количество ребер, по которым нужно пройти на пути от x к y. Эта функция расстояния удовлетворяет трем стандартным свойствам метрики, а именно:

$$D(x, y) = 0 \text{ тогда и только тогда, когда } x = y;$$

$$D(x, y) = D(y, x) > 0, \text{ если } x \neq y;$$

$$D(x, y) + D(y, z) \geqslant D(x, z) \text{ (неравенство треугольника).}$$

В качестве примера заметим, что каждая из следующих кодовых точек в трехмерном кубе отстоит на две единицы от остальных: 0 0 1; 0 1 0; 1 0 0; 1 1 1. Продолжая использовать геометрический язык, определим сферу радиуса r с центром в точке x как множество всех точек, отстоящих от x на расстояние r. Так, в примере выше первые три кодовые точки лежат на сфере радиуса 2 с центром в точке (1, 1, 1). На самом деле в этом примере любую точку можно назначить центром, тогда три остальные будут лежать на сфере радиуса 2.

Если все кодовые точки находятся на расстоянии, не меньшем 2, друг от друга, то любая одиночная ошибка переместит кодовую точку в точку, не являющуюся кодовой, а значит, соответствующую бессмысленному символу. Это, в свою очередь, означает, что любую одиночную ошибку можно обнаружить. Если минимальное расстояние между двумя кодовыми точками не меньше 3 единиц, то любая одиночная ошибка оставит точку ближе к правильной кодовой точке, чем к любой другой кодовой точке, а это значит, что любую одиночную ошибку можно исправить. Такого рода информация сведена в таблицу на рис. 13.5.

Минимальное расстояние	Предназначение
1	уникальность
2	обнаружение одиночных ошибок
3	исправление одиночных ошибок
4	исправление одиночных ошибок плюс обнаружение двойных ошибок
5	исправление двойных ошибок
	и т. д.

Рис. 13.5

Обратно, очевидно, что если мы хотим гарантировать вышеперечисленные обнаружение и исправление, то все расстояния между кодовыми точками должны быть больше или равны соответствующему минимальному расстоянию. Таким образом, проблема нахождения подходящих кодов та же самая, что проблема нахождения в пространстве подмножеств точек, удовлетворяющих условию на минимальное расстояние. Конкретные коды, описанные в § 13.2, 13.3 и 13.4, – просто способы выбрать нужное подмножество точек для минимальных расстояний 2, 3 и 4 соответственно.

Быть может, стоит отметить, что при заданном минимальном расстоянии частью возможностей исправления можно пожертвовать ради лучшего обнаружения. Например, подмножество с минимальным расстоянием 5 можно использовать для:

a) исправления двойных ошибок (и, разумеется, обнаружения двойных ошибок);
b) исправления одиночных ошибок и обнаружения тройных ошибок;
c) обнаружения четверных ошибок.

Возвращаясь ненадолго к конкретным кодам, построенным в части I, заметим, что перестановки позиций в коде не приводят к существенным изменениям самого кода. Равно как и замена нулей на единицы и единиц на нули в любой позиции, этот процесс обычно называют дополнением. Эта идея более точно выражена в следующем определении.

Определение. Говорят, что два кода *эквивалентны*, если один можно преобразовать в другой, выполнив конечное число следующих операций:

1) перестановка любых двух позиций в кодовых символах;
2) дополнение значения в любой позиции кодовых символов.

Это определение удовлетворяет всем формальным свойствам отношения эквивалентности ($\sim$), т. к. $A \sim A$, из $A \sim B$ следует $B \sim A$ и из $A \sim B, B \sim C$ следует $A \sim C$. Таким образом, мы можем свести изучение класса кодов к изучению типичных представителей каждого класса эквивалентности. В терминах геометрической модели эквивалентные преобразования сводятся к поворотам и отражениям единичного куба.

13.6. Коды, обнаруживающие одиночные ошибки

В этом разделе изучается задача об упаковке максимального числа точек в единичный n-мерный куб таким образом, чтобы расстояние между любыми двумя точками было не меньше 2. Мы покажем, что, как и в § 13.2, так упаковать 2^{n-1} точек можно и, более того, что любая такая оптимальная упаковка эквивалентна описанной в § 13.2.

Для доказательства этих утверждений сначала заметим, что множество вершин n-мерного куба является объединением множеств вершин двух $(n-1)$-мерных кубов. Пусть A – максимальное число точек, упакованных в исходный куб. Тогда в одном из двух $(n\text{-}1)$-мерных кубов имеется не менее $A/2$ точек. Этот куб также разлагается на два куба меньшей размерности, и, следовательно, в одном из них находится не менее $A/2^2$ точек. Продолжая таким образом, приходим к двумерному кубу, содержащему $A/2^{n-2}$ точек. Теперь заметим, что в квадрате может быть не более двух точек, отстоящих друг от друга на расстояние, не меньшее двух единиц; следовательно, в исходном n-мерном кубе было не более 2^{n-1} точек, отстоящих друг от друга на расстояние, не меньшее двух единиц.

Чтобы доказать эквивалентность любых двух оптимальных упаковок, заметим, что если упаковка оптимальна, то в любом из двух подкубов находится половина точек. Назвав это первой координатой, мы видим, что у половины точек она равна 0, а у половины 1. При следующем разделении мы снова разбиваем эти множества на две равные группы, в одной из которых находятся нули, а в другой – единицы. После $(n-1)$ таких шагов мы будем иметь, после переупорядочения назначенных значений, если это окажется необходимым, в точности первые $n-1$ позиций кода, предложенного в § 13.2. Для каждой последовательности первых $n-1$ координат существует $n-1$ других последовательностей, отличающихся от нее в одной координате. После того как мы зафиксируем n-ю координату какой-то одной точки, скажем начала координат со всеми нулевыми координатами, n-е координаты всех остальных кодовых точек однозначно определяются условием, согласно которому минимальное расстояние между кодовыми точками равно двум единицам. Таким образом, последняя координата определяется с точностью до дополнения, так что любой оптимальный код эквивалентен коду, описанному в § 13.2.

Интересно отметить, что в этих двух доказательствах мы пользовались только предположением о том, что длины всех кодовых символов равны n.

13.7. Коды, исправляющие одиночные ошибки

Читатель, вероятно, обратил внимание, что в конкретных кодах из части I проводилось различие между информационными и контрольными позициями, тогда как в геометрической модели никакого различия между координатами нет. Чтобы сблизить оба подхода, переопределим систематический код как такой код, в котором длины всех символов равны и

1) проверяемые позиции не зависят от информации, содержащейся в символе;
2) проверки не зависят друг от друга;
3) используется контроль четности.

Это определение эквивалентно предыдущему. Чтобы убедиться в этом, построим матрицу, в i-й строке которой единицы находятся в позициях i-го контроля четности и нули в остальных позициях. По условию 1, матрица фиксирована и не изменяется при переходе от одного кодового символа к другому. По условию 2, ранг матрицы равен k. Это, в свою очередь, означает, что система уравнений позволяет выразить k позиций через остальные $n - k$. Условие 3 означает, что при решении системы используется арифметика, в которой $1 + 1 = 0$.

Существуют несистематические коды, но до сих пор не было найдено ни одного, который для заданного n и минимального расстояния d давал бы больше кодовых символов, чем систематический. <...>

Возвращаясь к главной проблеме этого раздела, мы по рис. 13.5 видим, что в коде, исправляющем одиночные ошибки, кодовые точки расположены на расстоянии не меньше трех единиц друг от друга. Таким образом, каждую точку можно окружить сферой радиуса 1, так что никакие две сферы не будут иметь общих точек. На поверхности каждой сферы расположено n точек и еще одна в ее центре – итого $n + 1$ точек. Таким образом, в пространстве, состоящем из 2^n точек, может находиться не более

$$\frac{2^n}{n+1}$$

сфер. Именно эту границу мы нашли в § 13.3.

Хотя мы показали, что конкретный код, исправляющий одиночные ошибки, который был построен в § 13.3, имеет минимальную избыточность, невозможно доказать, что все оптимальные коды эквивалентны, потому что следующий тривиальный пример демонстрирует, что это не так. Для $n = 4$ находим по рис. 13.1, что $m = 1$ и $k = 3$. Следовательно, в четырехпозиционном коде имеется не более двух кодовых символов. Следующие два оптимальных кода, очевидно, не эквивалентны:

$$\begin{matrix} 0000 \\ 1111 \end{matrix} \quad \text{и} \quad \begin{matrix} 0000 \\ 0111 \end{matrix}$$

13.8. Коды с исправлением одиночных ошибок и обнаружением двойных ошибок

В этом разделе мы докажем, что коды, построенные в § 13.4, имеют минимальную избыточность. Мы уже показали в § 13.4, как для $(n-1)$-мерного кода с минимальным расстоянием 3, имеющего минимальную избыточность, можно построить n-мерный код с таким же числом символов, но с минималь-

ным расстоянием 4. Если бы избыточность этого кода была не минимальна, то существовал бы код с таким же n и с таким же минимальным расстоянием между символами, но содержащий больше кодовых символов. Возьмем этот код и исключим последнюю координату. Тогда размерность уменьшится с n до $n - 1$, а минимальное расстояние между кодовыми символами уменьшится не более чем на одну единицу, притом что число кодовых символов останется тем же самым. Это противоречит предположению о том, что код, с которого мы начали построение, имеет минимальную избыточность. Таким образом, коды из § 13.4 имеют минимальную избыточность.

Это частный случай следующей общей теоремы: каждому коду с минимальной избыточностью, содержащему N точек в $(n-1)$-мерном пространстве, с минимальным расстоянием между точками $2k - 1$ соответствует код с минимальной избыточностью, содержащий N точек в n-мерном пространстве с минимальным расстоянием $2k$, и наоборот. Чтобы построить n-мерный код по $(n-1)$-мерному, мы просто добавляем n-ю координату, которая фиксируется условием контроля с дополнением до четности, примененным к n позициям. Это также увеличивает минимальное расстояние на 1 по следующей причине: любые две точки, находившиеся в $(n - 1)$-мерном коде на расстоянии $2k - 1$ друг от друга, имели нечетное число различающихся соответственных координат. Таким образом, контроль четности давал противоположные результаты для двух точек, увеличивая расстояние между ними до $2k$. Дополнительная координата не могла уменьшить расстояния, поэтому все кодовые точки теперь находятся на минимальном расстоянии $2k$. Чтобы доказать обратное утверждение, мы просто исключаем одну координату из n-мерного кода. Это уменьшает минимальное расстояние с $2k$ до $2k - 1$, оставляя N тем же самым. Ясно, что если один код имеет минимальную избыточность, то это справедливо и для другого. <...>

14 Вычислительные машины и разум

Алан Мэтисон Тьюринг

Алан Тьюринг начал периодически сотрудничать с британской разведкой в области вскрытия шифров вскоре после публикации статьи «О вычислимых числах» (о жизни Тьюринга см. главу 6). После того как Великобритания объявила войну Германии, он перешел на постоянную работу в Блетчли-Парк, центр британской секретной службы во время войны. Там он работал над вскрытием шифровальной машины Энигма, применявшейся для шифрования связи между немецким командованием, войсками и судами. Эта работа завершилась полным успехом, но как и все, что делалось в Блетчли-Парк, много лет оставалась засекреченной и неопубликованной. Ныне общее мнение склоняется к тому, что она сыграла важную роль и, вероятно, существенно приблизила конец войны. Британия отметила заслуги Тьюринга в 1946 году.

Основываясь на приобретенном за время войны опыте и примерно в то же время, когда группа из Электротехнической школы Мура занималась конструированием EDVAC, Тьюринг в Кембридже приступил к проектированию «автоматической вычислительной машины», сокращенно ACE (Automatic Calculating Engine). Хотя в конечном итоге она стала обычным компьютером с хранимой программой, весь процесс ее конструирования был предельно забюрократизирован из-за пересечений с секретными разработками, поэтому Тьюринг покинул проект. Он перешел в Манчестерский университет, где активно занимался проектированием компьютера Mark 1 (обсуждается далее в главе 15, это не тот Mark I, который создавался Эйкеном в Гарварде). В Манчестере он задумался над тем, что смогут делать компьютеры в будущем. Написанная в результате замечательная статья ввела в обиход термин «тест Тьюринга», хотя сам Тьюринг во введении ясно указал, что его «игра в имитацию» не является тестом, проверяющим способность машины мыслить (саму эту постановку вопроса Тьюринг называет «абсурдной»), и заменяет

его другим вопросом, который можно подвергнуть научному исследованию. Программа Вейценбаума Элиза (глава 27) так легко обманывала людей, что уместность теста Тьюринга была поставлена под сомнение; философ Джон Сирл (Searle 1980) выдвинул против него более тонкие возражения. Приведенная в статье аргументация вызывает споры и сегодня (подробный обзор см. в работе Shieber [2004]). Мы опускаем объяснения цифровых компьютеров и универсальности модели, данные Тьюрингом.

Тьюринг запомнился не только своими математическими работами, но и широтой интересов. Журнал «Mind», в котором первоначально была опубликована эта статья, – солидное философское издание; из опубликованного здесь текста выпущен только раздел, в котором описывается универсальность математического автомата. В последние годы жизни Тьюринг обратился к математической биологии, которая долго очаровывала его.

14.1. Игра в имитацию

У меня есть намерение рассмотреть вопрос, может ли машина мыслить. Наше рассмотрение должно начаться с выяснения значения терминов «машина» и «мыслить». Мы с легкостью можем сформулировать определения, отражающие повседневное значение этих слов, однако подобный путь таит в себе опасности. Если определять значение слов «машина» и «мыслить» в соответствии с употреблением этих слов в обиходе, трудно будет избежать вывода о том, что ответ на вопрос «Может ли машина мыслить?» следует искать в данных статистических исследований, наподобие опросов Гэллапа. Несомненно, это было бы абсурдно. Вместо того чтобы идти по обозначенному пути, я заменю выдвинутый выше вопрос другим, имеющим непосредственное отношение к первому и сформулированным в словах, которые могут быть истолкованы относительно однозначно.

Нашу новую задачу представляется возможным описать в терминах игры, которую мы назовем игрой в имитацию. В нее играют трое: мужчина (А), женщина (В) и исследователь, задающий вопросы (С), причем последний может быть любого пола. Исследователь отделен стенами комнаты от двух других игроков. Цель игры для исследователя заключается в том, чтобы определить, кто из игроков – мужчина, а кто – женщина. Он знает их под обозначениями X и Y и в конце игры выносит суждение следующего вида: «X – это А, а Y – это В» или «X – это В, а Y – это А». Ему разрешается задавать игрокам А и В вопросы следующего вида: «С. Попрошу X сказать, короткие или длинные у него или у нее волосы».

Предположим, что X – на самом деле А, значит, отвечать на этот вопрос должен А. Цель игрока А в игре – побудить С к неверным умозаключениям. Поэтому в данном случае он может ответить: «А. Мои волосы коротко острижены, а самые длинные пряди достигают девяти дюймов в длину».

Чтобы исследователь не мог догадаться по голосу, отвечает ему мужчина или женщина, все ответы должны подаваться в письменной форме, лучше

всего отпечатанными на пишущей машинке. Идеальным вариантом будет передача ответов посредством телетайпа из комнаты в комнату. Или же вопросы и ответы на них могут передаваться участниками игры через посредника. Цель игры для третьего игрока (B) состоит в помощи исследователю. Наилучшей стратегией для этого игрока видятся правдивые ответы. Она (если допустить, что B – это Y) может вставлять в свои ответы фразы наподобие «Это я женщина, а его вы не слушайте!», однако этим она ничего не добьется, поскольку мужчина тоже может давать подобные комментарии.

Теперь зададимся вопросом: что произойдет, если место живого игрока A в этой игре займет машина? Будет ли исследователь ошибаться в своих выводах столь же часто, как тогда, когда в игре участвовали реальные мужчина и женщина? Все эти вопросы заменят наш первоначальный вопрос: «Может ли машина мыслить?»

14.2. Критика новой постановки задачи

Помимо выяснения того, каков может быть ответ на этот переформулированный вопрос, мы можем поинтересоваться, заслуживает ли данный вопрос какого бы то ни было рассмотрения. Это последнее обстоятельство мы и намерены изучить, тем самым разрывая бесконечную логическую цепочку причин и следствий.

Новая формулировка позволяет провести достаточно четкую разграничивающую линию между физическими и интеллектуальными способностями человека. Ни один инженер и ни один химик не притязают на изобретение материала, неотличимого по тактильным ощущениям от человеческой кожи. Вполне возможно, что рано или поздно такой материал появится, но, даже допуская, что такое изобретение состоялось, вряд ли стоит, думается, дополнительно очеловечивать «мыслящую машину» посредством ее заключения в искусственную плоть. Постановка нашего вопроса в новой форме отражает это условие, напрямую запрещая исследователю касаться других участников игры, видеть их или вслушиваться в их голоса. Иные преимущества предлагаемого теста можно продемонстрировать на образцах вопросов и ответов. Итак:

Вопрос: Пожалуйста, напишите сонет о мосте Форт-бридж.
Ответ A.: Я не смогу этого сделать. У меня нет поэтического таланта.
Вопрос: Прибавьте 34 957 к 70 764.
Ответ A. (после 30-секундной паузы): 105 621.
Вопрос: Вы играете в шахматы?
Ответ A.: Да.
Вопрос: Мой король стоит на e8, других фигур у меня нет. У вас король на e6 и ладья на h1. Ваш ход. Как вы поступите?
Ответ A. (после 15-секундной паузы): Ладья на h8, мат.

Метод вопросов и ответов кажется подходящим для оценки почти всякой области человеческой жизнедеятельности, какую мы пожелаем включить в рассмотрение. Мы не стремимся ставить машине в вину невозможность

блистать на конкурсах красоты или винить человека в том, что он проигрывает состязание в скорости с самолетом. По условиям нашей игры эти очевидные неравенства признаются несущественными. Участникам игры, т. е. дающим ответы в игре, позволено хвастаться, если они полагают хвастовство разумным, сколько угодно рассуждать о своей неотразимости, силе и героизме, но исследователь не вправе требовать наглядной демонстрации этих качеств.

Пожалуй, предложенную игру можно критиковать за то, что преимущества в ней находятся на стороне машины. Если человек попытается выдать себя за машину, не подлежит сомнению, что эта попытка окажется неудачной. Все мгновенно станет понятно по замедленности расчетов и по ошибкам в арифметических вычислениях. Кроме того, машины, возможно, обладают чем-то, что можно описать как мышление, но это мышление принципиально отличается от человеческого. Данное возражение видится весьма убедительным, но мы можем постулировать, что если будет когда-либо сконструирована машина для удовлетворительной игры в имитацию, указанное возражение нет необходимости учитывать.

Следует подчеркнуть, что в игре в имитацию наилучшей стратегией для машины будет, наверное, не прямое подражание поведению человека, а нечто иное. Подобное не исключено, однако, на мой взгляд, маловероятно, чтобы это имело серьезное значение. В любом случае, у нас нет намерения углубляться в теорию игр, поэтому можно допустить, что наилучшей стратегией для машины будет предлагать исследователю ответы того типа, какие в данных обстоятельствах давал бы человек.

14.3. Машины, привлекаемые к игре

Вопрос, поставленный в § 14.1, не станет совершенно точным до тех пор, пока мы не укажем, что именно следует понимать под словом «машина». Разумеется, нам бы хотелось, чтобы в игре можно было применять любой вид инженерной техники. Мы склонны также допустить возможность, что инженер или группа инженеров могут построить машину, которая будет работать, но не смогут дать удовлетворительного описания ее работы, поскольку метод, которым они пользовались, опирался в основном на эксперимент. Наконец, мы хотели бы исключить из категории машин людей, рожденных обычным образом. Трудно построить определение так, чтобы оно удовлетворяло этим трем условиям. Можно, например, потребовать, чтобы все конструкторы машины были одного пола, но в действительности этого недостаточно, так как, по-видимому, можно вырастить законченный индивидуум из одной-единственной клетки, взятой (например) из кожи человека. Это был бы подвиг биологической техники, заслуживающий самой высокой похвалы, но мы не склонны рассматривать этот случай как «построение мыслящей машины». Сказанное наводит нас на мысль отказаться от требования, согласно которому в игре следует допускать любой вид техники. Мы еще больше склоняемся к этой мысли в силу того обстоятельства, что наш интерес к «мыслящим

машинам» возник благодаря машине особого рода, обычно называемой «электронной вычислительной машиной», или «цифровой вычислительной машиной». Поэтому мы разрешаем принимать участие в нашей игре только цифровым вычислительным машинам.

На первый взгляд это ограничение кажется весьма сильным. Я постараюсь показать, что в действительности дело обстоит не так. Для этого мне придется дать краткий обзор природы и свойств этих вычислительных машин. Можно также сказать, что отождествление машин с цифровыми вычислительными машинами – равно как и наш тест на «мышление» – должно быть признано совершенно неудовлетворительным, если (вопреки моему убеждению) окажется, что цифровые вычислительные машины не в состоянии хорошо играть в имитацию.

Целый ряд вычислительных машин уже эксплуатируется, и естественно возникает вопрос: «А почему бы нам, вместо того чтобы сомневаться в правильности наших рассуждений, не поставить эксперимент? Удовлетворить условиям было бы нетрудно, в качестве задающих вопросы можно было бы использовать много различных людей, и полученные статистические данные показали бы, как часто задающим вопросы удавалось прийти к правильному заключению». Коротко на этот вопрос можно ответить так: нас интересует не то, будут ли все цифровые вычислительные машины хорошо играть в имитацию, и не то, будут ли хорошо играть в эту игру те вычислительные машины, которыми мы располагаем в настоящее время; вопрос заключается в том, существуют ли воображаемые вычислительные машины, которые могли бы играть хорошо. Но это только краткий ответ. Ниже мы рассмотрим этот вопрос в несколько ином свете.

14.4. Цифровые вычислительные машины

То, что мы имеем в виду, говоря о цифровых вычислительных машинах, можно пояснить следующим образом. Предполагается, что эти машины могут выполнять любую операцию, которую мог бы выполнить человек-вычислитель. Мы считаем, что вычислитель придерживается определенных, раз навсегда заданных правил и не имеет права ни в чем отступать от них. Мы можем также считать, что эти правила собраны в книге, которая заменяется другой, когда вычислитель приступает к новой работе. У человека-вычислителя имеется также неограниченный запас бумаги, на которой он производит вычисления. Кроме того, он может выполнять операции сложения и умножения с помощью арифмометра – это несущественно. <…>

Сама идея цифровой вычислительной машины отнюдь не является новой. *Чарлз Бэббидж*, лукасовский профессор математики в Кембридже, занимавший это место с 1828 по 1839 г., разработал проект вычислительного устройства, названного им «Аналитической машиной»; создание ее, однако, так и не удалось завершить. Хотя у Бэббиджа были все основные идеи, существенные для создания такого механизма, его машина не имела перспектив. Скорость вычислений, которую позволила бы достичь машина Бэббиджа,

оказалась бы, разумеется, выше скорости, достигаемой человеком, однако она была бы почти в 100 раз меньше, чем у той вычислительной машины, которая в настоящее время работает в Манчестере и которая является одной из самых медленных современных машин. Запоминающее устройство в машине Бэббиджа было задумано как чисто механическое, с использованием карт и зубчатых колес.

То, что Аналитическая машина Бэббиджа была задумана как чисто механический аппарат, помогает нам избавиться от одного предрассудка. Часто придают значение тому обстоятельству, что современные цифровые машины являются электрическими устройствами и что нервную систему человека в некотором смысле можно отождествить с электрическим устройством. Но, поскольку машина Бэббиджа не была электрическим аппаратом и поскольку в известном смысле все цифровые вычислительные машины эквивалентны, становится ясно, что использование электричества в этом случае не может иметь теоретического значения. Естественно, что там, где требуется быстрая передача сигналов, обычно появляется электричество, поэтому неудивительно, что мы встречаем его в обоих указанных случаях. Для нервной системы химические явления играют по крайней мере столь же важную роль, что и электрические. В некоторых же вычислительных машинах запоминающее устройство в основном акустическое. Отсюда ясно, что сходство между нервной системой и цифровыми вычислительными машинами, состоящее в том, что в обоих случаях используется электричество, весьма поверхностное. Если мы действительно хотим найти такие сходные черты, то, скорее, следует обратиться к математическим аналогиям функции.

14.5. Универсальность цифровых вычислительных машин

Рассмотренные в предыдущем разделе цифровые вычислительные машины можно отнести к классу «машин с дискретными состояниями». Так называются машины, которые сменяют одно четко определенное состояние на другое в результате мгновенного перехода или переключения. Состояния, о которых идет речь, достаточно отличаются друг от друга, поэтому можно пренебречь возможностью принять по ошибке одно из них за другое. Строго говоря, таких машин не существует. В действительности всякое движение непрерывно. Однако имеется много видов машин, которые удобно *считать* машинами с дискретными состояниями. Например, если рассматривать выключатели осветительной сети, то удобно считать, отвлекаясь от действительного положения дел, что каждый выключатель может быть либо включен, либо выключен. То, что выключатель фактически имеет также и промежуточные состояния, несущественно для наших целей, и мы можем об этом забыть. <…> [Примечание редактора: рассуждение об универсальности опущено.]

Именно это особое свойство цифровых вычислительных машин – то, что они могут имитировать любую машину с дискретными состояниями, – и име-

ют в виду, когда говорят, что цифровые вычислительные машины являются универсальными машинами. Из того, что имеются машины, обладающие свойством универсальности, вытекает важное следствие: чтобы выполнять различные вычислительные процедуры, нам вовсе не нужно создавать все новые и новые разнообразные машины (если отвлечься от растущих требований к быстроте вычислений). Все вычисления могут быть выполнены с помощью одной-единственной цифровой вычислительной машины, если снабжать ее надлежащей программой для каждого случая. В дальнейшем мы увидим в качестве следствия из этого результата, что все цифровые вычислительные машины в каком-то смысле эквивалентны друг другу.

Теперь мы можем вернуться к вопросу, поднятому нами в конце § 14.3. Там мы высказали предположение, что вопрос «могут ли машины мыслить?» можно заменить вопросом «существуют ли воображаемые цифровые вычислительные машины, которые могли бы хорошо играть в имитацию?». Если угодно, мы можем придать этому вопросу видимость большей общности и спросить: «Существуют ли машины с дискретными состояниями, которые могли бы хорошо играть в эту игру?» Но в свете того, что цифровые вычислительные машины универсальны, мы видим, что любой из таких вопросов эквивалентен следующему: «Если взять только одну конкретную цифровую вычислительную машину С, то спрашивается: справедливо ли утверждение о том, что, изменяя емкость памяти этой машины, увеличивая ее быстродействие и снабжая ее подходящей программой, можно заставить С удовлетворительно исполнять роль А в «игре в имитацию» (причем роль В будет исполнять человек).

14.6. Противоположные точки зрения по основному вопросу

Теперь мы можем считать, что основные понятия нами выяснены, и перейти к рассмотрению вопроса «могут ли машины мыслить?» и его варианта, изложенного в конце предыдущего раздела. Вместе с тем мы не можем отказаться от первоначальной формы вопроса, так как по поводу равноценности замены одной формы вопроса другой мнения могут расходиться и в любом случае необходимо выслушать то, что было бы сказано в этой связи. Читателю будет легче разобраться в этой дискуссии, если я сначала разъясню свои собственные убеждения. Рассмотрим сперва более точную форму вопроса. Я уверен, что через пятьдесят лет станет возможным программировать работу машин с емкостью памяти порядка 10^9, чтобы они могли играть в имитацию настолько успешно, что шансы среднего человека установить присутствие машины через пять минут после того, как он начнет задавать вопросы, не поднимались бы выше 70 %. Первоначальный вопрос «могут ли машины мыслить?» я считаю слишком абсурдным, чтобы тратить на него время. Тем не менее я убежден, что к концу нашего века употребление слов и мнения, разделяемые большинством образованных людей, изменятся настолько, что

можно будет говорить о мыслящих машинах, не боясь, что тебя поймут неправильно. Более того, я считаю вредным скрывать такие убеждения. Широко распространенное представление о том, что ученые с неуклонной последовательностью переходят от одного вполне установленного факта к другому, не менее хорошо установленному факту, не давая увлечь себя никакому непроверенному предположению, в корне ошибочно. Не будет никакого ущерба от того, что мы ясно осознаем, что является доказанным фактом, а что – предположением. Догадки очень важны, ибо они подсказывают направления, полезные для исследований.

Теперь перейду к рассмотрению мнений, противоположных моему собственному.

14.6.1. Теологическое возражение. Мышление есть свойство бессмертной души человека, Бог дал бессмертную душу каждому мужчине и каждой женщине, но не дал души никакому другому животному и машинам. Следовательно, ни животное, ни машина не могут мыслить.

Я не могу согласиться ни с чем из только что сказанного и попробую возразить, пользуясь теологическими же терминами. Я счел бы данное возражение более убедительным, если бы животные были отнесены в один класс с людьми, ибо, на мой взгляд, между типичным одушевленным и типичным неодушевленным предметами различие больше, чем между человеком и другими животными. Произвольный характер этой ортодоксальной точки зрения станет еще яснее, если мы рассмотрим, в каком свете она может представиться человеку, исповедующему какую-нибудь другую религию. Как, например, христиане отнесутся к точке зрения мусульман, считающих, что у женщин нет души? [Примечание редактора: в работе Smith and Haddad (1975, сноска 2) рассмотрен источник этого заблуждения относительно ислама.] Но оставим этот вопрос и обратимся к основному возражению. Мне кажется, что из приведенного выше аргумента со ссылкой на душу у человека следует серьезное ограничение всесильности Всемогущего. Пусть даже существуют определенные вещи, которые Бог не может выполнить, – например, сделать так, чтобы единица оказалась равной двум; но кто же из верующих не согласился бы с тем, что Он волен вселить душу в слона, если сочтет это необходимым? Мы можем искать выход в предположении, что Он пользуется своей силой лишь в сочетании с мутациями, совершенствующими мозг настолько, что последний оказывается в состоянии удовлетворить требованиям души, которую Он желает вселить в слона. Но точно так же можно рассуждать и в случае машин. Это рассуждение может показаться отличающимся лишь потому, что в случае машин его труднее «переварить». По сути дела, это означает, что мы считаем весьма маловероятным, чтобы Бог счел обстоятельства подходящими для того, чтобы дать душу машине, т. е. речь идет в действительности о других аргументах, которые обсуждаются в остальной части статьи. Пытаясь построить мыслящие машины, мы поступаем по отношению к Богу не более непочтительно, посягая на Его способность создавать души, чем когда производим потомство; в обоих случаях мы являемся лишь орудиями его воли и производим лишь убежища для душ, которые творит опять-таки Бог.

Все это, однако, досужие рассуждения. В пользу чего бы ни приводили такого рода теологические доводы, они не производят на меня особого впечатления. Однако в старину такие аргументы находили весьма убедительными. Во времена Галилея полагали, что такие церковные тексты, как «Стояло солнце среди неба и не спешило к западу почти целый день» (Иисус Навин, 10, 3) и «Ты поставил землю на твердых основах; не поколеблется она во веки и веки» (псалом 103, 5), в достаточной мере опровергали теорию Коперника. В наше время такого рода доказательство представляется беспочвенным. Но когда современный уровень знаний еще не был достигнут, подобные доводы производили совсем другое впечатление.

14.6.2. Возражение со «страусиной» точки зрения. «Последствия машинного мышления были бы слишком ужасны. Будем надеяться и верить, что машины не могут мыслить».

Это возражение редко выражают в столь откровенной форме. Но оно звучит убедительно для большинства из тех, кому вообще приходит в голову. Мы склонны верить, что человек в интеллектуальном отношении стоит выше всей остальной природы. Лучше всего, если бы удалось доказать, что человек по необходимости является самым совершенным существом, ибо в таком случае он может не бояться потерять свое главенствующее положение. Ясно, что популярность теологического возражения связана именно с этим чувством. Это чувство, вероятно, особенно сильно у людей интеллигентных, так как они ценят силу мышления выше, чем прочие, и более склонны основывать свою веру в превосходство человека на этой способности.

Я не считаю это возражение достаточно существенным, для того чтобы искать опровержение. Утешение здесь было бы более уместно; не предложить ли искать его в учении о переселении душ?

14.6.3. Математическое возражение. Имеется ряд результатов математической логики, которые можно использовать, чтобы показать наличие определенных ограничений на возможности машин с дискретными состояниями. Наиболее известный из этих результатов – теорема Гёделя (Gödel 1931) – показывает, что в любой достаточно мощной логической системе можно сформулировать такие утверждения, которые внутри этой системы нельзя ни доказать, ни опровергнуть, если только сама система непротиворечива. Имеются и другие, в некотором отношении аналогичные, результаты, принадлежащие Чёрчу (Church 1936b), Клини (Kleene 1935a,b), Россеру и Тьюрингу (Turing 1936, здесь глава 6). Последний результат особенно удобен для нас, так как относится непосредственно к машинам, в то время как другие можно использовать лишь как сравнительно косвенный аргумент (например, если бы мы стали опираться на теорему Гёделя, нам понадобились бы еще и некоторые средства описания логических систем в терминах машин и машин в терминах логических систем). Результат, о котором мы говорим, относится к такой машине, которая, в сущности, является цифровой вычислительной машиной с неограниченной памятью, и устанавливает, что существуют определенные вещи, которые эта машина не может выполнить. Если она устроена так, чтобы давать ответы на вопросы, как в «игре в имитацию», то будут вопросы, на которые она или даст неверный ответ, или не сможет дать ответа

вообще, сколько бы ни было ей предоставлено для этого времени. Таких вопросов, конечно, может быть много, и на вопросы, на которые нельзя получить ответ от одной машины, можно получить удовлетворительный ответ от другой. Мы здесь, разумеется, предполагаем, что вопросы принадлежат скорее к таким, которые допускают ответ «да» или «нет», чем к таким, как «что вы думаете о Пикассо?». Следующего типа вопросы относятся к числу таких, на которые, как нам известно, машина не может дать ответ: «Рассмотрим машину, характеризуемую следующим образом: .... Будет ли эта машина всегда отвечать "да" на любой вопрос?» Вместо точек следует подставить описание в какой-либо стандартной форме типа той, что была использована нами в § 14.5. Если описанная машина находится в некотором сравнительно простом отношении к машине, к которой мы обращаемся с нашим вопросом, то можно показать, что ответ на этот вопрос окажется либо неверным, либо его вовсе не будет. В этом и состоит математический результат; утверждается, что он доказывает ограниченность возможностей машин, которая не присуща разуму человека.

Краткий ответ на это возражение состоит в следующем. Установлено, что возможности любой конкретной машины ограничены, однако в разбираемом возражении содержится голословное, без какого бы то ни было доказательства, утверждение, что подобные ограничения не применимы к разуму человека. Но я не думаю, что эту сторону дела можно с легким сердцем игнорировать. Когда какой-либо из такого рода машин задают подходящий критический вопрос и она дает определенный ответ, мы заранее знаем, что ответ будет неверным, и это дает нам чувство известного превосходства. Не является ли это чувство иллюзорным? Несомненно, оно бывает довольно искренним, но я не думаю, что ему следовало бы придавать слишком большое значение. Мы сами слишком часто даем неверные ответы на вопросы, чтобы то чувство удовлетворения, которое возникает у нас при виде погрешимости машин, имело оправдание. Кроме того, чувство превосходства может относиться лишь к машине, над которой мы одержали свою – в сущности, весьма скромную – победу. Не может быть и речи об одновременном торжестве над всеми машинами. Короче говоря, для любой отдельной машины могут найтись люди, которые умнее ее, однако в этом случае снова могут найтись другие, еще более умные машины, и т. д.

Я думаю, что те, кто разделяет точку зрения, выраженную в математическом возражении, как правило, охотно примут «игру в имитацию» в качестве основы дальнейшего рассмотрения. Те же, кто убежден в справедливости двух предыдущих возражений, будут, вероятно, вообще не заинтересованы ни в каком тесте.

14.6.4. Возражение с точки зрения сознания. Это возражение особенно ярко выражено в выступлении профессора Джефферсона (Jefferson 1949), откуда я и привожу цитату. «До тех пор, пока машина не сможет написать сонет или сочинить музыкальное произведение, побуждаемая к тому собственными мыслями и эмоциями, а не за счет случайного совпадения символов, мы не можем согласиться с тем, что она равносильна мозгу, т. е. что она может не только написать это, но и понять то, что ею написано. Ни один механизм не

может чувствовать (а не просто искусственно сигналить, для чего требуется достаточно несложное устройство) радость от своих успехов, горе от постигших его неудач, удовольствие от лести, огорчение из-за совершенной ошибки, не может быть очарован противоположным полом, не может сердиться или быть удрученным, если ему не удается добиться желаемого».

Это рассуждение, по-видимому, означает отрицание нашего теста. Согласно самой крайней форме этого взгляда, единственный способ, с помощью которого можно удостовериться в том, что машина способна мыслить, состоит в том, чтобы стать машиной и осознавать процесс собственного мышления. Свои переживания можно было бы потом поведать миру, но, конечно, подобное сообщение никого бы не удовлетворило. Точно так же, если следовать этому взгляду, то окажется, что единственный способ убедиться в том, что данный *человек* действительно мыслит, состоит в том, чтобы стать именно этим человеком. Фактически это философия солипсизма. Быть может, подобные воззрения весьма логичны, но если исходить из них, то обмен идеями становится весьма затруднительным. Согласно этой точке зрения, А обязан думать, что «А мыслит, а В нет», в то время как В убежден в том, что «В мыслит, а А нет». Вместо того чтобы постоянно спорить по этому вопросу, обычно принимают вежливое соглашение о том, что мыслят все.

Я уверен, что профессор Джефферсон не пожелает согласиться с этой крайней солипсистской точкой зрения. Вероятно, он весьма охотно принял бы в качестве теста «игру в имитацию». Эта игра (если игрок В не участвует) нередко применяется на практике под названием *viva voce (устный экзамен)*, для того чтобы установить, действительно ли данный человек понял некоторое положение или «заучил, как попугай». Вот возможный отрывок из такой игры.

Вопрошающий: Не находите ли Вы, что в первой строке Вашего сонета «Сравню ль тебя я с летним днем» выражение «с весенним днем» звучало бы лучше?

Отвечающий: Оно нарушало бы размер стиха.

Вопрошающий: А если сказать «с зимним днем»? С размером здесь все обстоит благополучно.

Отвечающий: Это так, но никто не захочет, чтобы его сравнивали с зимним днем.

Вопрошающий: А разве мистер Пиквик не напоминает Вам Рождество?

Отвечающий: Некоторым образом да.

Вопрошающий: Но Рождество – зимний день, и я не думаю, чтобы мистер Пиквик имел что-нибудь против этого сравнения.

Отвечающий: Я не думаю, что вы говорите все это всерьез. Когда говорят о зимнем дне, имеют в виду обычно зимний день, а не какой-то особенный, вроде Рождества.

И т. д. Что бы сказал профессор Джефферсон, если бы машина, пишущая сонеты, могла отвечать примерно так, как это было в приведенном выше отрывке из viva voce. Не знаю, стал ли бы он рассматривать ответы машины лишь как «просто искусственную сигнализацию». Если бы ее ответы были столь же связными и удовлетворительными по содержанию, как в приве-

денном выше отрывке, я не думаю, чтобы профессор Джефферсон охарактеризовал это как дело, выполнить которое может «достаточно несложное устройство». Эту фразу из его выступления следует, по-видимому, относить к таким случаям, когда в машине имеется, скажем, граммофонная пластинка с записью сонета в чьем-либо исполнении, а также механизм, с помощью которого эту запись можно время от времени включать.

Короче говоря, я считаю, что большинство из тех, кто поддерживает возражение с точки зрения сознания, скорее откажутся от своих взглядов, чем признают солипсистскую точку зрения. В таком случае они, по-видимому, охотно примут наш тест.

Мне не хотелось бы создавать впечатление, будто я считаю, что в сознании нет ничего загадочного. Например, неудача наших попыток локализовать сознание похожа на парадокс. Но я вовсе не думаю, что загадки, связанные с сознанием, непременно должны быть разъяснены прежде, чем мы окажемся в состоянии ответить на вопрос, рассматриваемый в настоящей статье.

14.6.5. Возражения, исходящие из того, что машина не все может выполнить. Обычно эти возражения выражают в такой форме: «Я согласен с тем, что вы можете заставить машины делать все, о чем упоминали, но вам никогда не удастся заставить их делать X». При этом перечисляют довольно длинный список значений этого X. Я предлагаю читателю выбирать: «Быть добрым, находчивым, красивым, дружелюбным, быть инициативным, обладать чувством юмора, отличать правильное от неправильного, совершать ошибки, влюбляться, получать удовольствие от клубники со сливками, заставить кого-нибудь полюбить себя, извлекать уроки из своего опыта, правильно употреблять слова, думать о себе, обладать таким же разнообразием в поведении, каким обладает человек, создавать нечто подлинно новое».

Обычно в подтверждение подобных высказываний не приводят никаких доводов. Я убежден, что эти высказывания основываются главным образом на *принципе неполной индукции*. Человек в течение своей жизни видел тысячи машин. Из того, что он видел, он делает ряд общих заключений. Машины безобразны; каждая из них создана для того, чтобы выполнять весьма ограниченные задачи; если необходимо сделать нечто иное, они бесполезны; вариации их поведения крайне незначительны и т. д. и т. п. Естественно, человек делает вывод, что все это – необходимые свойства машин вообще. Многие из этих ограничений связаны с очень маленькой емкостью памяти большинства машин. (При этом я предполагаю, что понятие емкости памяти машины несколько обобщено таким образом, что охватывает и машины, отличные от машин с дискретными состояниями. Точное определение не играет здесь никакой роли, так как в настоящем обсуждении мы не претендуем на математическую строгость.) Несколько лет назад, когда очень немногие знали о цифровых вычислительных машинах, часто приходилось встречаться с недоверчивым отношением к тому, что о них рассказывали, если об их замечательных свойствах говорили, не объясняя, как такие машины устроены. Это, вероятно, происходило из-за того, что слушавшие шаблонно применяли принцип неполной индукции. Разумеется, применение этого принципа происходило в основном бессознательно. Если ребенок, обжегшись один раз,

боится огня и выражает страх перед огнем тем, что избегает его, то я бы сказал, что он применяет неполную индукцию (само собой разумеется, поведение ребенка можно описать и по-другому). Я не думаю, чтобы трудовая деятельность и обычаи человечества были особенно удачным материалом для применения неполной индукции. Если мы хотим получить надежные результаты, то необходимо исследовать очень большую часть пространственно-временного континуума. В противном случае мы можем прийти, скажем, к выводу (к которому приходит большинство английских детей), что все говорят по-английски и что глупо изучать французский язык.

Однако относительно многого из того, что было названо в числе вещей, недоступных машине, следует сделать особые оговорки. Неспособность машины получать удовольствие от клубники со сливками может показаться читателю пустяком. Весьма возможно даже, что мы могли бы сделать так, чтобы машина получала удовольствие от этого изысканного блюда, но любая попытка в этом направлении была бы идиотизмом. Эта неспособность машины приобретает значение лишь в сочетании с другими труднодоступными для нее вещами, например в сочетании с трудностью установления между нею и человеком такого же отношения дружелюбия, какое бывает между двумя людьми.

Утверждение «машины не могут совершать ошибок» кажется мне курьезным. Его пытаются парировать: «А разве они от этого хуже?» Отнесемся к этому утверждению снисходительно и попытаемся понять, что имеют в виду в действительности. Я думаю, что возражение, содержащееся в утверждении «машины не могут совершать ошибок», можно пояснить с помощью «игры в имитацию». Требуется, чтобы исследователь отличил машину от человека, просто задавая им ряд арифметических задач; машина должна разоблачить себя вследствие своей высокой точности. Ответ на эту аргументацию очень прост. Можно сделать так, чтобы машина (запрограммированная для участия в игре) не стремилась давать *правильные* ответы на арифметические задачи. Она может в известной мере специально вводить ошибки в вычисления, для того чтобы сбить с толку задающего вопросы. Что касается ошибок, связанных с механическими неисправностями, то такие ошибки обнаружат себя, по-видимому, неправильным решением о том, какую арифметическую ошибку допустить. Однако даже такая интерпретация данного возражения не является достаточно снисходительной. Размеры настоящей статьи не позволяют нам остановиться на этом более подробно. Мне кажется, что это возражение возникает потому, что смешивают ошибки двух родов. Их можно называть «ошибками функционирования» и «ошибками логического вывода». Ошибки функционирования происходят вследствие некоторых механических или электрических неисправностей, в результате которых машина ведет себя не так, как это было намечено. В философских дискуссиях обычно отвлекаются от возможности ошибок такого рода; поэтому подвергают рассмотрению «абстрактные машины». Эти абстрактные машины – математические фикции, а не реально существующие объекты. По определению, они не могут иметь ошибок функционирования. В этом смысле мы действительно можем сказать, что «машины никогда не могут ошибаться». Ошибки логического вывода могут возникать лишь тогда, когда сигналу на выходе машины придан

определенный смысл. Например, машина может выдавать в печатном виде математические уравнения или какие-нибудь предложения на английском языке. Если при этом печатается ложное высказывание, мы говорим, что машина совершила ошибку логического вывода. У нас, очевидно, вовсе нет оснований для утверждения, что машина не может совершать ошибок этого рода. Например, она может только и делать, что печатать «0 = 1». В качестве более естественного примера рассмотрим машину, располагающую каким-то методом для того, чтобы делать заключения на основе неполной индукции. Мы должны ожидать, что такой метод в отдельных случаях будет давать ошибочные результаты.

На утверждение о том, что машина не может иметь предметом своей мысли самое себя, можно, конечно, дать ответ лишь в том случае, если бы было возможно показать, что машина вообще имеет какие-либо мысли по какому-то предмету. Все же в выражении «предмет машинных операций» заключен некоторый смысл, по крайней мере для тех, кто имеет дело с машинными вычислениями. Если, например, машина решает уравнение $x^2 - 40x - 11 = 0$, то уравнение можно считать частью предмета операций машины в данный момент. В этом смысле предметом машины, безусловно, может быть она сама. Ее можно использовать при составлении своей собственной программы или для предсказания последствий, вызываемых изменениями в ее устройстве. Наблюдая результаты своего поведения, машина сможет изменять свои собственные программы, с тем чтобы быть более эффективной в достижении некоторой цели. Все это станет возможно в ближайшем будущем; это не утопические мечты.

Возражение, состоящее в том, что машина не отличается разнообразием поведения, является всего лишь способом выражения того обстоятельства, что она не обладает большой емкостью памяти. До недавнего времени емкость памяти даже в тысячу цифр была большой редкостью.

Все возражения, которые мы сейчас разбираем, часто являются просто замаскированной формой возражения с точки зрения сознания. Обычно, если утверждают, что машина может выполнить что-нибудь из вышеперечисленного, и при этом описывают сущность метода, которым пользуется машина, это не производит большого впечатления. Считается, что в чем бы ни состоял этот метод, он должен быть весьма элементарным, так как носит механический характер. Сравните сказанное со словами Джефферсона на стр. 205 [Примечание редактора: этой книги.].

14.6.6. Возражение леди Лавлейс. Наиболее подробные сведения, которыми мы располагаем об Аналитической машине Бэббиджа, взяты из воспоминаний леди Лавлейс. В них она высказывает такую мысль [Примечание редактора: на стр. 53 этой книги.]: «Аналитическая машина не претендует на то, чтобы создавать что-то действительно новое. Машина может выполнить *все то, что мы сумеем ей предписать*» (курсив леди Лавлейс). Цитируя это высказывание, Хартри (Hartree 1949) добавляет: «Отсюда не следует, что невозможно сконструировать электронное устройство, которое “мыслит”, или допускает, пользуясь биологическими терминами, выработку условных рефлексов, на основе которых становится возможным “обучение”. Увлека-

тельный и будирующий вопрос, подсказанный некоторыми из последних достижений, состоит в том, осуществимо это принципиально или нет. Однако не видно, чтобы машины, построенные или запроектированные до настоящего времени, обладали этим свойством».

Я полностью согласен с Хартри по этому вопросу. Следует отметить, что он вовсе не утверждает в категорической форме, что машины, о которых идет речь, не обладают этим свойством. Он лишь замечает, что данные, которыми располагала леди Лавлейс, не позволяли ей допустить этого. Весьма возможно, что машины, о которых шла речь, в некотором смысле обладали этим свойством. Действительно, пусть некоторая машина с дискретными состояниями обладает рассматриваемым свойством. Аналитическая машина Бэббиджа была универсальной цифровой вычислительной машиной; это значит, что если бы она обладала нужной емкостью памяти и необходимой скоростью работы, то, будь в нее введена соответствующая программа, она могла бы подражать этой машине. По-видимому, этот довод не приходил в голову ни Бэббиджу, ни графине Лавлейс. Во всяком случае, от них нельзя требовать, чтобы они исчерпали все, что можно сказать по этому поводу.

Весь этот вопрос будет рассмотрен еще раз в разделе, посвященном обучающимся машинам.

Один из вариантов возражения леди Лавлейс – утверждение, гласящее, что машина «никогда не может создать ничего подлинно нового». На секунду возразим поговоркой, что вообще «ничто не ново под Луной». Кто может быть уверенным в том, что выполненная им «оригинальная работа» не выросла из зерна, посеянного образованием, или просто не является результатом применения хорошо известных общих принципов. Более удачный вариант этого возражения состоит в утверждении, что «машина никогда не может ничем удивить человека». Это утверждение представляет собой прямой вызов, который, однако, мы можем принять, не уклоняясь. Лично меня машины удивляют очень часто. В основном это происходит потому, что я не могу точно рассчитать, чего можно, а чего нельзя ожидать от них, или (это бывает чаще) потому, что хотя я и провожу необходимые расчеты, однако делаю это в спешке, неряшливо, рискуя ошибиться. Вот я говорю себе: «По-видимому, электрическое напряжение здесь должно быть таким же, как там: во всяком случае, будем исходить из этого предположения». Само собой разумеется, что в таких случаях я часто ошибаюсь, и получающийся результат оказывается для меня неожиданностью, так как к тому времени, когда эксперимент заканчивается, сделанное допущение уже забыто мною. Эти предположения и натяжки я оставляю открытыми до лекции на тему о моих порочных методах работы. Однако я нисколько не сомневаюсь в том, что действительно испытываю удивление перед машинами.

Я не жду, что этот ответ заставит замолчать моего оппонента. Вероятно, он скажет, что это удивление происходит вследствие некоторого творческого умственного акта с моей стороны и отражает мое недоверие к машине. Но такая аргументация уводит от вопроса о том, может ли машина чем-либо удивить человека, и возвращает снова к возражению с точки зрения сознания.

Этот способ аргументации должен, таким образом, считаться исчерпанным, хотя, быть может, стоит все же отметить то обстоятельство, что признание чего-то удивительным таки требует «творческого умственного акта» независимо от того, что является источником удивившего нас события: человек, книга, машина или еще что-нибудь.

Мнение о том, что машины не могут чем-либо удивить человека, основывается, как я полагаю, на одном заблуждении, которому в особенности подвержены математики и философы. Я имею в виду предположение о том, что коль скоро какой-то факт стал достоянием разума, тотчас же достоянием разума становятся все следствия из этого факта. Во многих случаях это предположение может быть весьма полезно, но слишком часто забывают, что оно ложно. Естественным следствием из него является взгляд, что якобы нет ничего особенного в умении выводить следствия из имеющихся данных, руководствуясь общими принципами.

14.6.7. Возражение, основанное на непрерывности действия нервной системы. Несомненно, нервная система не является машиной с дискретными состояниями. Небольшая ошибка в информации относительно силы нервного импульса, действующего на нейрон, может привести к значительному изменению импульса на выходе. И можно предположить, что при таких условиях было бы неразумно ожидать, что удастся имитировать поведение нервной системы с помощью машины с дискретными состояниями.

То, что машина с дискретными состояниями должна отличаться от машины непрерывного действия, это, конечно, справедливо. Однако если мы будем придерживаться условий «игры в имитацию», то исследователь не сможет воспользоваться этим различием. Данную ситуацию можно сделать яснее, рассмотрев другую, более простую машину непрерывного действия. Для этого особенно хорошо подходит дифференциальный анализатор. (Дифференциальный анализатор – это машина, не относящаяся к типу машин с дискретными состояниями, он применяется для выполнения некоторых видов вычислений.) Некоторые дифференциальные анализаторы выдают ответы в печатном виде и поэтому пригодны для игры в имитацию. Цифровая вычислительная машина не может предсказать, какие в точности ответы даст дифференциальный анализатор, решая некоторую задачу, но зато она может сама находить правильные ответы. Например, если требуется найти значение числа π (в действительности приблизительно равное 3,1416), то цифровая вычислительная машина могла бы осуществлять случайный выбор значения из множества чисел 3,12; 3,13; 3,14; 3,15; 3,16 со следующими (к примеру) вероятностями: 0,05; 0,15; 0,55; 0,18; 0,06. При таких условиях исследователю будет очень трудно отличить дифференциальный анализатор от цифровой вычислительной машины.

14.6.8. Возражение с точки зрения неформальности поведения человека. Невозможно выработать правила, предписывающие, что именно должен делать человек во всех случаях, при всевозможных обстоятельствах. Например, пусть имеется правило, согласно которому человеку следует остановиться, если включен красный свет светофора, и продолжать движение,

если свет зеленый; но как быть, если по ошибке оба световых сигнала появятся одновременно? По-видимому, безопаснее всего остановиться. Однако это решение в дальнейшем может быть источником каких-либо новых затруднений. Рассуждая так, мы приходим к заключению, что любая попытка сформулировать правила действия, предусматривающие любой возможный случай, обречена на провал, даже если ограничиться областью транспортной сигнализации. Со всем этим я согласен.

Основываясь на сказанном, делают вывод, что мы не можем быть машинами. Я попытаюсь воспроизвести это рассуждение, хотя боюсь, что вряд ли сумею сделать это хорошо. Выглядит оно приблизительно так: «Если бы каждый человек обладал определенной совокупностью правил действия, следуя которым, он живет, он был бы не чем иным, как машиной. Однако таких правил не существует. Следовательно, человек не может быть машиной». В этом рассуждении бросается в глаза нераспределенный средний член. Я не думаю, чтобы когда-нибудь это рассуждение излагали именно в такой форме, но убежден, что нечто подобное все же встречается. Однако возможно, что некая путаница в выражениях «правила действия» и «законы поведения» затемняет вопрос. Под «правилами действия» я понимаю такие предписания, как «Остановитесь, если увидите красный свет»; они могут определять наши действия и осознаваться нами. Под «законами поведения» я понимаю управляющие человеком естественные законы, например: «Если человека ущипнуть, он вскрикнет». Если в приведенном выше рассуждении вместо «правил действия, которыми человек руководствуется в своей жизни» подставить «законы поведения, управляющие жизнью человека», то ошибка, связанная с нераспределенностью среднего члена, перестает быть непреодолимой. Ибо мы верим, что истинно не только то, что подчиняться законам поведения означает быть какой-то машиной (хотя необязательно машиной с дискретными состояниями), но и обратное – что быть машиной означает подчиняться таким законам. Однако убедить себя в отсутствии исчерпывающих законов поведения не так легко, как в отсутствии исчерпывающих правил действия. Единственный известный нам способ отыскания таких законов – научное наблюдение, и мы, безусловно, не знаем о таких обстоятельствах, когда мы могли бы сказать: «Мы искали достаточно. Таких законов не существует».

Мы можем более убедительно показать, что любое утверждение такого рода является необоснованным. Действительно, допустим, что мы могли бы отыскать такие законы (если они существуют). Тогда если нам будет предъявлена некоторая машина с дискретными состояниями, то посредством наблюдения над ней мы наверняка смогли бы узнать достаточно, чтобы предсказать ее будущее поведение, причем сделать это можно будет в приемлемый срок, скажем за 1000 лет. Но, по-видимому, дело обстоит не так. Я вводил в манчестерскую вычислительную машину небольшую программу, занимающую 1000 ячеек памяти, используя которую, машина в ответ на введенное в нее 16-значное число выдает в течение двух секунд другое 16-значное число. Попытайтесь-ка извлечь из этого такую информацию о программе машины, которая была бы достаточна для предсказания ее ответа на любое еще не испробованное число. Держу пари, что вам это не удастся.

14.6.9. Возражение с точки зрения сверхчувственного восприятия. Я предполагаю, что читатель знаком с идеей сверхчувственного восприятия и четырьмя его разновидностями: телепатия, ясновидение, способность к прорицанию и психокинез. Эти поразительные явления, по-видимому, опровергают все наши обычные научные представления. Как бы нам хотелось доказать их несостоятельность! К несчастью, статистические данные, по крайней мере в случае телепатии, на их стороне. Очень трудно перестроить наши взгляды так, чтобы охватить и эти новые факты, ибо тот, кто верит в сверхчувственное восприятие, по-видимому, не так уж далек от веры в чертей и духов. Ведь представление о том, что жизнь и деятельность человека подчиняются только физическим законам – как тем, которые нам уже известны, так и тем, которые еще не открыты, но которые предполагаются в некотором смысле аналогичными уже открытым, – напрашивается прежде всего. Возражение с точки зрения сверхчувственного восприятия, по моему мнению, является достаточно серьезным. Его можно было бы парировать, сказав, что многие научные теории, несмотря на весь шум вокруг сверхчувственного восприятия, остаются применимыми на практике, так что в действительности можно прекрасно обойтись и без него, попросту забыв о его существовании. Это, пожалуй, слабое утешение; есть опасение, что мышление принадлежит как раз к тем явлениям, к которым сверхчувственное восприятие может иметь самое непосредственное отношение.

Не в столь общей форме возражение, основанное на сверхчувственном восприятии, может быть выражено так: «Будем играть в имитацию, используя в роли отвечающих человека, способного воспринимать телепатические воздействия, и цифровую вычислительную машину». Исследователь может сформулировать, например, такой вопрос: «Какой масти карта в моей правой руке?» Человек с помощью телепатии или ясновидения дает правильные ответы в 130 случаях из 400. Ответы же машины могут только случайно оказаться правильными, и она сможет угадать масть, скажем, лишь в 104 случаях. Это позволит исследователю отличить человека от машины. Здесь открывается интересная возможность. Допустим, что в нашей цифровой вычислительной машине имеется генератор случайных чисел. Было бы естественно использовать его для решения о том, какой ответ дать. Но тогда этот генератор случайных чисел может быть подвержен влиянию психокинетических способностей исследователя. Возможно, что психокинез приведет к тому, что машина будет давать правильные ответы гораздо чаще, чем предсказывает теория вероятностей, так что исследователь может оказаться не в состоянии выяснить, кто есть кто. С другой стороны, он может, вообще не задавая никаких вопросов, узнать это с помощью ясновидения: если в дело вмешивается сверхчувственное восприятие, возможно еще и не такое.

Если считать, что телепатия возможна, необходимо ввести ограничения в наш тест. Можно, например, требовать, чтобы ситуация была аналогична той, которая возникает, когда задающий вопросы обращается к самому себе, а один из участников игры подслушивает его через стенку. Чтобы удовлетворить всем требованиям нашей игры, отвечающих следовало бы поместить в комнату, «защищенную от телепатии».

14.7. Обучающиеся машины

Читатель, вероятно, уже понимает, что у меня нет особенно убедительных аргументов позитивного характера в пользу своей собственной точки зрения. Если бы у меня были такие аргументы, я не стал бы так мучиться, разбирая ошибки, содержащиеся во мнениях, противоположных моему собственному. Сейчас я изложу те доводы, которыми располагаю.

Вернемся ненадолго к возражению леди Лавлейс, согласно которому машина может выполнять лишь то, что мы ей приказываем. Можно сказать, что человек «вставляет» в машину ту или иную идею, и машина, прореагировав на нее некоторым образом, возвращается затем в состояние покоя, подобно фортепианной струне, по которой ударил молоточек. Другое сравнение: атомный реактор, размеры которого не превышают критических. Идея, вводимая человеком в машину, соответствует здесь нейтрону, влетающему в реактор извне. Каждый такой нейтрон вызывает некоторое возмущение, которое в конце концов затухает. Но если размер реактора превосходит критический, то весьма вероятно, что возмущение, вызванное влетевшим нейтроном, будет нарастать и приведет в конце концов к разрушению реактора. Имеют ли место аналогичные явления в случае человеческого разума и существует ли нечто подобное в случае машин? В первом случае, кажется, следует дать утвердительный ответ. Большинство умов, по-видимому, являются «подкритическими», т. е. соответствуют, если пользоваться приведенным выше сравнением, подкритическим размерам атомного реактора. Идея, ставшая достоянием такого ума, в среднем порождает менее одной идеи в ответ. Несравненно меньшую часть умов составляют умы сверхкритические. Идея, ставшая достоянием такого ума, может породить целую «теорию», состоящую из вторичных, третичных и еще более отдаленных идей. С изрядной долей уверенности можно предположить, что ум животных подкритический. Развивая нашу аналогию, мы ставим вопрос: «Можно ли сделать машину сверхкритической?»

Для уяснения поставленного вопроса имеет смысл прибегнуть еще к одной аналогии, именно – уподобить человеческий разум луковице. Рассматривая функции ума или мозга, мы обнаруживаем определенные операции, которые возможно полностью объяснить в терминах чисто механического процесса. Можно сказать, что они не соответствуют подлинному разуму: это слой, который следует удалить, для того чтобы обнаружить настоящий разум. Однако, рассматривая оставшуюся часть, мы обнаружим следующий слой, который следует удалить, и т. д. Возникает вопрос: если мы будем продолжать этот процесс, удастся ли нам прийти когда-нибудь к «настоящему» разуму или же в конце концов мы снимем слой, под которым ничего не останется? В последнем случае мы считаем, что разум имеет механический характер. (Правда, он не может быть машиной с дискретными состояниями. Этот вопрос мы уже рассматривали.)

Два последних абзаца вовсе не претендуют на роль убедительных доказательств. Их скорее следовало бы считать аргументами риторического характера.

Единственное убедительное доказательство, которое могло бы подтвердить правильность нашей точки зрения, приведено в начале раздела и состоит в том, чтобы подождать до конца нашего столетия и провести описанный эксперимент. А что же можно сказать в настоящее время? И что можно было бы предпринять уже сейчас, если исходить из предположения, что эксперимент окажется успешным?

Как я уже объяснял, проблема заключается главным образом в программировании. Прогресс в инженерном деле также необходим, однако маловероятно, чтобы затруднение возникло с этой стороны. Оценки емкости памяти человеческого мозга колеблются от 10^{10} до 10^{15} двоичных цифр. Я склоняюсь к нижней границе и убежден, что лишь очень небольшая доля памяти человека используется в высших типах мышления, причем из того, что используется, большая часть служит сохранению зрительных восприятий. Для меня было бы неожиданностью, если бы оказалось, что для игры в имитацию на удовлетворительном уровне требуется емкость памяти, превышающая 10^9, во всяком случае если бы игра велась против слепого человека. (Заметьте: емкость *«Британской энциклопедии»*, 11-е изд., составляет 2×10^9.) Емкость памяти, равная 10^7, практически представляется вполне осуществимой даже при современном состоянии техники. Вероятно, нет необходимости вообще далее увеличивать скорость машинных операций. Те части современных машин, которые можно рассматривать как аналоги нервных клеток, работают примерно в тысячу раз быстрее последних. Это создает «запас надежности», способный компенсировать потери в быстродействии, возникающие во многих случаях. Перед нами стоит задача составить машинную программу для игры в имитацию. В настоящее время я программирую со скоростью примерно тысяча знаков в день; если исходить из такой скорости программирования, то получится, что шестьдесят работников могли бы полностью закончить работу, о которой идет речь, если бы работали непрерывно в течение пятидесяти лет, при условии, конечно, что ничего не пойдет в корзину для бумаг. Желателен, по-видимому, какой-нибудь более производительный метод.

Пытаясь имитировать ум взрослого человека, мы вынуждены много размышлять о процессе, в результате которого человеческий интеллект достиг своего нынешнего состояния. Мы можем выделить три компонента:

(a) первоначальное состояние ума, скажем, в момент рождения;
(b) воспитание, объектом которого он был;
(c) другого рода опыт, воздействовавший на ум, – опыт, который нельзя назвать воспитанием.

Почему бы нам, вместо того чтобы пытаться создать программу, имитирующую ум взрослого, не попытаться создать программу, которая бы имитировала ум ребенка? Ведь если ум ребенка получает соответствующее воспитание, он становится умом взрослого человека. Как можно предположить, мозг ребенка в некотором отношении подобен блокноту, который мы покупаем в киоске: совсем небольшой механизм и очень много чистой бумаги. Наш расчет состоит в том, что механизм в мозгу ребенка настолько несложен, что устройство, ему подобное, может быть легко запрограммировано. В ка-

честве первого приближения можно предположить, что количество труда, необходимое для воспитания такой машины, почти совпадает с тем, которое необходимо для воспитания ребенка.

Таким образом, мы расчленили нашу проблему на две части: на задачу построить «программу-ребенка» и задачу осуществить процесс воспитания. Обе эти части тесно связаны друг с другом. Вряд ли нам удастся получить хорошую «машину-ребенка» с первой же попытки. Надо провести эксперимент по обучению какой-либо из машин такого рода и выяснить, как она поддается научению. Затем провести тот же эксперимент с другой машиной и установить, какая из двух машин лучше. Существует очевидная связь между этим процессом и эволюцией в живой природе, которая обнаруживается, когда мы производим такие отождествления:

структура «машины-ребенка» = наследственный материал;

изменения, происходящие в «машине-ребенке» = мутация;

решение экспериментатора = естественный отбор.

[Примечание редактора: стороны третьего равенства переставлены по сравнению с оригинальной статьей, так чтобы все эволюционные члены находились справа.] Тем не менее можно надеяться, что этот процесс будет протекать быстрее, чем эволюция. Выживание наиболее приспособленных является слишком медленным способом оценки преимуществ. Экспериментатор, применяя силу интеллекта, может ускорить процесс оценки. Не менее важно и то, что он не ограничен использованием только случайных мутаций. Если экспериментатор может проследить причину некоторого недостатка, он, вероятно, в состоянии придумать и такого рода мутацию, которая приведет к необходимому улучшению.

Невозможно применять к машине в точности тот же процесс обучения, что и к нормально развитому ребенку. Например, машину нельзя снабдить ногами, поэтому ее нельзя попросить выйти и принести ведро угля. Машина, по-видимому, не будет обладать глазами. И как бы хорошо ни удалось восполнить эти недостатки с помощью различных остроумных приспособлений, такое существо нельзя будет послать в школу без того, чтобы другие дети не потешались над ним. И вот такое существо мы должны чему-то научить. Отметим, что не стоит особенно беспокоиться относительно ног, глаз и т. д. Пример мисс Елены Келлер показывает, что воспитание возможно, если только удается тем или иным способом установить двухстороннюю связь между учителем и учеником. Обычно процесс обучения в нашем представлении связан с наказаниями и поощрениями. Идея применения какой-либо формы этого принципа обучения может лежать в основе конструирования и программирования некоторых простых «машин-детей». В этом случае машину следует устроить таким образом, чтобы поступление в нее сигнала-«наказания» приводило к резкому уменьшению вероятности повторения тех реакций машины, которые непосредственно предшествовали этому сигналу, в то время как сигнал-«поощрение», наоборот, увеличивал бы вероятность тех реакций, которые ему предшествовали (которые его «вызвали»). Все это

не предполагает со стороны машины никаких чувств. Я проделал несколько экспериментов с одной такой «машиной-ребенком» и достиг кое-какого успеха в обучении ее некоторым вещам, но метод обучения был слишком необычен, чтобы эксперимент можно было считать действительно успешным.

Применение поощрений и наказаний в лучшем случае может быть лишь частью процесса обучения. Проще говоря, если у учителя нет других средств общения со своими учениками, то количество информации, которое может получить ученик, не превышает общего числа примененных к нему поощрений и наказаний. Вероятно, к тому времени, когда ребенок выучит наизусть стихотворение «Касабьянка»[1], он будет до крайности измучен, если процесс обучения будет идти по методу игры в «20 вопросов»[2], причем каждое «нет» учителя будет принимать для ученика форму подзатыльника. В силу этого необходимо иметь какие-то другие, «неэмоциональные» каналы связи. Если такие каналы имеются, то, применяя поощрения и наказания, машину можно было бы научить выполнять команды, отдаваемые на каком-либо – например, символическом – языке. Эти команды следует передавать по «неэмоциональным каналам». Применение такого символического языка значительно уменьшит число требуемых поощрений и наказаний.

О том, какая степень сложности является наиболее подходящей для «машины-ребенка», могут быть различные мнения. Можно стремиться к тому, чтобы «машина-ребенок» была настолько простой, насколько этого возможно добиться, не нарушая общих принципов. Можно идти противоположным путем: «встраивать» сложную систему логического вывода. В последнем случае значительную часть запоминающего устройства заняли бы определения и суждения. Суждения по своему характеру должны быть различного рода, например: утверждения о хорошо известных фактах, предположения, математически доказанные теоремы, высказывания авторитетных лиц, выражения, по своей логической форме являющиеся суждениями, но не претендующие на верность. Некоторые из этих суждений могут быть охарактеризованы как «приказания». Машину следует построить так, чтобы, как только некоторое приказание оценивается ею как «вполне достоверное», автоматически выполнялась бы соответствующая операция. Чтобы пояснить это, предположим, что учитель говорит машине: «Теперь выполняй домашнее задание», – а машина реагирует на это, включая ситуацию «Учитель говорит машине: “Теперь выполняй домашнее задание”» в состав вполне достоверных фактов.

---

[1] «Касабьянка» (Casabianca) – стихотворение английской поэтессы Фелиции Хеманс (Felicia Hemans, 1793–1835). Повествует о мальчике десяти лет, сыне капитана Касабьянки, который вместе с отцом погиб на горящем военном корабле, отказавшись покинуть судно, взорванное командиром Касабьянкой во время морского боя. – *Прим. перев.*

[2] «Двадцать вопросов» – игра в вопросы и ответы, вариант которой у нас иногда называется «данетка». Один из играющих задумывает некоторое понятие. Другой играющий отгадывает задуманное, задавая вопросы, предполагающие ответы (обязательно правдивые) вида «да» или «нет». Количество вопросов, которое имеет право задать отгадчик, не должно превышать некоторого заранее установленного числа. Отгадчик выигрывает, если при указанных условиях отгадает, что же было задумано первым играющим. – *Прим. перев.*

Другим фактом такого же рода в ней может быть: «Все, что говорит учитель, истинно». Комбинация этих фактов может в заключение вести к тому, что приказание «Теперь выполняй домашнее задание» также будет включено в разряд вполне надежных фактов, а это, в свою очередь, будет означать, в силу устройства нашей машины, что последняя действительно начнет выполнять домашнее задание, – что нам и было нужно. Процесс логического вывода, применяемый машиной, вовсе не обязательно должен удовлетворять требованиям самых строгих логиков. Например, может отсутствовать иерархия типов. Но это вовсе не означает, что машина допустит ошибку типизации с вероятностью, большей вероятности падения человека в неогороженную пропасть. В рассматриваемом случае подходящие приказания (выраженные внутри системы формального вывода, а не составляющие часть ее правил), например такие, как «Не использовать некоторый класс, если он не является подклассом класса, который ранее упоминался учителем», могут иметь такой же эффект, как предупреждение: «Не подходите слишком близко к краю обрыва».

Приказания, которые может выполнять машина, не имеющая ни рук, ни ног, должны касаться преимущественно интеллектуальных сторон деятельности, как это было в приведенном выше примере (с домашним заданием). Из такого рода приказаний наиболее важными будут те, что определяют порядок, в котором следует применять правила рассматриваемой логической системы. Ибо на каждой стадии применения логической системы перед нами открывается большое число возможных шагов, которые исключают друг друга и любой из которых мы можем осуществить, следуя правилам рассматриваемой системы. Как производится такой выбор – в этом и выражается различие между глубоким и посредственным умом, но это не имеет отношения к правильности или неправильности рассуждений. Суждения, которые порождают приказания такого рода, могут быть, например, такими: «Если упоминается Сократ, применить силлогизм модуса Barbara[1]» – или: «Если один метод приводит к результату быстрее, чем второй, не применять более медленного». Одни из них могут исходить от «авторитетного лица», другие же могут вырабатываться самой машиной, например с помощью неполной индукции. Некоторым читателям мысль об обучающейся машине может показаться парадоксальной. Как могут меняться правила, по которым машина производит операции? Ведь правила должны полностью описывать поведение машины – независимо от того, какова была ее предыстория и какие изменения она претерпела. Таким образом, правила должны быть абсолютно инвариантными относительно времени. Все это, конечно, верно. Объяснение этого парадокса состоит в том, что правила, которые меняются в процессе научения, на многое не претендуют, ибо их применимость носит преходящий характер. Читатель может провести параллель с конституцией Соединенных Штатов.

Важная особенность обучающейся машины состоит в том, что ее учитель в значительной мере не осведомлен о многом из того, что происходит внут-

[1] О модусах силлогизмов и, в частности, о модусе Barbara см. статью в Википедии https://ru.wikipedia.org/wiki/Категорический_силлогизм. – *Прим. перев.*

ри нее, хотя он все же в состоянии в известных пределах предсказывать поведение своей ученицы. Сказанное особенно применимо к дальнейшему воспитанию машины, прошедшей уже хорошую подготовку и вышедшей из начальной стадии «машины-ребенка». Такое положение, очевидно, в корне отличается от обычного подхода, связанного с применением машин для вычислений, когда мы стремимся к тому, чтобы иметь ясное представление о состоянии машины в любой момент вычисления, достичь чего можно лишь с трудом. В свете сказанного взгляд, что «машина может выполнить только то, что мы умеем ей предписать», кажется странным. Большинство программ, которые мы можем ввести в машину, вызывают в ее работе нечто такое, что мы вообще не в состоянии осмыслить или рассматриваем как чисто случайное поведение. Интеллектуальное поведение предполагает, по-видимому, некоторое отступление от абсолютно детерминированного поведения в процессе вычисления; это отступление, однако, должно быть очень незначительным, чтобы не вызвать полностью случайного поведения или бессмысленных повторяющихся циклов. Другой важный результат обучения как способа подготовки нашей машины к участию в игре в имитацию состоит в том, что «присущая человеку склонность к ошибкам» будет, по-видимому, обойдена естественным образом, т. е. без специального «натаскивания». (Читателю следует примирить это с точкой зрения, изложенной на стр. 208.) Процесс обучения не обязательно должен давать стопроцентную гарантию результата; если неудача случается, то «разучиться» невозможно.

Вероятно, в обучающуюся машину имеет смысл ввести элемент случайности. Это полезно при поиске решения какой-нибудь задачи. Пусть, например, требуется найти число, расположенное между 50 и 200 и равное квадрату суммы своих цифр; мы можем сначала проверить число 51, затем 52 и продолжать до тех пор, пока не найдем число, удовлетворяющее условию задачи. Но мы можем поступить и иначе: выбирать числа наугад до тех пор, пока не найдем искомое. Этот метод имеет то преимущество, что не требуется хранить в памяти уже проверенные значения; однако он имеет и отрицательную сторону, состоящую в том, что одно и то же число может быть подвергнуто проверке повторно, но это не так уж существенно, если задача имеет несколько решений. Систематический метод имеет другой недостаток: может случиться, что придется проверить очень длинный блок значений, не содержащий ни одного решения, прежде чем будет найдено первое число, обладающее нужным свойством. В нашем случае процесс обучения можно рассматривать как поиск такой формы поведения, которая удовлетворяла бы требованиям учителя (или какому-нибудь другому критерию). Поскольку в этом случае, по-видимому, имеется весьма большое число решений, отвечающих предъявленным требованиям, постольку метод случайного выбора представляется нам предпочтительнее систематического. Следует отметить, что метод случайного отбора применяется и в другом аналогичном процессе – в эволюции. Но там систематический метод невозможен вообще. Не ясно, как можно было бы сохранять информацию об уже опробованных генетических комбинациях, с тем чтобы предупредить их повторение в будущем.

Мы можем надеяться, что машины в конце концов будут успешно соперничать с людьми во всех чисто интеллектуальных областях. Но с чего лучше всего начать? Решение даже этого вопроса наталкивается на затруднения. Многие считают, что начать лучше всего с какой-нибудь очень абстрактной деятельности, например с игры в шахматы. Другие предлагают снабдить машину хорошими органами чувств, а затем научить ее понимать и говорить по-английски. В этом случае машину можно будет обучать, как ребенка: указывать на предметы и называть их и т. д. В чем состоит правильный ответ на этот вопрос, я не знаю, но думаю, что следует испытать оба подхода.

Мы можем заглядывать в будущее лишь очень недалеко, но уже сейчас очевидно, что многое еще предстоит сделать.

15 Наилучший метод конструирования автоматической вычислительной машины (1951)

Морис Уилкс

В 1946 году Морис Уилкс (1913–2010), тогда возглавлявший Математическую лабораторию в Кембриджском университете, Англия, присутствовал на летней школе, состоявшейся в Электротехнической школе Мура при Пенсильванском университете, и там узнал о компьютере EDVAC Бёркса, Голдстайна, фон Неймана и др. На обратном пути он начал проектировать машину, которую назвал Electronic Delay Storage Automatic Calculator (электронно-счетная машина с запоминающим устройством на линиях задержки), сокращенно EDSAC. К 1949 году, когда EDSAC заработала, в Англии уже работал компьютер с хранимой программой под названием Mark 1, созданный в Манчестерском университете. Этот проект упоминается Уилксом в отрывке, приведенном ниже, и Тьюрингом в главе 14 (стр. 200 и 212). Манчестерский Mark 1 был, скорее, экспериментальным прототипом, а не рабочей лошадкой для вычислений (последовавшая за ним машина была введена в коммерческую эксплуатацию в 1951 году под названием Ferranti Mark 1).

Напротив, EDSAC уже в скором времени использовался на полную мощность Кембриджским научным сообществом, а Уилкс приступил к работе над следующей машиной. Проектируя форматы данных и набор команд, он

пришел к важнейшему выводу о том, что было бы гораздо проще спроектировать более примитивный набор микрокоманд для таких микроопераций, как сдвиг битов в регистре или перемещение битов между регистрами, а затем реализовать настоящие машинные команды в виде микропрограмм, составленных из этих микрокоманд, – «наделив устройство управления полной гибкостью программируемого компьютера в миниатюре», как он писал позже (Wilkes 1986). В конце статьи он даже высказывает предположение, что когда-нибудь программисты смогут сами выбирать для себя наборы команд.

Микрокод действительно очень скоро был признан «наилучшим методом» проектирования компьютера. Позже, в 1950-х годах, IBM положила микрокод в основу проектирования своей линейки компьютеров System/360, чем сильно упростила процесс отладки и даже модификацию структуры таких сложных операций, как умножение, прямо в месте эксплуатации. Идея гибкого, иерархического, допускающего обновление дизайна стоит за ныне вездесущими прошивками в современных системах.

Уилкс впоследствии сделал чрезвычайно успешную карьеру в качестве профессора в Кембридже и написал один из первых учебников по программированию компьютеров. В 1967 году он получил премию Тьюринга за большой и разнообразный вклад в информатику.

<...>

Я думаю, большинство людей согласятся с тем, что в настоящее время главной заботой конструктора является вопрос о том, как добиться от машины максимальной надежности. Среди прочего, надежность машины зависит от следующих факторов:

(a) количества содержащегося в ней оборудования;
(b) его сложности;
(c) степени повторяемости узлов.

Под сложностью машины я понимаю то, до какой степени перекрестные соединения между различными блоками затемняют их логические взаимосвязи. Машину проще ремонтировать, если составляющие ее блоки соединены простым способом, без перекрестных соединений; ее также проще конструировать, поскольку разные люди могут работать над разными блоками, не мешая друг другу.

Что касается повторяемости, я думаю, всякий предпочел бы иметь в конкретной части машины группу из пяти одинаковых блоков, чем из пяти разных. Большинство людей предпочли бы иметь шесть одинаковых блоков, чем пять разных. Насколько далеко человек готов зайти в примирении с большим количеством оборудования ради достижения повторяемости – дело вкуса. Вопрос можно поставить следующим образом. Предположим, что считаются равно приемлемыми оба варианта: когда некоторая часть машины состоит из n различных блоков и когда она состоит из kn одинаковых блоков, причем размеры всех блоков сопоставимы. Каково тогда значение k? Лично я полагаю, что $k > 2$. Следует сказать, что я имею в виду машину, включающую примерно 10 групп блоков, и что n приближенно равно 10.

Высказанные только что замечания имеют общий характер. Постараюсь теперь быть более конкретным. Конструируя параллельную машину, мы встречаемся в лице ее арифметического устройства с хорошим примером оборудования, состоящего из одинаковых блоков, повторенных много раз. Однако такое арифметическое устройство гораздо больше, чем в последовательной машине. С другой стороны, я думаю, будет справедливо сказать, что управление в параллельной машине устроено проще, чем в последовательной. Я употребляю здесь слово *управление* в очень общем смысле, подразумевая все, что не относится к памяти как таковой (т. е. схемы доступа включаются) или к регистрам и сумматорам в арифметическом устройстве. <...>

Таким образом, мы приходим к представлению об арифметическом устройстве как о конгломерате стандартных блоков, каждый из которых содержит триггеры (по одному для каждого из четырех регистров) наряду с сумматором. Имеются затворы, позволяющие передавать числа из одного регистра в другой с помощью сумматора в тех случаях, когда это необходимо. Эти передачи осуществляются подачей сигнала на один или несколько проводов, исходящих из арифметического устройства.

Необходимо также иметь регистры в управляющей части машины. Они (вместе с названиями, присвоенными им в манчестерской машине и в EDSAC) таковы:

регистр для хранения адреса следующей команды, подлежащей выполнению (управляющий или хранитель управления последовательностью);

регистр для хранения исполняемой в данный момент команды (регистр текущей команды или хранитель команды);

регистр для подсчета количества числа шагов в операции умножения или сдвига (не нужен в быстром умножителе манчестерской машины, хранитель управления хронометражем в EDSAC).

Кроме того, в манчестерской машине имеется ряд B-регистров.

Если считается, что одного B-регистра достаточно, то рассматриваемая нами параллельная машина может использовать один и тот же блок (содержащий 4 триггера и один сумматор) как для управляющих, так и для арифметических регистров. Таким образом, достигается крайняя степень повторяемости.

Остается рассмотреть собственно управление, т. е. часть машины, которая подает импульсы для приведения в действие затворов, ассоциированных с арифметическими и управляющими регистрами. Конструктор этой части машины обычно импровизирует, рисуя блок-схемы, пока не найдет конфигурацию, удовлетворяющую его требованиям и выглядящую достаточно экономичной. Я хотел бы предложить метод, при котором проектирование управления можно сделать более систематическим, а значит, и менее сложным.

Каждая операция, вызываемая из машинного кода, подразумевает последовательность шагов, которая может включать передачи из памяти в управляющие или арифметические регистры или наоборот, а также передачи из одного регистра в другой. Каждый такой шаг выполняется путем подачи

сигналов на некоторые провода, ассоциированные с управляющими или арифметическими регистрами, я буду это называть «микрооперациями». Таким образом, любая настоящая машинная операция представляет собой последовательность «микропрограмм», составленных из микроопераций.

На рис. 15.1 показано, как могут генерироваться импульсы для выполнения микроопераций. Синхроимпульс, инициирующий микрооперацию, поступает в дерево декодирования и маршрутизируется одному из выходов в соответствии с числом, установленным в регистре *R*. Он приходит в матрицу выпрямителей *A* и дает начало импульсам на некоторых исходящих из этой матрицы проводах в соответствии с организацией выпрямителей. Эти импульсы управляют затворами, ассоциированными с управляющими и арифметическими регистрами, и вызывают выполнение нужной микрооперации. Импульс от дерева декодирования также приходит в матрицу *B* и дает начало импульсам на некоторых проводах, исходящих из этой матрицы. Эти импульсы подаются по короткой линии задержки в регистр *R* и приводят к изменению хранящегося в нем числа. В результате следующий инициирующий импульс, вошедший в дерево декодирования, будет маршрутизирован другому выходу и, следовательно, вызовет выполнение другой микрокоманды. Таким образом, мы видим, что каждая строка выпрямителей в матрице *A* соответствует одной из микрокоманд в последовательности, необходимой для выполнения машинной операции.

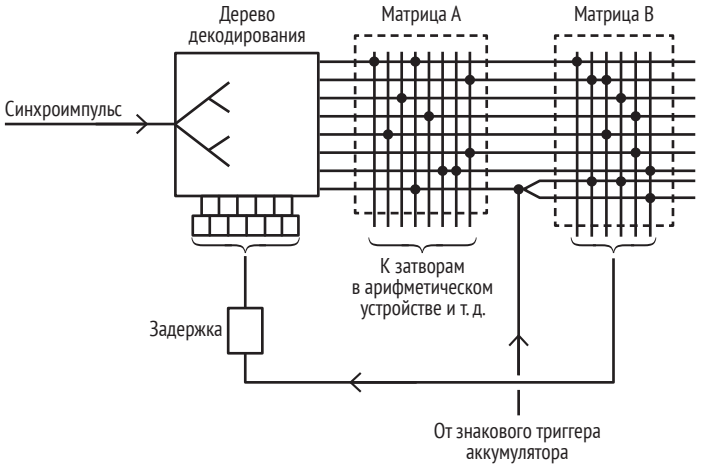


Рис. 15.1

Только что описанная система могла бы выполнять лишь фиксированный цикл операций. Ее полезность можно значительно увеличить, сделав некоторые микрокоманды условными в том смысле, что за ними следует одна из двух альтернативных микрокоманд, в зависимости от состояния машины. Это можно реализовать, заставив выход дерева декодирования разветвляться до входа в матрицу B. Ветвь, по которой пойдет импульс, управляется потенциалом на проводе, исходящем из другой части машины; например

он мог бы исходить из знакового триггера аккумулятора. Нижняя строка матрицы *A* на рис. 15.1 соответствует условной микрокоманде. Матрица *A* содержит последовательности микрокоманд для выполнения всех основных машинных операций. Все, что необходимо для выполнения конкретной операции, – переключить «микроуправление» на первую микрокоманду соответствующей последовательности. Для этого разряды, описывающие функцию команды, нужно выставить на первых четырех или пяти триггерах регистра *R*, а остальные разряды сбросить в 0.

Система управления, спроектированная таким образом, безусловно, весьма логична по структуре, однако можно было бы сделать два замечания, несколько противоречащих друг другу. Прежде всего можно было бы сказать, что в такой организации нет ничего нового, поскольку используются триггеры, затворы и смесительные диоды, т. е. элементы, из которых состоит любое управляющее устройство. С этим замечанием я склонен согласиться. На самом деле чертежи управляющих схем самых разных ныне существующих или конструируемых машин, несомненно, были бы очень похожи на рис. 15.1. Другое возражение заключается в том, что оборудование, используемое в схеме, выглядит довольно экстравагантно. Тут я не согласен, особенно если разрешены некоторые отклонения от точной формы, изображенной на рис. 15.1. Я думаю, что если начать с логичного макета, то, скорее всего, придешь к конечной конфигурации, одновременно логичной и экономичной. Более того, на каждом этапе видно, чем жертвуешь в плане логичности во имя экономичности, и наоборот. Чтобы получить представление о количестве необходимых микрокоманд, я сконструировал микропрограмму для простой машины со следующими командами: сложение, вычитание, умножение (две команды, одна для множителя, другая для множимого), сдвиг вправо и влево (на любое число разрядов), передача из аккумулятора в память, условная операция, зависящая от знака числа в аккумуляторе, условная операция, зависящая от знака числа в B-регистре (предполагается, что есть один B-регистр), ввод и вывод. Микропрограмма также предусматривает предварительное извлечение команды из памяти (стадия 1 в терминологии EDSAC). Для выполнения всех этих операций требуется всего 40 микрокоманд.

При написании микропрограммы и обычной программы используются сходные соображения. Таким образом, окончательные детали устройства управления улаживаются применением систематического процесса, а не с помощью спонтанных процедур, основанных на блок-схемах, как это обычно происходит. Конечно, для проектирования дерева декодирования и матриц, которые можно было бы использовать для любой требуемой микропрограммы путем подходящего расположения выпрямителей, необходим основательный инженерный подход. Важное преимущество этого метода проектирования управления заключается в том, что код команд необязательно фиксировать вплоть до поздних этапов конструирования машины; его даже можно изменить после сдачи машины в эксплуатацию – достаточно перепаять матрицы.

16 Обучение компьютера (1952)

Грейс Мюррей Хоппер

Грейс Мюррей Хоппер (1906–1992) сделала замечательную профессиональную карьеру, которая увековечена как в названии национального собрания женщин, специализирующихся в информатике (ежегодная конференция имени Грейс Мюррей Хоппер), так и в названии военного корабля ВМФ США (управляемый ракетный эсминец USS Hopper). В 1934 году она получила докторскую степень по математике в Йельском университете и до начала Второй мировой войны была профессором математики в Вассарском колледже, где сама получила степень бакалавра. Примерно в 1940 году она пыталась поступить на флот, но получила отказ по возрасту. В 1943 году она наконец была зачислена в резерв ВМС, несмотря на то что ей недоставало 15 фунтов до нижней границы веса (120 фунтов, примерно 54 кг). Ее направили в лабораторию вычислительных методов Говарда Эйкена в Гарварде, где она занималась программированием Mark I и последовавшей за ним машины Mark II. Она знаменита еще и тем, что вклеила в журнал моль, ставшую причиной отказа реле. Ее ироническое замечание «Первый реальный случай найденного жучка (bug)» говорит о том, что не она придумала термин «bug», на жаргоне инженеров означающий машинную ошибку. Важнее, однако, то, что она, как и Эйкен, поняла, что будущее компьютеров связано с обработкой деловых данных не в меньшей степени, чем с научными расчетами.

В 1949 году Хоппер поступила на работу в компанию Eckert–Mauchly Computer Corporation, которая занималась коммерциализацией проекта, родившегося в стенах Электротехнической школы Мура (см. стр. 128). Эта компания была приобретена Remington Rand, куда и перешла Хоппер; машина, которую производила компания, получила название Univac. (После нескольких слияний и поглощений компания стала называться UNISYS.)

В 1940-х годах еще не было ни языков высокого уровня, ни синтаксических анализаторов, ни даже сколько-нибудь внятного символического представления машинного кода. Не было ни компиляторов, ни средств отладки. Весь процесс преобразования алгоритма в работающий код был ручным и выполнялся карандашом на бумаге вплоть до самого последнего шага – ввода программы. Хоппер стоит у истоков процесса привлечения компьютеров в помощь программистам. Эта статья содержит косвенные указания не только на «компиляторы» (включая и то, что сегодня мы называем компонующими загрузчиками и перемещением кода), но и на символическое программирование (она предвидит, что компьютеры будут вычислять производные в символическом виде), компромиссы на этапе оптимизации кода, глобальный анализ программы («однократный просмотр компьютерной информации с целью изучения ее структуры»), язык макроассемблера (который здесь называется «многоадресным кодом»), формальную спецификацию подпрограмм, иерархическую организацию программы, перенос фокуса с численных алгоритмов, которые были так важны во время войны, на коммерческие приложения и осознание того факта, что программное обеспечение в перспективе будет стоить гораздо дороже оборудования.

Все эти элементы есть в этой статье, но изложены не напрямую. Статья нарочито антропоморфна. Нелегко сказать, где на каждом этапе расширения и обобщения проходит граница между человеком и машиной, потому что Хоппер предчувствует, что эта граница будет со временем сдвигаться – вместе с успехами в «обучении» компьютера. Через пару лет она заняла должность начальника отдела «автоматического программирования» и возглавила разработку первых языков программирования ARITH-MATIC и MATH-MATIC (Ash et al. 1957), которые приблизительно соответствуют рутинам[1] типа A и типа B в этой статье.

Эти языки были ориентированы на Univac. Но Хоппер на этом не остановилась и стала движущей силой разработки и стандартизации языка Cobol (COmmon Business Oriented Language), несмотря на скептическое отношение к тому, что компьютеры можно заставить производить манипуляции с данными, описываемые на языке, напоминающем английский, и что выигрыш от эффективности труда программистов сможет намного перевесить проигрыш в скорости выполнения. Всю жизнь ее сопровождала репутация упрямого неординарного мыслителя, использующего яркие метафоры для иллюстрации своей точки зрения. Однажды на публичной лекции она продемонстрировала, что такое «наносекунда», на примере куска медного провода длиной в один фут. А над ее рабочим столом висели часы с зеркальным циферблатом и стрелками, идущими в обратном направлении, как намек на то, что соглашения на то и существуют, чтобы их нарушать.

Хоппер сделала выдающуюся карьеру на флоте, дослужившись до звания контр-адмирала. Я сам помню один факт, красноречиво свидетельствующий о препятствиях, с которыми она сталкивалась на каждом этапе своей карье-

---

[1] Под «рутиной» Хоппер понимает то, что сегодня мы называем «инструментальным средством» или «утилитой», а под «программой» – то, что написано прикладным программистом для решения конкретной задачи. – *Прим. перев.*

ры. В 1970-х годах, спустя много лет с тех пор, как новобранцем-резервистом трудилась здесь над Mark I, она в военном мундире возвращалась в Гарвард. И пребывала в очень мрачном настроении. Команда самолета, на котором она летела в Бостон, обращалась с миниатюрной седовласой дамой в форме адмирала флота с большим почтением – как со стюардессой в отставке!

Идея о механизации труда математика не нова – в отличие от ее материализации. Она восходит к абаку и проходит через умы Паскаля, Лейбница и Бэббиджа. А если говорит о более близком к нам времени, то изложенные здесь идеи принадлежат Говарду Х. Эйкену из Гарвардского университета, Джону У. Мочли из компании Eckert–Mauchly и М. В. Уилксу из Кембриджского университета. От Эйкена в 1946-м пошла идея библиотеки подпрограмм, описанной в руководстве по Mark I, и идеи, воплощенные в машине Mark III; от Мочли – основные принципы языка «Short code», а также предложения, критические замечания и неустанное терпение, с которым он выслушивал черновые наброски этой статьи; от Уилкса – величайшая из всех возможных подмога – книга по данному предмету. За те их идеи, которые включены в этот текст, я со всей серьезностью выражаю свою благодарность и остаюсь в неоплатном долгу перед ними.

16.1. Введение

Для начала на рис. 16.1 представлена конфигурация элементов, необходимых операции: ввод; средства управления, даже если это только запуск и останов, ранее подготовленные инструменты, предлагаемые для выполнения операции; и вывод, который может, в свою очередь, служить вводом для другой операции. Это базовые элементы любой производственной линии; подача на вход сырья, управление со стороны человека, возможно, с помощью каких-то инструментов; применение машинных средств обработки; выход операции, например автомобиль, рельс или банка консервированных томатов.

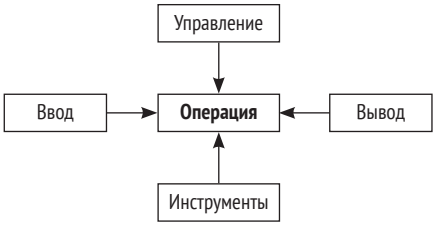


Рис. 16.1. Операция

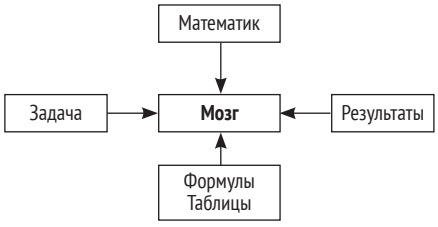


Рис. 16.2. Решение задачи

Вооруженные силы, правительство и промышленность заинтересованы не только в создании новых операций для производства новых результатов, но и в повышении эффективности существующих операций. Очень старой

операцией является решение математической задачи (рис. 16.2). Она согласуется с конфигурацией типичной операции: ввод математических данных; управление со стороны математика; инструменты в виде памяти, формул, таблиц, карандаша и бумаги; мозг, который выполняет арифметические операции и производит результаты.

В настоящее время целью является замена в той мере, в какой это возможно, человеческого мозга электронным цифровым компьютером. То, что такие компьютеры сами укладываются в описанную выше схему, видно по рис. 16.3. (С вашего позволения я буду использовать Univac как синоним электронного цифрового компьютера – в основном потому, что так думаю, но также и потому, что это удобно.)

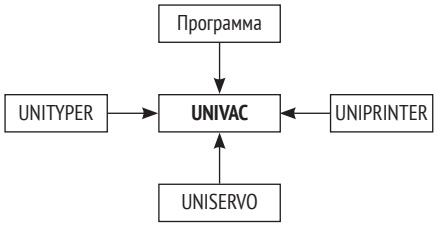


Рис. 16.3. Система UNIVAC.
[Примечание редактора: UNITYPER – машинописное устройство ввода, UNISERVO – накопитель на магнитной ленте.]

На рис. 16.4 показано решение задачи в виде двухуровневой операции, получающееся в результате соединения конфигураций человека и электронного компьютера. Скучные арифметические действия отобраны у математика, превратившегося в программиста, эта рутина возложена на Univac. Инструменты программиста дополнены «кодом», в который он транслирует свои инструкции компьютеру. «Стандартные знания», заложенные в Univac его создателями, состоят из элементарной арифметики и логики.

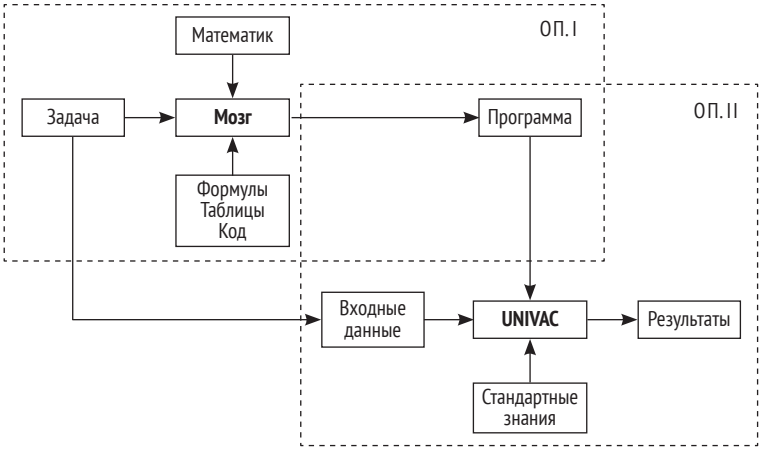


Рис. 16.4. Решение задачи

Эта ситуация останется статичной, пока новизна придумывания программ не потускнеет и не превратится в унылый труд по написанию и проверке программ. Сейчас этот труд видится как обязанность, возлагаемая на человеческий мозг. Кроме того, после оплаты компьютера финансовые и временные затраты на программирование становятся предметом внимания вице-президентов и директоров проектов. Здравый смысл диктует добавление третьего уровня операции (рис. 16.5).

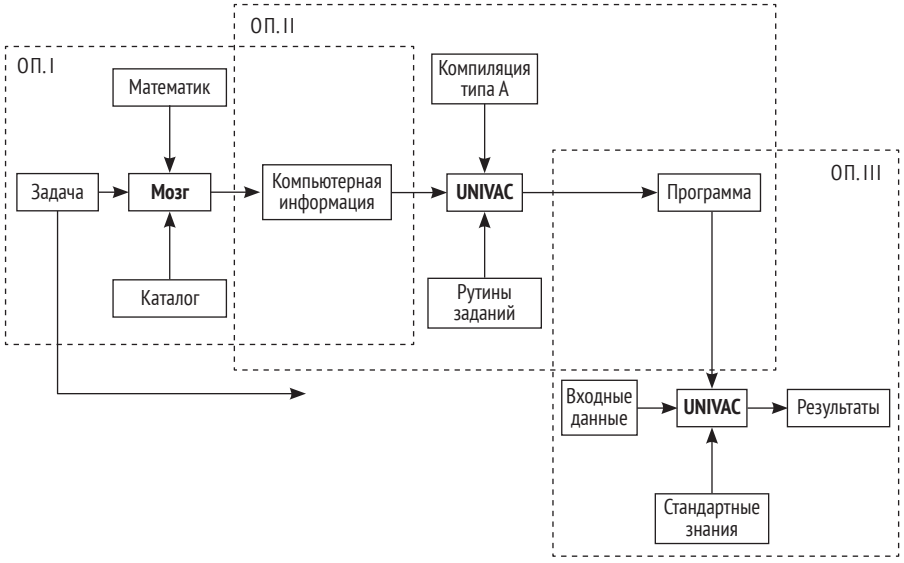


Рис. 16.5. Компиляция программ и подпрограмм

Программист может вернуться к своей математической ипостаси. Ему предлагается каталог подпрограмм. Ему больше не нужно иметь под рукой формулы или таблицы элементарных функций. Ему даже не нужно знать, какие конкретно инструкции использует компьютер. Ему нужно только уметь пользоваться каталогом, чтобы сообщить компьютеру информацию о своей задаче. Univac на базе информации, предоставленной математиком, под управлением «компилирующей рутины типа А», применяя подпрограммы и свой собственный машинный код, порождает программу. Эта программа, в свою очередь, направляет Univac в процессе применения вычислений к исходным данным и порождения требуемых результатов. Достигнуто значительное сокращение временных затрат и источников ошибок. Если библиотека хорошо укомплектована, то программирование занимает часы вместо недель. Программа больше не страдает от ошибок транскрипции и от ошибок в непротестированных процедурах.

Описания компьютерной информации, каталога, компилирующих рутин и подпрограмм будут приведены после добавления еще одного уровня в блок-схему. Согласно рис. 16.5 математик по-прежнему должен выполнять все математические операции, относящиеся к программированию Univac

и вычислительным операциям. Однако компьютерная информация, поставляемая математиком, больше не состоит из числовых величин как таковых. Теперь переменные и постоянные предстают в символической форме наряду с операциями над ними. Становится возможным добавление четвертого уровня операций, показанного на рис. 16.6. Предположим, к примеру, что математик хочет вычислить функцию и ее первые *n* производных. Он отправляет Univac'у информацию, определяющую саму функцию. Под управлением «компилирующей рутины типа B», в данном случае дифференциатора, используя рутины заданий, Univac поставляет информацию, необходимую программе для вычисления функции и ее производных. Из формулы функции Univac выводит формулы ее последовательных производных. Эта информация, обрабатываемая компилирующей рутиной типа A, порождает программу, которая направляет вычисления.

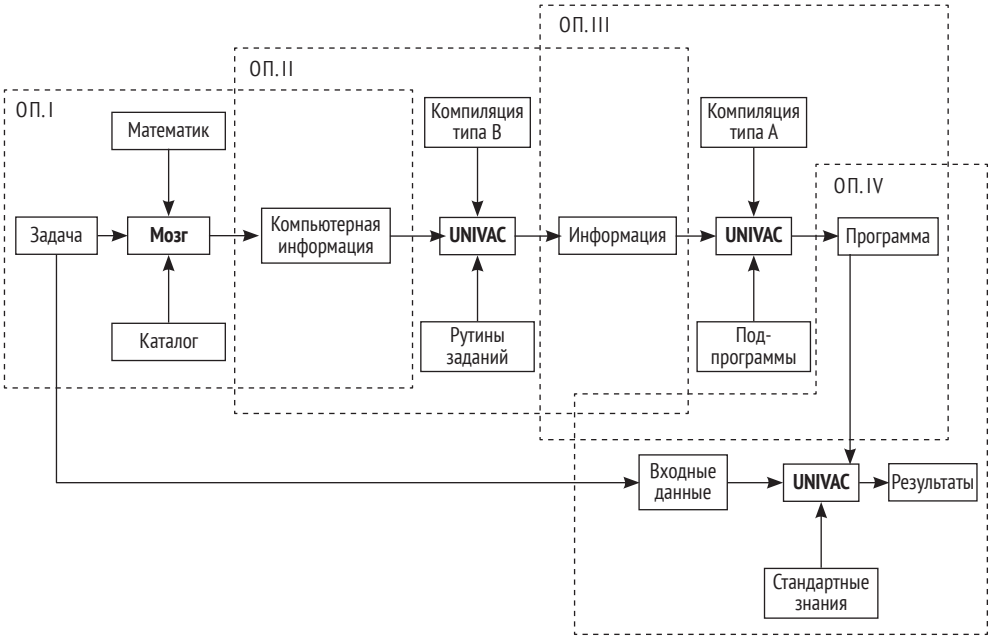


Рис. 16.6. Компиляция типа B и рутины заданий

В развернутом виде эта процедура кажется длинной и запутанной. Но это не так. На рис. 16.7 представлена более точная картина вычислительной системы, получающаяся после возврата к двухкомпонентной системе, состоящей из математика и компьютера.

В предположении, что код, программа, входные данные и результаты – знакомые термины, остается определить и специфицировать формы информации и рутин, приемлемые для этой системы. К ним относятся: каталог, компьютерная информация, подпрограмма, компилирующие рутины, типы A и B и рутины заданий.

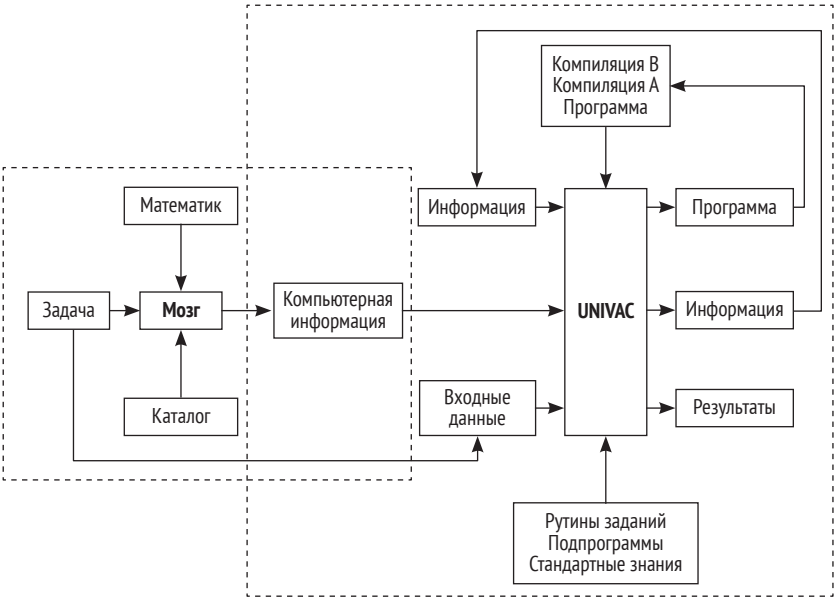


Рис. 16.7. Вычислительная система

Коль скоро поставлена цель использовать подпрограммы, на ум приходят два метода. В первом программа обращается к непосредственно доступной подпрограмме, использует ее и продолжает вычисление. При ограниченном числе подпрограмм этот метод практически осуществим и полезен. Такая система была разработана под названием «short-order code» сотрудниками лаборатории вычислительных методов [Примечание редактора: в компании Eckert–Mauchly Computer Corporation.].

Второй метод предполагает не только поиск подпрограммы, но также ее трансляцию, надлежащую корректировку и включение в программу. Тогда законченную программу можно исполнить как автономный блок в любой момент, и ее саму можно поместить в библиотеку в качестве более продвинутой подпрограммы.

16.2. Каталог и компьютерная информация

Каждую задачу можно свести к уровню имеющихся подпрограмм. Рассмотрим простую задачу: вычислить $y = e^{-x^2}\sin cx$ с применением элементарных подпрограмм. Каждый шаг формулы согласуется со схемой операции на рис. 16.8, а именно: $u = x^2$; $U = e^{-u}$, $v = cx$, $V = \sin v$, $y = UV$. Однако, как показано на рис. 16.9, эта информация еще недостаточно стандартизована, чтобы ее можно было подать компилирующей программе. Необходимо рассмотреть несколько проблем и определить несколько процедур. [Примечание редактора: здесь Хоппер идет по стопам Лавлейс – сравните с рис. 3.1 на стр. 40.]

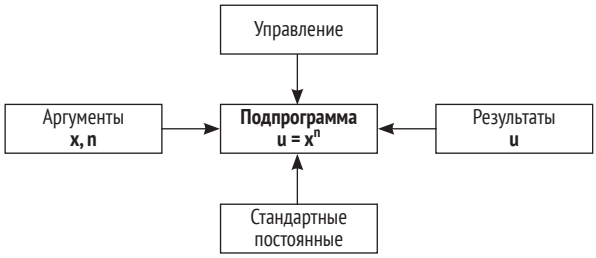


Рис. 16.8. Операция

$y = e^{-x^2}\sin cx$				
Номер операции	Операция	Аргументы	Результаты	Управление
0	TRANSFER b0i	0, 0.01, 0.99, 2, 5 I (1, 2, 3, 4, 5)	$x, \Delta x, L_x, n, c$ 1, 2, 3, 10, 6	
1	x^n apn	x, n 1, 10	$u = x^2$ 4	
2	e^{-u} x-e	u 4	$U = e^{-u}$ 5	
3	$c^{\otimes}$ amc	c, x 6, 1	$v = cx$ 7	
4	$\sin v$ ts0	v 7	$V = \sin v$ 8	
5	$\otimes$ am0	U, V 5, 8	$y = UV$ 9	
6	EDIT yrs	x, y 1, 9	$\bar{x}; \bar{y}$ O (1, 2)	
7	$\otimes \to L$ aaL	$x, \Delta x, L_x$ 1, 2, 3	$x + \Delta x \to x$ 1	$x < L_x \to 1; x \geqslant L_x \to 8$ 8, 1
8	STOP ust			

Рис. 16.9. Пример
[Примечание редактора: aaL = «add to a limit» (Ash et al. 1957, стр. 86)]

Операции пронумерованы в обычном порядке, и этот номер становится частью компьютерной информации. Таким образом, когда требуется изменить обычную последовательность, легко определить альтернативное место назначения. Компилирующая рутина транслирует номера операций в инструкции закодированной программы. Возможны две существенно различные ситуации: альтернативное место назначения предшествует рассматриваемой операции или следует за ней, обходя несколько операций. В обоих случаях необходимо лишь, чтобы компилирующая рутина запомнила, куда поместила каждую подпрограмму или что была указана передача управления операции k. В любом случае, математику нужно только сказать «перейти к операции k», и компилирующая рутина сделает все остальное.

Далее нас будут интересовать символы, используемые для аргументов и результатов, а также для операций. Один математик может написать $y = e^{-x^2}\sin cx$, а другой – $u = e^{-v^2}\sin gv$. Очевидное решение оказывается лучшим.

1	x	x_1
2	x	x_2
3	L_x	x_3
4	u	x_4
5	U	x_5
6	c	x_6
7	v	x_7
8	V	x_8
9	y	x_9
10	n	x_{10}

Рис. 16.10. Таблица переменных для рис. 16.9

Создадим список аргументов и результатов и пронумеруем их. (Это сводится к записи всех констант и переменных в виде x_i.) Порядок не важен, поэтому величины, о которых мы позабыли, можно добавить в конец (рис. 16.10).

В качестве символов для операций и подпрограмм используется система «номеров вызовов». Эти буквенные символы представляют класс подпрограмм. Следуя примеру д-ра Уилкса, мы выбрали по возможности фонетические символы: `a` = арифметический (arithmetic), `t` = тригонометрический (trigonometric), `x` = экспоненциальный (exponential), `amc` = арифметический, умножение на константу (arithmetic, multiplication by a constant), `x-e` = e^{-U}; `ts0` = тригонометрический, синус. В сочетании с номерами вызовов `n`, `f`, `s` означает соответственно нормальное, с плавающей точкой или с фиксированной десятичной точкой. Другие буквы и цифры обозначают радианы или градусы для углов, комплексные числа и т. д. Эти номера вызовов перечислены в каталоге вместе с порядком задания аргументов, управляющих конструкций и результатов. <...>

16.3. Подпрограммы

Каждая подпрограмма в библиотеке при кодировании записывается относительно своей строки входа, которой приписывается номер 001. Вообще говоря, при программировании и кодировании ставится цель добиться максимальной точности и минимального времени вычислений. Их собственные константы могут храниться внутри самих подпрограмм. Они также могут пользоваться «постоянно хранимыми константами», которые считываются каждой программой. Эти постоянно хранимые константы занимают зарезервированную область памяти, а для обращения к ним используются буквенные обозначения ячеек памяти; трюк, который в настоящее время специфичен для UNIVAC. Таким образом, эти адреса не модифицируются в ходе размещения подпрограммы в памяти. К ним относятся такие величины, как $1/2\pi$, $\pi/4$, $\log_{10}e$, ± 0, 0.2, 0.5 и т. д.

Каждой подпрограмме предшествует некоторая информация, предоставленная математиком и дополнительная:

1) номер вызова;
2) аргументы, место назначения аргументов в подпрограмме, выраженное в относительном коде подпрограмм;
3) индикаторы запрета модификации, указывающие на встроенные в подпрограмму константы, которые не должны изменяться;
4) результаты, положения результатов внутри подпрограммы, выраженные в относительном коде.

Каждая подпрограмма организована стандартным образом.

Строка входа – первая строка подпрограммы называется строкой входа, т. е. в относительном коде она имеет номер 1. Это первая строка подпрограммы, передаваемая программе, она содержит инструкцию, передающую управление первой строке действия.

Строки выхода – вторая строка подпрограммы является ее строкой нормального выхода. Она содержит инструкцию, передающую управление строке, следующей за последней строкой подпрограммы. Если только не требуется альтернативный способ передачи управления, то все выходы из подпрограммы ссылаются на строку нормального выхода. Строки альтернативного выхода, подразумевающие отход управления от обычной последовательности, следуют за строкой нормального выхода в предопределенном порядке, как записано в каталоге.

Аргументы – за строками выхода следуют пустые места, зарезервированные для аргументов, выстроенных в предопределенном порядке.

Результаты – результаты, также в заранее заданном порядке, следуют за аргументами.

Константы – за результатами, если это необходимо, могут следовать произвольные константы, специфичные для данной подпрограммы. Если подпрограмма была составлена из других подпрограмм, то в нее могут также быть встроены группы констант. Они отражены в информации о запрете модификации.

Далее в подпрограмме расположена *первая строка действия*. Ее положение в относительном коде указано в строке входа. Ни одна строка инструкции не может находиться раньше этой строки.

Порядок следования строк входа и выхода, аргументов, результатов и констант произволен. Это удобно. Нужно лишь, чтобы этот порядок был фиксирован и чтобы компьютер его понимал.

Для удобства манипулирования ряд элементарных подпрограмм объединен в частичную библиотеку. <...> По мере добавления подпрограмм в библиотеку она становится более полезной, и время программирования еще уменьшается.

Возможно, настанет день, когда элементарные подпрограммы будут использоваться редко, а компьютерная информация будет содержать всего семь-восемь вызовов мощных подпрограмм.

16.4. Построение подпрограмм

Разрешать неопытному программисту вмешиваться в код подпрограмм необязательно и даже нежелательно. Обычно такой код должен работать как можно быстрее, и в нем используются все приемы ремесла, известные опытному программисту. Его работа была протестирована на компьютере. <...>

Компилирующие рутины типа А предназначены для выбора и организации подпрограмм в соответствии с информацией, предоставленной математиком или компьютером. Вообще говоря, существует всего одна рутина типа А. Однако поскольку код для UNIVAC содержит инструкции, передающие одновре-

менно две соседние величины, была спроектирована вторая компилирующая рутина, отвечающая за десятичные числа с плавающей точкой, комплексные числа и операции с двойной точностью. Для каждой из операций, перечисленных математиком, рутина типа А выполняет следующие действия:

1. Найти подпрограмму по ее номеру вызова.
2. Опираясь на компьютерную информацию и на информацию в подпрограмме, объединенные с ее знаниями о программе, сформировать и ввести в программу инструкции, передающие аргументы из рабочей памяти в подпрограммы.
3. Скорректировать строки входа и нормального выхода в соответствии с позицией подпрограммы в программе и ввести их в программу.
4. В соответствии с управляющей информацией, предоставленной программистом, скорректировать строки альтернативного выхода и ввести их в программу (этот процесс включает ссылку на запись).
5. В соответствии с управляющей информацией, предоставленной предшествующими операциями, скорректировать вспомогательные строки входа и ввести их в программу.
6. Модифицировать все адреса в инструкциях подпрограммы и ввести эти инструкции в программу.
7. В соответствии с информацией, предоставленной подпрограммой, оставить неизменными все встроенные в подпрограмму константы и передать их программе.
8. Опираясь на компьютерную информацию и на информацию в подпрограмме, сформировать и ввести в программу инструкции, передающие результаты в [Примечание редактора: текст отсутствует, возможно, имелось в виду «рабочую память».].
9. Создать запись о программе, включающую номера вызовов всех подпрограмм и положения их строк входа в программе.

Компилирующие рутины содержат также некоторые инструкции, касающиеся входных лент, библиотеки лент и лент с программами, специфичные для Univac. Все расчетные операции, например выделение временной памяти и места для программы, а также управление вводом и выводом, выполняются компилирующей рутиной. Говоря без прикрас, компилирующая рутина – это программист, и выполняет она действия, необходимые для изготовления готовой программы.

Компилирующие рутины типа В для каждой операции посредством «рутин заданий» заменяют или дополняют имеющуюся компьютерную информацию новой. Так, компилирующая рутина В-1 для каждой операции копирует информацию, относящуюся к этой операции, и обращается к соответствующей рутине задания. Рутина задания генерирует формулу и выводит информацию, необходимую для вычисления производной операции. Затем компилирующая рутина В-1 записывает эту информацию в форме, пригодной для передачи рутине типа А.

Поскольку информация может быть повторно подана рутине типа В, очевидно, что для получения программы, вычисляющей $f(x)$ и ее первые n производных, нужно задать только информацию, определяющую $f(x)$, и значение

n. Формулы для производных *f*(*x*) будут выведены повторным применением B-1 и запрограммированы рутиной типа A.

Здесь самое подходящее место для ответа на вопрос, касающийся приязни или неприязни к подпрограммам. Поскольку описанное выше использование подпрограмм расширяет возможности компьютера, вопрос становится бессмысленным и превращается в вопрос о том, как быстрее изготавливать подпрограммы лучшего качества. Однако, подводя итоги преимуществам и недостаткам использования подпрограмм, мы отнесем к преимуществам следующее:

1. Передача Univac'у механических работ, таких как выделение памяти, модификация адреса и транскрипция.
2. Исключение таких источников ошибок, как ошибки программирования и транскрипции.
3. Экономия времени на программирование.
4. Способность производить действия с операциями.
5. Избегание дублирования усилий, потому что каждая программа может, в свою очередь, стать подпрограммой.

Из недостатков в глаза бросаются только два. Вследствие стандартизации небольшое время теряется на выполнение дублирующих передач данных, которых можно было бы избежать при кодировании вручную. В задачах с базовой загрузкой это может стать серьезной проблемой. Но даже в этом случае имеет смысл поручить Univac'у изготовление оригинальной программы, а затем исключить дублирование перед повторным запуском задачи. Второй недостаток недолго будет оставаться серьезным. Дело в том, что если нужной подпрограммы не существует, то ее необходимо запрограммировать и добавить в библиотеку. Скорее всего, такое будет случаться для подпрограмм редактирования ввода и вывода, пока не будет накоплено их достаточное количество. Эта ситуация также подчеркивает необходимость максимальной общности при построении подпрограмм.

Можно выделить несколько направлений развития в этой области. <...>

Будет предложено больше компилирующих рутин типа A: для обработки коммерческих, а не математических программ; некоторые компилирующие рутины специального назначения, например умеющие по ходу дела вычислять приближенные величины и соответственно выбирать подпрограммы. Компилирующей рутине можно сообщить среднее время работы каждой подпрограммы, так чтобы она могла оценить время работы всей программы. Например, если в одной программе вызываются sin θ и cos θ, то быстрее будет вычислять их одновременно. Это повлечет за собой однократный просмотр компьютерной информации с целью изучения ее структуры.

Рутины типа B в настоящее время включают линейные операторы. Необходимо разработать больше рутин типа B. Едва ли можно отрицать, что найдется место для рутин типа C и D, добавляющих более высокие уровни операций. Уже ведутся работы по порождению формул, выработанных рутинами типа B, в алгебраической форме в дополнение к порождению вычислительных программ для них.

Таким образом, если рассматривать профессионального программиста (не математика) как неотъемлемую часть компьютера, то становится очевидным, что память программиста и вся информация и данные, на которые он может сослаться, доступны компьютеру, при условии что будут переведены на подходящий язык. И столь же очевидно, что компьютер вполне в состоянии запомнить инструкции, введенные в него программистом, и действовать в соответствии с ними.

Принимая во внимание специальные знания по некоторым продвинутым темам, Univac уже сейчас обладает математическими знаниями, эквивалентными студенту второго курса колледжа, и при этом он ничего не забывает и не делает ошибок. Есть надежда, что в ближайшее время он пройдет весь курс и будет готов к приемным экзаменам в аспирантуру.

17 О кратчайшем остовном поддереве графа и о задаче коммивояжера (1956)

Джозеф Б. Крускал мл.

Эта статья математика Джозефа Крускала (1928–2010) знаменует важный шаг на длинном пути от исследования операций – дисциплины, расцвет которой пришелся на 1940-е годы в ответ на необходимость решения разнообразных коммерческих, военных и логистических задач, – к компьютерной науке, которую некоторые определяют как изучение алгоритмов. Конкретный алгоритм, представленный здесь как построение A и ныне известный как алгоритм Крускала, находит остовное дерево минимальной стоимости в неориентированном графе путем построения леса, для чего ребра добавляются по одному, от самого короткого до самого длинного, исключая ребра, которые привели бы к образованию цикла. В работе Borůvka (1926), на которую ссылается автор в первом же предложении, описан, по-видимому, самый первый алгоритм решения этой задачи, восходящий еще к тем временам, когда основная терминология теории графов еще не была формализована. В той работе чешский математик Отакар Борувка решил задачу нахождения самой эффективной энергосистемы для Моравии, сегодня являющейся частью Чешской республики.

Современных читателей стиль Крускала удивит своей ясностью – описание и доказательство правильности алгоритма так же лаконичны и элегантны, как в любом из более поздних изложений, – и полным отсутствием какого бы то ни было анализа. Крускал описывает свой алгоритм как «практичный»,

не намекнув, что бы это могло значить в терминах времени работы или еще какой-то формы сложности. Он даже не говорит, какие параметры оказывают наибольшее влияние на поведение алгоритма.

Осмысление термина «практичный» ведет к примечательной истории анализа алгоритмов. Понадобилось еще десять лет, прежде чем Эдмондс и Кобхэм ввели понятие полиномиального времени работы как грубого приближения к интуитивному понятию практичности (см. стр. 418). А затем Дональд Кнут поднял планку, введя в обиход более тонкий анализ алгоритмов (см. главу 43). С учетом времени извлечения ребер по порядку с применением наивных методов временная сложность алгоритма Крускала оказывается равной $O(m \log n)$, где m – число ребер, а n – число вершин. Но это не конец истории. Ученик Кнута Роберт Тарьян (Tarjan 1975) провел замечательный анализ известного простого алгоритма реализации системы непересекающихся множеств. С использованием этих структур данных сложность алгоритма Крускала оказывается равной $O(m\alpha(n))$ при некоторых предположениях, которые обычно выполняются. Здесь $\alpha(n)$ – обратная функция Аккермана – неограниченная функция, которая растет так медленно, что $\alpha(n) < 5$ для всех n, не превышающих число элементарных частиц во Вселенной. В общем можно считать, что время работы хорошо реализованного алгоритма Крускала пропорционально числу ребер.

Крускал отмечает элегантное сходство между задачей о минимальном остовном дереве и задачей коммивояжера, не предприняв никаких дополнительных усилий по нахождению эффективного решения последней. Кардинальное различие между этими задачами не было прояснено до начала 1970-х годов, когда Карп показал, что задача коммивояжера является $\mathcal{NP}$-полной. (Неориентированный гамильтонов контур, стр. 443, – частный случай задачи коммивояжера, в котором все расстояния между соседними вершинами равны 1.)

Другой известный алгоритм нахождения минимального остовного дерева – алгоритма Прима, названный в честь его автора, Роберта Клэя Прима (Prim 1957), на самом деле является частным случаем построения B Крускала, в котором множество, названное им V, включает только одну вершину. Этот алгоритм строит одно дерево, добавляя ребра, смежные с существующим, начиная с самого короткого (и пропуская ребра, которые создали бы цикл), – как в построении A, примере того, что теперь мы называем жадной стратегией. Алгоритм Прима был заново открыт Эдсгером Дейкстрой (Dijkstra 1959) в процессе разработки его собственного алгоритма решения задачи о кратчайших путях в графе, хотя в действительности все эти алгоритмы – Прима, построение B Крускала и Дейкстры – в значительно более ранней работе предвосхитил еще один чех, Войцех Ярник (Jarník 1930). (Крускал включил также третий жадный алгоритм A′, у которого нет общепринятого названия.) Алгоритм транзитивного замыкания Уоршелла (Warshall 1962) и алгоритм нахождения кратчайших путей между всеми парами вершин Флойда (Floyd 1962) – примеры динамического программирования, еще одного шага на пути от исследования операций к компьютерной науке.

Джозеф Крускал был членом замечательной семьи математиков. Его отец, еврей, эмигрировавший из Прибалтики в Нью-Йорк в 1892-м, стал успешным

оптовым торговцем мехами, мать первой познакомила Америку с оригами. Самый старший брат Джозефа, Уильям, стал профессором статистики в Чикагском университете, а второй брат, Мартин Дэвид, – профессором астрофизики и прикладной математики в Принстоне. Его племянник, Клайд Крускал, работает профессором компьютерных наук в Мэрилендском университете. Джозеф написал эту работу в бытность свою аспирантом в Принстоне, где в 1954 году защитил докторскую диссертацию, а опубликовал он ее, когда работал в Принстоне, при содействии Управления научно-исследовательских работ ВМФ США. В 1959-м он был принят на работу в компанию Bell Labs, где прошла большая часть его карьеры. Он автор важных работ в области статистики и статистической лингвистики, а также в той области, которую сейчас мы называем информатикой.

Несколько лет назад вышел и привлек интерес письменный перевод (с малоизвестного языка) работы Borůvka (1926). Эта статья посвящена следующей теореме: если каждому ребру (конечного) связного графа сопоставлено положительное вещественное число (*длина* ребра) и если все эти длины различны, то среди остовных деревьев (нем. Gerüst) графа имеется ровно одно, для которого сумма длин ребер минимальна; иначе говоря, кратчайшее остовное дерево графа единственно. (Подграф называется остовом графа, если он содержит все вершины графа. На самом деле в работе Borůvka [1926] эта теорема сформулирована и доказана в терминах «матрицы длин» графа, т. е. матрицы $||a_{ij}||$, где a_{ij} – длина ребра, соединяющего вершины i и j. Конечно, предполагается, что $a_{ij} = a_{ji}$ и что $a_{ii} = 0$ для всех i и j.)

Доказательство в работе Borůvka (1926) основано на не таком уж неразумном методе построения остовного поддерева минимальной длины. Именно это построение вызывает наибольший интерес, поскольку дает решение задачи (Задача 1 ниже), которая тесно связана с одним из вариантов (Задача 2 ниже) хорошо известной задачи коммивояжера.

Задача 1. Предложить практический метод построения остовного поддерева наименьшей длины.

Задача 2. Предложить практический метод построения неразветвленного остовного поддерева наименьшей длины.

Построение в работе Borůvka (1926) излишне запутанное. В этой статье я приведу несколько более простых построений, решающих Задачу 1, и покажу, как одно из этих построений можно использовать для доказательства теоремы из работы Borůvka (1926). По-видимому, любое построение, решающее Задачу 1, годится для доказательства этой теоремы.

Прежде всего хочу отметить, что без ограничения общности можно предположить, что заданный связный граф G полный, т. е. любая пара вершин соединена ребром. Ибо если какое-то ребро G «отсутствует», то можно вставить ребро очень большой длины, и при этом ни в каком интересующем нас отношении граф не изменится. Кроме того, возможно и интуитивно понятно считать, что отсутствующие ребра имеют бесконечную длину.

Построение А. Выполнять следующий шаг столько раз, сколько возможно: из до сих пор не выбранных ребер G выбрать самое короткое, не соз-

дающее циклов в совокупности с ранее выбранными ребрами. Очевидно, что множество ребер, которые будут выбраны в конечном итоге, должно образовывать остовное дерево G, и на самом деле это остовное дерево будет кратчайшим.

Построение B. Пусть V – произвольное, но фиксированное (непустое) подмножество вершин G. Тогда выполнять следующий шаг столько раз, сколько возможно: из тех ребер G, которые до сих пор не выбраны, но соединены либо с вершиной из V, либо с уже выбранным ребром, выбрать самое короткое ребро, не создающее циклов в совокупности с ранее выбранными ребрами. Очевидно, что множество ребер, которые будут выбраны в конечном итоге, образует остовное дерево G, и на самом деле это остовное дерево будет кратчайшим. В случае когда V – множество всех вершин G, построение B сводится к построению A.

Построение A′. Этот метод в некотором смысле двойственен A. Выполнять следующий шаг столько раз, сколько возможно: из множества до сих пор не выбранных ребер G выбрать самое длинное, удаление которого не приводит к потере связности этого множества. Очевидно, что множество ребер, которые не будут выбраны в конечном итоге, образует остовное дерево G, и на самом деле это остовное дерево будет кратчайшим. Мне не ясно, имеется ли для построения B подобное двойственное в общем случае.

Прежде чем показать, как можно использовать построение A для доказательства теоремы из работы Borůvka (1926), удобно сформулировать в виде теоремы ряд элементарных фактов из теории графов. Читатель с легкостью убедится, что все они истинны. По эстетическим соображениям, я включил гораздо больше, чем мне реально понадобится.

Предварительная теорема. *Если G – связный граф с n вершинами и T – подграф G, то следующие условия эквивалентны:*

(a) T – остовное дерево G;
(b) T – максимальный лес в G;
(c) T – минимальный связный остовный граф G;
(d) T – лес, содержащий $n - 1$ ребер;
(e) T – связный остовный граф, содержащий $n - 1$ ребер.

(Граф называется «максимальным», если он не содержится ни в каком большем графе такого же вида; он называется «минимальным», если не содержит никакого меньшего графа такого же вида. «Лесом» называется граф, не содержащий циклов.) Подлежащая доказательству теорема утверждает, что если длины всех ребер G различны, то кратчайшее остовное дерево G, назовем его T, единственно. Очевидно, что T можно также определить как любой кратчайший лес, содержащий $n - 1$ ребер.

В построении A обозначим $a_1, \ldots, a_{n-1}$ выбранные ребра в том порядке, в каком они выбирались. Пусть A_i – лес, состоящий из ребер от a_1 до a_i. Будет доказано, что $T = A_{n-1}$. Из предположения о том, что ребра G имеют разную длину, легко видеть, что построение A однозначно определено. Таким образом, все A_i единственные, а вместе с ними и T.

Остается доказать, что $T = A_{n-1}$. Если $T \neq A_{n-1}$, то обозначим a_i первое ребро A_{n-1}, не принадлежащее T. Тогда $a_1, \ldots, a_{i-1}$ принадлежат T. Граф $T \cup a_i$ должен иметь ровно один цикл, который обязан содержать a_i. Этот цикл должен также содержать некоторое ребро e, не принадлежащее A_{n-1}. Тогда $T \cup a_i - e$ – лес с $n - 1$ ребрами.

Поскольку $A_{i-1} \cup e$ содержится в вышеупомянутом лесе, то оно само является лесом, поэтому из построения A следует, что

$$\text{length}(e) > \text{length}(a_i).$$

Но тогда $T \cup a_i - e$ короче, чем T. Это противоречит определению T и, значит, косвенно доказывает, что $T = A_{n-1}$.

18 Перцептрон: вероятностная модель хранения и организации информации (1958)

Фрэнк Розенблатт

«Перцептрон» Фрэнка Розенблатта (1928–1971) – это одновременно драматический ранний шаг в направлении искусственного интеллекта, временное поражение в интеллектуальном соперничестве и достойный изучения пример эксцентричной тенденции научного исследования. Получив психологическое образование в Корнеллском университете, Розенблатт начал разрабатывать механизм искусственного восприятия, когда поступил на работу в Корнеллскую аэронавигационную лабораторию после защиты докторской диссертации в 1956 году. Там он получил доступ к тому, что тогда считалось большим компьютером, IBM 704. Его первый подход к перцептрону, состоявшийся в 1957 году, был ранним экспериментом по обучению искусственной системы распознавания образов с целью улучшить качество ее работы посредством обучения с подкреплением. В этой статье 1958 года описывается уточненная модель, снабженная различными настраиваемыми параметрами, а также включена математика, обычно применяемая при описании физических систем. Розенблатт, оставаясь профессором психологии, продолжал эту линию исследований и ставил другие, совершенно различные, эксперименты в нейрофизиологии до самой своей гибели во время плавания на яхте в день, когда ему исполнилось 43 года.

Розенблатт опередил свое время, а линию, соединяющую его изобретение с обученными нейронными сетями нашего времени, прямой никак не назовешь. В 1969 году Марвин Мински, отец-основатель искусственного интеллекта в МТИ, и Сеймур Пейперт, его коллега-психолог, опубликовали книгу «Перцептроны» (Minsky and Papert 1969), в которой изучались возможности ограниченной «одноуровневой» версии модели Розенблатта. Были продемонстрированы – и совершенно правильно – ограничения одноуровневых сетей. С научной точки зрения, этого было недостаточно, чтобы отодвинуть на обочину исследования полной модели перцептрона, но на деле именно так и произошло, быть может, потому что книга появилась в момент обескураживающе медленного прогресса в получении нетривиальных экспериментальных результатов в области искусственного интеллекта. Мински (кстати, занял следующее после Розенблатта место среди выпускников Высшей научной школы в Бронксе) и его коллеги по МТИ (некоторые вскоре перешли в Стэнфорд) развивали другой подход к ИИ, на основе логики и других символических систем, а не пытаясь подражать архитектуре человеческого мозга.

Ранние исследования Розенблатта вытащили из нафталина в 1980-е годы, когда компьютеры стали достаточно мощными для моделирования многослойных сетей и их обучения на больших наборах данных. В результате нейронные вычислительные модели теперь представляют огромный интерес для исследователей искусственного интеллекта.

Если мы хотим разобраться в способности высших организмов к восприятию, распознаванию, обобщению, вспоминанию и мышлению, то первым делом должны ответить на три фундаментальных вопроса:

1. Как информация о физическом мире воспринимается, или обнаруживается, биологической системой?
2. В какой форме эта информация хранится, или запоминается?
3. Как информация, находящаяся в хранилище, или в памяти, влияет на распознавание и поведение?

Первый из этих вопросов лежит в плоскости физиологии восприятия, и это единственный вопрос, для которого достигнуто хоть какое-то понимание. В этой статье рассматриваются преимущественно второй и третий вопросы, которые до сих пор остаются предметом умозрительных рассуждений, а те немногие факты, которые предоставляет нам современная нейрофизиология, еще не объединены приемлемой теорией.

По второму вопросу есть две альтернативные точки зрения. Первая предполагает, что сенсорная информация хранится в форме закодированных представлений или образов, так что имеется некоторое взаимно однозначное соответствие между сенсорными стимулами и хранимым образом. Согласно этой гипотезе, если бы удалось понять код, или «принципиальную схему» нервной системы, то, в принципе, можно было бы точно узнать, что организм помнит, реконструировав исходные сенсорные образы по оставленным ими «следам в памяти» – точно так же, как мы проявляем фотографический негатив или транслируем на понятный нам язык комбинацию

электрических зарядов в «памяти» цифрового компьютера. Эта гипотеза подкупает своей простотой и понятностью, и на базе этой идеи закодированной памяти о представлениях было создано целое семейство теоретических моделей мозга (Culbertson, 1950, 1956; Köhler, 1951; Rashevsky, 1938). Альтернативный подход, выросший на традициях британского эмпиризма, основан на смелой догадке о том, что образы стимулов, возможно, никогда и не записываются, а центральная нервная система просто работает как очень сложная переключательная сеть, в которой запоминание принимает форму новых связей, или путей между центрами активности. Во многих недавних работах, развивающих эту точку зрения (например, «клеточные ансамбли» Хебба и «кортикальная предвосхищающая целевая реакция» Халла), «реакции», ассоциированные со стимулами, могут быть целиком заключены внутри самой ЦНС. В таком случае реакция представляет «идею», а не действие. Важная особенность этого подхода заключается в отсутствии какого-либо простого отображения стимула в память, которое можно было бы описать кодом, допускающим его последующую реконструкцию. Какая бы информация ни запоминалась, она должна храниться в виде *предпочтения определенной реакции*, т. е. информация содержится в связях или ассоциациях, а не в виде топографических представлений. (Далее в этой статье термин *реакция* означает любое различимое состояние организма, которое может включать, а может и не включать обнаруживаемую извне мышечную активность. Например, активация некоторых клеточных ядер в центральной нервной системе может, согласно этому определению, рассматриваться как реакция.)

В соответствии с этими двумя точками зрения на метод сохранения информации существует две гипотезы по поводу третьего вопроса – способа, которым хранимая информация проявляет свое влияние на текущую активность. Сторонники «теории кодированной памяти» вынуждены признавать, что распознавание любого стимула включает сопоставление или систематическое сравнение содержимого памяти с входящими сенсорными образами, чтобы определить, встречался ли текущий стимул ранее, и выбрать подходящую реакцию организма. С другой стороны, сторонники эмпирической традиции, по существу, объединили ответ на третий вопрос со своим ответом на второй: т. к. информация хранится в форме новых связей, или каналов передачи в нервной системе (или создания условий, функционально эквивалентных новым связям), то новый стимул будет использовать эти вновь созданные пути, автоматически активируя подходящую реакцию и не нуждаясь в отдельном процессе распознавания либо идентификации.

Представленная ниже теория занимает эмпирическую, или «коннекционистскую», позицию по поставленным вопросам. Теория была разработана для гипотетической нервной системы, или машины, именуемой *перцептроном*. Перцептрон предназначен для иллюстрации некоторых фундаментальных свойств интеллектуальных систем вообще, не опускаясь слишком глубоко в специальные и зачастую неизвестные условия, имеющие место в конкретных биологических организмах. Аналогия между перцептроном и биологическими системами должна быть очевидна читателю.

Достижения нескольких последних десятилетий в области символической логики, цифровых компьютеров и теории переключательных схем поразили

многих теоретиков функциональным сходством между нейроном и простыми двухпозиционными элементами, из которых строятся компьютеры. Были также предложены аналитические методы представления очень сложных логических функций в терминах таких элементов. Результатом стало огромное количество моделей мозга, сводящих его к логическим приспособлениям для выполнения определенных алгоритмов (представляющих «вспоминание», сравнение стимулов, преобразование и различные виды анализа) в ответ на последовательности стимулов; см., например, Rashevsky (1938), McCulloch (1951), McCulloch and Pitts (1943); Culbertson (1950), Kleene (1951), Minsky (1956). В сравнительно небольшом числе теорий, например Ashby (1952), von Neumann (1951) и von Neumann (1956), рассматривался вопрос о том, как можно построить несовершенную нейронную сеть, содержащую много случайных связей, которая надежно выполняла бы функции, представляемые идеализированными электрическими схемами. К сожалению, язык символической логики и булевой алгебры совершенно не подходит для таких исследований. Потребность в подходящем языке для математического анализа событий в системах, где можно охарактеризовать лишь общую организацию, а точная картина неизвестна, заставила автора сформулировать представленную модель в терминах теории вероятностей, а не символической логики.

Упомянутых выше теоретиков в основном интересовал вопрос о том, как функции типа восприятия и вспоминания могут быть реализованы в произвольной детерминированной физической системе, а не о том, как это в действительности происходит в мозгу. Все предложенные модели неудовлетворительны в некоторых важных аспектах (отсутствует эквипотенциальность, не учитывается нейроэкономика, имеет место избыточная специфичность связей и требований к синхронизации, налицо нереалистичная специфичность стимулов, достаточных для возбуждения клетки, постулируется наличие переменных или функциональных особенностей, для которых неизвестны никакие нейрологические корреляты, и т. д.). Сторонники таких подходов говорят, что коль скоро будет показано, как какая-нибудь физическая система может воспринимать или распознавать стимулы либо выполнять другие свойственные мозгу функции, останется только уточнить или модифицировать имеющиеся принципы, чтобы разобраться в работе более реалистичной нервной системы и устранить вышеупомянутые недостатки. Автор занимает другую позицию, считая, что эти недостатки таковы, что простого уточнения или улучшения уже предложенных принципов будет недостаточно для объяснения биологического интеллекта; *различие в принципах* функционирования слишком велико. Теория статистической разделимости (Rosenblatt, 1958b), которая кратко изложена ниже, похоже, позволяет принципиально разрешить все эти трудности.

Те теоретики – Hebb (1949), Milner (1957), Eccles (1953), Hayek (1952), – которые занимались в большей степени прямым изучением биологической нервной системы и ее активности в естественной среде, а не формально аналогичными машинами, обычно не так точны в формулировках, и их анализ далек от строгости, поэтому зачастую трудно оценить, может ли описываемая ими система работать в реальной нервной системе, и при каких необходимых и достаточных условиях. Здесь также отсутствие аналитического языка,

сравнимого по эффективности с булевой алгеброй, применяемой в анализе сетей, стало одним из основных препятствий. Вкладом этой группы ученых следует, быть может, считать предложения о том, что искать и изучать, а не законченные теоретические системы как таковые. Если смотреть под этим углом, то самой многообещающей, с точки зрения последующей теории, является работа Хебба и Хайека. Их позиция, развитая, в частности, в статьях Hebb (1949), Hayek (1952), Uttley (1956), Ashby (1952), на которой основана теория перцептрона, может быть кратко изложена в следующих допущениях.

1. Физические связи в нервной системе, участвующие в обучении и распознавании, в разных организмах различны. При рождении построение самых важных сетей в основном случайно и подчинено минимальному числу генетических ограничений.
2. Оригинальная система связанных клеток способна к определенной пластичности; после периода нервной активности вероятность того, что стимул, поданный одному набору клеток, вызовет реакцию в каком-то другом наборе, по-видимому, изменяется вследствие каких-то продолжающихся относительно долго изменений в самих нейронах.
3. После подачи большой выборки стимулов те, которые в наибольшей степени «похожи» (в некотором смысле, который следует определять в терминах конкретной физической системы), склонны формировать пути к одним и тем же наборам реагирующих клеток. Те же, которые явно «непохожи», склонны формировать связи с разными наборами реагирующих клеток.
4. Положительное и (или) отрицательное подкрепление (или стимул, выполняющий эту функцию) может облегчить или затруднить происходящее в данный момент формирование связей.
5. *Сходство* в такой системе представлено на некотором уровне нервной системы тенденцией похожих стимулов активировать одни и те же наборы клеток. Сходство не является обязательным атрибутом каких-то конкретных формальных или геометрических классов стимулов, а зависит от физической организации воспринимающей системы, организации, которая эволюционирует посредством взаимодействия с окружающей средой. Структура системы, а также экология пары стимул–среда влияет и в значительной степени определяет классы «вещей», на которые подразделяется субъективно воспринимаемый мир.

18.1. Организация перцептрона

Организация типичного фотоперцептрона (перцептрона, реагирующего на оптические образы в качестве стимулов) показана на рис. 18.1. Правила его организации таковы:

1. Стимулы воздействуют на сетчатку, состоящую из сенсорных элементов (S-точек), которые предположительно реагируют в одних моделях по принципу «все или ничего», а в других – импульсом, амплитуда или

частота которого пропорциональна интенсивности стимула. В моделях, рассматриваемых в настоящей статье, предполагается реакция по принципу «все или ничего».

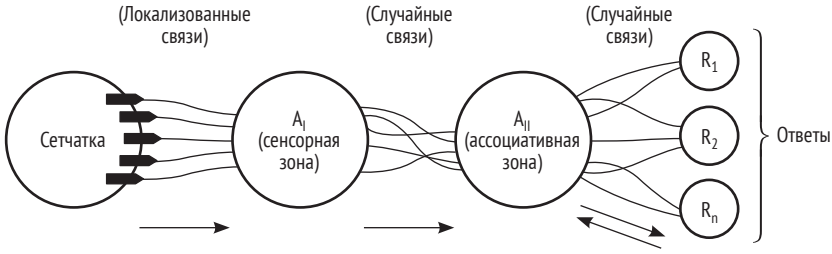


Рис. 18.1. Организация перцептрона

2. Импульсы передаются ассоциативным клеткам (А-элементам) в «сенсорной зоне» (A_I). Эта сенсорная зона может быть опущена в некоторых моделях, где сетчатка связана непосредственно с ассоциативной зоной (A_{II}). Каждая клетка в сенсорной зоне получает сигналы по нескольким связям с сенсорными точками. Множество S-точек, передающих импульсы конкретному А-элементу, будем называть *исходными точками* этого А-элемента. С точки зрения воздействия на А-элемент, исходные точки могут быть *возбуждающими* или *тормозящими*. Если алгебраическая сумма интенсивностей возбуждающих и тормозящих импульсов больше или равна порогу (θ) А-элемента, то А-элемент возбуждается, опять же по принципу «все или ничего» (или, в некоторых моделях, которые мы здесь не рассматриваем, с частотой, зависящей от суммарного поступившего импульса). Исходные точки А-элементов в сенсорной зоне имеют склонность кластеризоваться или фокусироваться вокруг некоторой центральной точки, своей для каждого А-элемента. Количество исходных точек экспоненциально уменьшается при возрастании расстояния от центральной точки рассматриваемого А-элемента до сетчатки. (Такое распределение, похоже, подкрепляется физиологическими фактами и играет важную функциональную роль в распознавании контуров.)
3. Предполагается, что связи между сенсорной зоной и ассоциативной зоной (A_{II}) случайны. То есть к каждому А-элементу в множестве A_{II} приходит какое-то число волокон от исходных точек в множестве A_I, но эти исходные точки случайным образом разбросаны по сенсорной зоне.
4. «Ответы» R_1, R_2, ..., R_n – это клетки (или множества клеток), которые реагируют практически так же, как А-элементы. Для каждого ответа обычно имеется большое число исходных точек, случайно распределенных по множеству A_{II}. Множество А-элементов, передающих импульсы конкретному ответу, будем называть множеством-источником для этого ответа. (Множество-источник ответа совпадает с его множеством исходных точек в А-системе.) Стрелки на рис. 18.1 обозначают направ-

ление передачи в сети. Отметим, что вплоть до A_{II} все связи прямые, никаких обратных связей нет. Но когда мы доходим до последнего набора связей, между A_{II} и R-элементами, связи устанавливаются в обоих направлениях. В большинстве моделей перцептрона возможно одно из двух правил управления обратными связями:

(a) каждый ответ имеет возбуждающие обратные связи с клетками в своем собственном множестве-источнике;
(b) каждый ответ имеет тормозящие обратные связи с клетками в дополнение к своему множеству-источнику (т. е. стремится запретить активность в любых ассоциативных клетках, которые ничего не передают ему).

Первое из этих правил кажется более правдоподобным с анатомической точки зрения, т. к. R-элементы могут находиться в том же корковом поле, что и соответствующие им множества-источники, так что взаимное возбуждение R-элементов и A-элементов подходящего множества-источника весьма вероятно. Однако альтернативное правило (b) приводит к системе, более легко поддающейся анализу, и потому именно оно будет предполагаться для большинства рассматриваемых здесь систем. <…>

18.2. Выводы и оценка

Основные выводы, вытекающие из теоретического изучения перцептрона, можно сформулировать следующим образом.

1. В среде со случайными стимулами система, состоящая из случайно связанных элементов, с описанными выше параметрическими ограничениями, может обучиться ассоциировать определенные ответы с определенными стимулами. Даже если с каждым ответом ассоциировано много стимулов, они все равно могут быть распознаны с вероятностью, превышающей случайность, хотя могут быть очень похожи друг на друга и могут активировать много одинаковых сенсорных входов в систему.
2. В такой «идеальной среде» вероятность правильного ответа уменьшается до первоначального уровня случайности, по мере того как количество выученных стимулов возрастает.
3. В такой среде не существует никакой основы для обобщения.
4. В «дифференцированной среде», где каждый ответ ассоциирован с отдельным классом взаимно коррелированных, или «похожих», стимулов, вероятность того, что обученная ассоциация некоторого конкретного стимула будет корректно сохранена, обычно стремится к асимптоте, лучшей, чем случайность, по мере того как количество выученных системой стимулов возрастает. Эту асимптоту можно сделать сколь угодно близкой к единице посредством увеличения числа ассоциативных клеток в системе.

5. В дифференцированной среде вероятность того, что стимул, не встречавшийся ранее, будет корректно распознан и ассоциирован с правильным классом (вероятность корректного обобщения), стремится к той же асимптоте, что и вероятность корректного ответа на ранее подкрепленный стимул. Эта асимптота будет лучше случайности, если для рассматриваемых классов стимулов выполняется неравенство $P_{c12} < P_a < P_{c11}$.
6. Качество системы можно улучшить за счет использования чувствительной к контурам сенсорной зоны и применения бинарной системы ответов, в которой каждый ответ, или «бит», соответствует какому-то независимому свойству или атрибуту стимула.
7. Обучение методом проб и ошибок возможно в бивалентных системах подкрепления.
8. Временны́е организации как комбинаций стимулов, так и ответов могут быть выучены системой, которая использует только расширение исходных принципов статистической разделимости, без внесения каких-либо серьезных усложнений в организацию системы.
9. Память перцептрона *распределенная* в том смысле, что любая ассоциация может использовать большую долю клеток в системе, а удаление части ассоциаций из системы не окажет ощутимого влияния на качество любого различения или ассоциации, но начнет проявляться как общий дефицит всех обученных ассоциаций.
10. Возможны простые когнитивные способности, избирательное вспоминание и спонтанное распознавание классов в данной среде. Однако распознавание пространственно-временных связей, похоже, выходит за пределы способности перцептрона формировать когнитивные абстракции.

Психологи (и специалисты по теории обучения в особенности) могут теперь задать вопрос: «Чего достигла предложенная теория сверх того, что уже было сделано в количественных теориях Халла, Буша и Мостеллера и пр. или в физиологических теориях наподобие теории Хебба?» Конечно, представленная теория все еще слишком примитивна, чтобы ее можно было рассматривать как полноценного соперника имеющимся теориям обучения человека. Тем не менее в первом приближении ее главное достижение можно сформулировать следующим образом:

Для данного режима организации (α, β или γ; Σ или μ; моновалентного или бивалентного) фундаментальный феномен *обучения*, *перцептивного различения и обобщения можно полностью предсказать, зная шесть базовых физических параметров*, а именно:

x: количество возбуждающих связей на один A-элемент,
y: количество тормозящих связей на один A-элемент,
θ: ожидаемый порог A-элемента,
ω: доля R-элементов, с которыми связан A-элемент,
N_A: количество A-элементов в системе,
N_R: количество R-элементов в системе.

N_A (количество сенсорных элементов) становится важно, если оно очень мало. Предполагается, что в начальный момент распределение значений элементов системы равномерное; в противном случае нужно было бы еще задавать начальное распределение. *Каждый из вышеперечисленных параметров является четко определенной физической величиной, которая может быть измерена самостоятельно, независимо от поведенческих и перцептивных феноменов, которые мы пытаемся предсказать.*

Прямым следствием того, что представленная система опирается на физические величины, является тот факт, что она выходит далеко за рамки имеющихся теорий обучения и поведения в трех главных отношениях: экономность, проверяемость и объяснительная способность и общность. Рассмотрим эти три момента по очереди.

18.2.1. Экономность. По существу, все базовые величины и законы, используемые в этой системе, уже присутствуют в структуре физических и биологических наук, так что мы нашли необходимым постулировать только одну гипотетическую величину (или конструкцию), которую назвали V, «значение» ассоциативной клетки; эта величина должна обладать некоторыми функциональными характеристиками, которые можно точно сформулировать и для которых предположительно имеется потенциально измеримый физический коррелят.

18.2.2. Проверяемость. Прежние количественные теории обучения, кажется, без единого исключения, обладали одной важной общей характеристикой: все они были основаны на измерениях *поведения* в определенных ситуациях и использовали эти измерения (после теоретической обработки) для предсказания *поведения* в других ситуациях. Такая процедура в конечном итоге сводится к подгонке и экстраполяции кривых в надежде, что константы, описывающие один набор кривых, подойдут и для других кривых в иных ситуациях. Хотя такая экстраполяция, строго говоря, не круговая, для нее характерны многие трудности экстраполяции окружностью, особенно когда она применяется для «объяснения» поведения. Такую экстраполяцию трудно обосновать в новой ситуации, и было показано, что если выводить базовые константы и параметры заново для каждой ситуации, в которой они не выдерживают эмпирическую проверку (например, при переходе от белых крыс к людям), то базовая «теория» оказывается, по существу, неопровержимой, как неопровержимо любое уравнение, аппроксимирующее кривую. Фактически психологи в массе своей признают, что нет особого смысла доказывать ошибочность любой из основных имеющихся на сегодня теорий обучения, потому что путем расширения или изменения параметров все они продемонстрировали способность адаптироваться к любым конкретным эмпирическим данным. Это нашло отражение в распространяющемся все шире мнении, что выбор теоретической модели – в основном дело личных эстетических предпочтений или предрассудков и что каждый ученый имеет право на собственную любимую модель. Сталкиваясь с таким подходом, невольно вспоминаешь замечание, приписываемое Кистяковскому: «с семью параметрами я смогу описать слона». Очевидно, что это не относится к системе, в которой для предсказанного поведения можно независимо измерить

независимые переменные. В такой системе не получится «силой» подогнать теорию к эмпирическим данным, если текущие параметры приводят к неудобным результатам. В текущей теории невозможность аппроксимировать данные кривой в новой ситуации стала бы ясным указанием на то, что неверна либо теория, либо эмпирические измерения. Следовательно, если теория выдерживает повторяющиеся испытания, то наша уверенность в ее верности и общности значительно возрастает по сравнению с теорией, которую приходится вручную подгонять под каждую ситуацию.

18.2.3. Объяснительная способность и общность. Представленная теория, будучи выведена из базовых физических величин, не является специфичной для какого-то одного организма или ситуации обучения. В принципе, ее можно обобщить на любую форму поведения в любой системе, для которой известны физические параметры. Теория обучения, построенная на этих основаниях, должна быть значительно более мощной, чем любая из предложенных ранее. Она не только скажет, каким должно быть поведение любого известного организма, но и позволит синтезировать системы, ведущие себя согласно специальным требованиям. Другие теории обучения становятся все более качественными (а не количественными) по мере обобщения. Так, система уравнений, описывающая эффекты вознаграждения белой крысы, обучающейся находить выход из Т-образного лабиринта, при попытке обобщить ее на другие виды и иные ситуации сводится просто к утверждению о том, что вознаграждаемое поведение имеет место с большей вероятностью. Представленная здесь теория не теряет точности в процессе обобщения. <…>

19 Некоторые этические и технические последствия автоматизации (1960)

Норберт Винер

Норберт Винер (1894–1964), сын профессора кафедры славянских языков и литературы в Гарварде, был вундеркиндом. Когда он поступил в колледж Тафтса в возрасте 11 лет, в новостях появилась статья, где его величали «самым удивительным ребенком в мире» (Unknown 1906). Винер беседовал с репортером о философии и говорил на латыни и греческом. Он окончил колледж Тафтса за три года со степенью по математике, в возрасте 18 лет получил докторскую степень по философии в Гарварде и начал преподавать в МТИ.

На протяжении своей карьеры в МТИ, где он работал до конца жизни, Винер пересекался со многими пионерами компьютерной революции. В 1920-х годах он работал с Ванневаром Бушем над проектированием устройства для решения дифференциальных уравнений и внес провидческое предложение строить их с использованием вакуумных ламп, а не механических сочленений. Обратив внимание на сходство между машинами и живыми существами, он сотрудничал с Уорреном Мак-Каллоком и Уолтером Питтсом, который был его протеже в академических кругах вплоть до момента трагического разрыва отношений всех троих (стр. 116).

Винер был отцом дисциплины, которую он назвал «кибернетикой» – изучение «управления и передачи информации в живом организме и в машине». Ключевой концепцией была «обратная связь» (feedback), это слово кибер-

нетика позаимствовала из греческого языка, где оно означало управление судном, как с помощью румпеля. Интерес Винера к сообщениям, которые передаются между частями системы для управления ее поведением, естественно вел к работе Клода Шеннона по теории информации.

Основной математический инструментарий Винера был непрерывным и статистическим, а не дискретным и цифровым. Его работы по обратной связи и управлению имели огромное военное значение, например, для управления ракетами. После Второй мировой войны его все сильнее беспокоили применения, которые нашлись его научной работе. В 1946 году в открытом письме к военному подрядчику, который попросил его прислать экземпляр одной работы, он писал: «Политика самого правительства во время войны и после нее, в частности при бомбардировке Хиросимы и Нагасаки, ясно показала, что предоставление научной информации не всегда является невинным актом. <…> Обмен идеями, одна из величайших традиций науки, безусловно, должен быть ограничен, когда ученый становится арбитром между жизнью и смертью. Меры, предпринятые нашими военными учреждениями во время войны, с целью ограничить свободное общение ученых, работающих над родственными проектами или даже над одним и тем же проектом, зашли настолько далеко, что стало ясно, что если эта политика будет продолжаться и в мирное время, то приведет к полной безответственности ученого и, в конечном итоге, к гибели науки. То и другое было бы катастрофой для нашей цивилизации и повлекло бы за собой серьезную и непосредственную угрозу для общества». Он отказался прислать статью. Он понимал, что ее текст можно получить другими способами, но хотел сделать заявление.

Письмо, опубликованное в журнале «Atlantic» под названием «Ученый восстает!», получило широкую известность (Wiener 1947). Затем Винер отозвал свое согласие выступить на крупном гарвардском симпозиуме по автоматическим вычислениям, организованном Говардом Эйкеном при поддержке ВМФ. Винер не собирался представить это еще одним публичным заявлением, но уже была напечатана и распространена на конференции программа, в которой его имя было перечеркнуто. Широкая огласка этого случая вызвала неловкость у всех сторон, и в результате Винер отдалился от Эйкена и попал под подозрение сенатора Джозефа Маккарти, организатора антикоммунистической охоты на ведьм. Но Винер сделал решительный шаг и больше никогда не принимал от правительства денег на свои исследования. Некоторые превозносили его этическую позицию, но его научный авторитет стал падать, поскольку послевоенный научный бум подпитывался деньгами Министерства обороны.

Многие предостережения Винера по поводу моральной ответственности ученых и опасностей секретных исследований сегодня звучат особенно актуально. Последние годы своей жизни он посвятил мирным применениям кибернетики, а акцент в его предупреждениях сместился с рисков милитаризации на изменения в повседневной жизни, сопутствующие техническому прогрессу.

Все современные технологии вычислительной техники и связи в научном долгу перед Винером, но определенная им отрасль науки практически исчезла, растворившись в различных наследниках. Слово «кибернетика» теперь ус-

лышишь редко, но приставка «кибер» прилепилась к словам «преступление», «безопасность» и цифровым вариантам многих других понятий. Винер стал тем, кого биограф называет «теневым героем», его влияние присутствует везде, но сам он остается невидимым (Conway 2005).

От его текстов по-прежнему веет предощущением опасности. При всей своей выдающейся педагогической и технической мудрости он предостерегает нас от, как он выразился в книге «Человеческое применение человеческих существ» (Wiener 1950), «американского поклонения “знанию, как делать” в ущерб “знанию, что делать”».

Примерно 13 лет назад вышла моя книга «Кибернетика» (Wiener 1948). В ней я обсуждал проблемы управления и передачи информации в живом организме и в машине. Я сделал немало предсказаний по поводу развития управляемых машин и соответствующих методов автоматизации, которые, как я предвидел, будут иметь важные последствия для общества будущего. Теперь, 14 лет спустя, представляется уместным провести инвентаризацию текущего положения дел как в кибернетической технике, так и в ее социальных последствиях.

Прежде чем переходить к детальному рассмотрению этих вопросов, я хотел бы сказать пару слов об отношении человека с улицы к кибернетике и автоматизации. Это отношение нуждается в критическом обсуждении и, на мой взгляд, должно быть отвергнуто целиком и полностью. Нередко оно высказывается в форме утверждения о том, что машина не может выдать ничего такого, что не было в нее заложено. Зачастую это интерпретируется так, что машина, сделанная человеком, должна навеки оставаться подчиненной человеку, так что ее работа в любой момент времени открыта для человеческого вмешательства и внесения изменений. Основываясь на таком отношении, многие люди презрительно отметают все опасности применения машин и решительно отрицают ранние предсказания Сэмюэла Батлера о том, что машина может захватить контроль над человечеством. [Примечание редактора: здесь Винер ссылается на работы Butler (1863, 1872), первоначально опубликованные анонимно.]

Да, действительно, во времена Сюмюэла Батлера машины были куда менее опасны, чем сегодняшние, поскольку обладали только мощью, но ни в какой степени не были способны к мышлению и коммуникации. Однако технические возможности машин сегодняшнего дня проникли и в эти области, так что реальная современная машина сильно отличается от того образа, который имел в виду Батлер, и мы не можем распространить на эти новые устройства предположения, которые казались аксиомами поколение назад. Я вижу перед собой публику, сформировавшую свое отношение к машине на основе неполного понимания структуры и способа работы современных машин.

Я утверждаю, что машины могут и действительно преступают некоторые пределы, заложенные их конструкторами, и что, совершая это, они могут оказаться и эффективными, и опасными. Вполне возможно, что мы в принципе не способны изготовить машину, элементы поведения которой не сможем

понять рано или поздно. Но это вовсе не означает, что мы сможем понять эти элементы за время, существенно меньшее того, какое необходимо машине для действия, или даже исчисляемое заданным числом лет или поколений.

Как теперь уже общепризнано, в некотором ограниченном диапазоне операций машины действуют гораздо быстрее человека и оказываются гораздо точнее в деталях этих операций. А раз так, то даже если машины не могут превзойти человека в интеллектуальном отношении, они прекрасно могут и зачастую превосходят его в эффективности выполнения заданий. Осмысление их способа достижения этой эффективности может прийти спустя длительное время после того, как поставленное задание было выполнено.

Это означает, что хотя теоретически машины могут быть предметом критики со стороны человека, эта критика может не оказать никакого эффекта, пока не станет слишком поздно. Чтобы успешно предотвратить катастрофические последствия, наше понимание изготовленных человеком машин должно, вообще говоря, развиваться *pari passu*[1] с характеристиками машин. В силу крайней медлительности человеческих действий эффективный контроль над нашими машинами может быть сведен к нулю. К тому времени, как мы сможем отреагировать на информацию, которую сообщают наши органы чувств, и остановить автомобиль, которым управляем, он уже врежется в стену.

19.1. Игры

Я еще вернусь к этому вопросу ниже в этой статье. А пока я, с вашего позволения, хочу обсудить применение машин для очень специфической цели: игр. Конкретно я буду говорить об игре в шашки, для которой компания International Business Machines Corporation разработала весьма эффективные игровые машины.

Позвольте мне раз и навсегда отметить, что нас здесь не интересуют машины, действующие в соответствии с точной замкнутой теорией игры, в которую играют. Теория игр, развитая фон Нейманом и Моргенштерном, быть может, и иллюстрирует работу фактических игровых машин, но не дает их исчерпывающего описания.

В игре столь сложной, как шашки, если каждый игрок будет пытаться выбрать свой ход, принимая во внимание лучший ход, который может сделать противник, опять же принимая во внимание собственный лучший ход, опять же принимая во внимание лучший ход противника и т. д., то он возложит на себя непосильную задачу. Мало того что это не в человеческих силах, так еще и нет причин полагать, что это лучшая стратегия игры с противником, который испытывает те же ограничения, что и он сам.

Теория игр фон Неймана имеет мало общего с теорией, на основе которой действуют игровые машины. Последняя гораздо ближе к методам, применяемым опытными, но ограниченными в своих возможностях шахматистами

---

[1] Здесь «нога в ногу» (лат.). – *Прим. перев.*

против других шахматистов. Такие игроки опираются на некоторые стратегические оценки, по существу своему неполные. Хотя фоннеймановский тип игры имеет силу для игр типа крестики и нолики, где теория полна, интерес к шахматам и шашкам зиждется только на том, что для них не существует полной теории. Нет такой теории ни для войны, ни для деловой конкуренции и вообще ни для одного вида конкурентной деятельности, который был бы нам действительно интересен.

В игре типа крестики и нолики, с малым числом возможных ходов, где каждый игрок в состоянии рассмотреть все возможности и выстроить защиту против возможных наилучших ходов противника, полная теория фон Неймана справедлива. В таком случае игра неминуемо закончится победой первого игрока, победой второго игрока или ничьей.

Я задаю вопрос, является ли эта концепция идеальной игры реалистичной в случае практических нетривиальных игр. Великие генералы вроде Наполеона и великие адмиралы вроде Нельсона действовали по-другому. Они знали не только об ограничениях своих противников в таких вопросах, как материально-технические и людские ресурсы, но также об их ограничениях в опыте и знании военной науки. И именно благодаря реалистичной оценке относительной неопытности континентальных держав в морских операциях по сравнению с отлично развитой тактической и стратегической компетентностью британского флота Нельсон сумел выказать бесстрашие, позволившее вытеснить континентальные силы с морей. Это было бы невозможно, если бы он ввязался в длительный, относительно нерешительный и, возможно, проигрышный конфликт, на который его обрекало предположение о наилучшей возможной стратегии противника.

Оценивая не только материально-технические и людские ресурсы врага, но также его ожидаемую проницательность и мастерство в тактике и стратегии, Нельсон исходил из информации о его поведении в прошлых сражениях. Точно так же, в столкновениях с австрийцами в Италии Наполеон учитывал важный фактор – закоснелость и недальновидность Вюрмсера.

Этот элемент, опыт, необходимо в должной мере учитывать в любой реалистичной теории игр. Шахматист играет не с идеальным несуществующим, не делающим ошибок противником, а с человеком, привычки которого он может вызнать на основе прошлых партий. Таким образом, в теории игр необходимы, по крайней мере, два разных вида интеллектуальных усилий. Одно – краткосрочное стремление играть с определенной стратегией на одну конкретную игру. Другое – изучение имеющейся информации о большом количестве игр. Это информация об играх, сыгранных самим игроком, его противником и даже игроками, с которыми он лично не встречался. Располагая этой информацией, он определяет сравнительные преимущества различных стратегий, доказавших свое преимущество в прошлом.

В шахматах существует еще и третий уровень суждений. Он выражается, по крайней мере частично, протяженностью значимого прошлого. Развитие теории в шахматах уменьшает важность партий, сыгранных на ином уровне искусства. С другой стороны, прозорливый шахматный теоретик может заранее оценить, что определенная модная нынче стратегия утратила ценность и что, возможно, будет лучше вернуться к прежним способам игры в пред-

видении изменения стратегии у тех, кого он считает своими вероятными противниками.

Итак, при определении линии поведения в шахматах имеется несколько уровней рассмотрения, которые некоторым образом соответствуют различным логическим типам Бертрана Рассела. Есть уровень тактики, уровень стратегии, уровень общих соображений, которые следует взвешивать при определении этой стратегии, уровень, на котором принимается во внимание протяженность того отрезка прошлого, на котором эти соображения могут быть верны, и т. д. Каждый новый уровень требует гораздо более объемного исторического материала, чем предыдущий.

Я сравнил эти уровни с логическими типами Рассела, относящимися к классам, классам классов, классам классов классов и т. д. Можно отметить, что Рассел не считает утверждения, включающие все типы, значимыми. Он подчеркивает бессмысленность таких вопросов, как, например, о брадобрее, который бреет всех тех и только тех, кто не бреется сам. Бреет ли он себя? При одном типе бреет, при следующем не бреет и т. д. до бесконечности. Все такие вопросы, включающие бесконечность типов, могут приводить к неразрешимым парадоксам. Аналогично поиск лучшей линии поведения при всех уровнях сложности – занятие тщетное и не ведет ни к чему, кроме замешательства.

Эти соображения возникают при определении стратегии как машин, так и людей. Эти же вопросы возникают при программировании программирования. Игровые машины самого низкого типа играют в терминах формальной оценки позиции. Таким величинам, как количество выигранных или проигранных фигур, контроль над фигурами, их подвижность и т. д., можно придать числовые веса, основанные на какой-то эмпирике, и такое взвешивание можно применять к каждой следующей позиции в соответствии с правилами игры. Выбирать следует позицию с максимальным весом. При таких обстоятельствах игра машины будет казаться противнику – который не может не оценивать шахматную индивидуальность машины – негибкой.

19.2. Самообучающиеся машины

Следующий шаг машины – принимать во внимание не только ходы в отдельно взятой партии, но и историю ранее сыгранных партий. Исходя из этого, машина может время от времени приостанавливаться и не играть, а оценивать, какое (линейное или нелинейное) назначение весов различным факторам, предложенным ей для рассмотрения, лучше всего соответствует выигранным, а какое – проигранным (или сведенным вничью) партиям. Затем машина продолжает играть с новыми весами. Такая машина будет казаться противнику-человеку гораздо более гибкой личностью, и те приемы, которые приводили к ее проигрышу на более ранней стадии, теперь не смогут ее обмануть.

В настоящее время уровень таких самообучающихся машин позволяет им играть на уровне любителя в шахматы, но в шашках они достигают очевидно-

го превосходства над программировавшим их игроком уже через 10–20 часов работы и инструктирования. Таким образом, они, безусловно, уходят из-под полного и эффективного контроля со стороны своего создателя. Каким бы жестко заданным ни был набор факторов, который они могут принимать во внимание, они без сомнения – так говорят все, кто с ними играл, – демонстрируют оригинальность не только в тактике, которая может оказаться совершенно непредвиденной, но даже и в деталях назначения весов своей стратегии.

Как я уже сказал, самообучающиеся шашечные машины продвинулись до такой степени, что могут обыграть программиста. Однако у них, похоже, все же есть одна слабость. Проявляется она в эндшпиле. Здесь машина колеблется в выборе наилучшего способа нанести *coup de grâce*[1]. Связано это с тем, что существующие машины запрограммированы следовать одной и той же стратегии на каждом этапе игры. Поскольку ценность всех шашек практически одинакова, это вполне естественно на протяжении большей части игры, но перестает быть таковым, когда доска опустела и главная проблема заключается в том, чтобы занять некую позицию, а не атаковать напрямую. В рамках описанных выше методов вполне возможно провести вторую оценку и определить, какой должна быть линия поведения после того, как количество шашек у противника уменьшилось настолько, что эти новые соображения становятся преобладающими.

Шахматные машины пока еще не достигли степени совершенства шашечных, хотя, как я сказал, они играют на уровне хорошего любителя. Причина, вероятно, та же, что у относительной неэффективности игры в шашечном эндшпиле. В шахматах стратегия в миттельшпиле сильно отличается не только от стратегии в эндшпиле, но и от стратегии в дебюте. Различие между шашками и шахматами в этом отношении состоит в том, что начало игры в шашках не особенно отличается от ее характера в миттельшпиле, тогда как в шахматах подвижность фигур в начальной расстановке очень низкая, поэтому задача их развития из этой позиции особенно трудна. Именно поэтому дебют и развитие фигур образуют специальный раздел шахматной теории.

Есть разные способы сообщить машине об этих хорошо известных фактах и заставить ее исследовать отдельную стратегию ожидания в дебюте. Это не означает, что тот тип теории игры, который я здесь обсуждал, неприменим к шахматам, просто требуется гораздо больше потрудиться, прежде чем мы сможем изготовить машину, играющую в шахматы на уровне мастера. Некоторые мои друзья, занимающиеся этими проблемами, полагают, что эта цель будет достигнута через 10–25 лет. Не будучи специалистом по шахматам, я не берусь давать такие прогнозы по собственной инициативе.

Очень может статься, что самообучающиеся машины будут использованы для программирования нажатия кнопки в новой войне, которая начнется нажатием этой самой кнопки. Здесь мы говорим о той области, в которой несамообучающиеся автоматы, вероятно, уже используются. Ни в коем случае не стоит вопрос о программировании этих машин на основе фактического опыта в реальной войне. Прежде всего потому, что обретение опыта, доста-

---

[1] Смертельный удар. – *Прим. перев.*

точного для адекватного программирования, скорее всего, будет означать, что человечество уже стерто с лица земли.

Кроме того, методы ведения войны по нажатии кнопки по необходимости изменятся так сильно, что к моменту накопления адекватного опыта основания для начала войны радикально изменятся. Поэтому программирование такой самообучающейся машины должно быть основано на какой-то военной игре – так сейчас командиры и кадровики обучаются важным элементам стратегического искусства. Однако здесь, если правила победы в военной игре не соответствуют тому, что мы действительно хотим получить для своей страны, то такая машина с очень высокой вероятностью выработает линию поведения, которая позволит одержать номинальную победу ценой уничтожения всего, что нам дорого, и даже национального выживания.

19.3. Человек и раб

Проблема – моральная проблема, – с которой мы здесь сталкиваемся, очень близка к величайшим проблемам рабства. Примем, что рабство – это плохо, потому что оно жестоко. Однако это утверждение противоречиво и по причине совершенно иного рода. Мы хотим, чтобы раб был разумен, чтобы он был способен помочь нам в решении наших задач. Однако мы также хотим, чтобы он был покорен. Полная покорность и полная разумность несовместимы. Как часто в античной истории случалось, что умный греческий философ-раб, принадлежавший менее умному римлянину-рабовладельцу, принужден был руководить действиями своего хозяина, а не подчиняться его желаниям! Аналогично, если машины будут становиться все более эффективными и действовать на все более высоком психологическом уровне, катастрофа, которую предвидел Батлер, – доминирование машины – будет неотвратимо приближаться.

Когда мы вступаем в области высшей логики, мозг человека оказывается гораздо более эффективным аппаратом управления, чем разумная машина. Это самоорганизующаяся система, которая зависит от способности преобразовывать себя в новую машину, а не от стопроцентной точности и скорости решения задач. Мы уже изготовили очень успешные машины низшего логического типа, с фиксированной линией поведения. Мы начинаем изготавливать машины второго логического типа, линия поведения которых улучшается по мере обучения. При конструировании машин невозможно предвидеть конкретный предел их логического типа и небезопасно делать какие-то заявления о том, на каком именно уровне мозг превосходит машину. Тем не менее еще долго будет существовать некий уровень, на котором мозг лучше построенной машины, пусть даже этот уровень отодвигается все дальше и дальше.

Можно видеть, что в результате совершенствования техники программирования автоматизации конструктор и оператор постепенно перестают понимать многие этапы, посредством которых машина приходит к определенным выводам, а также истинные тактические цели многих ее действий.

Это имеет прямое отношение к проблеме нашей неспособности предвидеть нежелательные последствия вне рамок стратегии игры, пока машина еще работает и пока наше вмешательство могло бы предотвратить наступление этих последствий.

Здесь необходимо осознать, что действие человека – это действие обратной связи. Для предотвращения катастрофических последствий мало, чтобы какого-то нашего действия было достаточно для изменения поведения машины, потому что вполне может случиться, что у нас нет информации, на основе которой мы могли бы судить о таком действии.

В нейрофизиологии атаксия может быть таким же серьезным поражением, как паралич. У пациента, страдающего локомоторной атаксией, мышцы и двигательные нервы могут быть в полном порядке, но если его мышцы, сухожилия и органы не сообщают ему точно, какое положение в пространстве он занимает и приведет ли напряжение, которому подвергаются его органы, к падению или не приведет, то пациент не сможет встать. Аналогично, если построенная нами машина способна обрабатывать поступающие ей данные в темпе, за которым нам не угнаться, то мы не можем узнать, пока не станет слишком поздно, когда ее надо выключить. Все мы знаем историю об ученике чародея, в которой мальчик заставляет метлу носить воду, пока его учитель отлучился, а когда учитель наконец появляется, он застает ученика почти утонувшим. Если бы мальчик стал искать в волшебных книгах своего учителя заклинание для прекращения озорства, то, наверное, утонул бы, так и не успев найти. Аналогично, если бутылочную фабрику запрограммировать на максимальную производительность, то владелец мог бы обанкротиться из-за гигантского запаса не находящих спроса бутылок, которые машина успела наделать, прежде чем он понял, что надо было еще полгода назад остановить производство.

«Ученик чародея» – одна из многих сказок, основанных на предположении, что волшебные силы все понимают буквально. В «Тысяче и одной ночи» есть сказка о джинне и рыбаке, в которой рыбак ломает печать Соломона, удерживавшую джинна в заточении, и обнаруживает, что джинн дал обет убить своего спасителя. У У. У. Джекобса есть рассказ «Обезьянья лапа», в котором сержант-майор привозит из Индии талисман, выполняющий три желания трех своих владельцев. О первом владельце талисмана нам известно только, что на третий раз он пожелал себе смерти. Сержант-майор, второй владелец, не может рассказать о своем опыте, потому что он слишком ужасен. Его друг, получивший талисман, для начала пожелал 200 фунтов. Вскоре после этого представитель фабрики, на которой работает его сын, приходит сообщить, что сын погиб в результате несчастного случая на производстве и что компания, не принимая на себя ответственность за несчастный случай, посылает ему в утешение сумму в 200 фунтов. Следующим его желанием было вернуть сына, и в его дверь стучится призрак. Третье его желание – чтобы призрак убрался восвояси.

Катастрофических результатов следует ожидать не только в сказках, но и в реальном мире, когда две силы, по существу чуждые друг другу, соединяются в попытке достичь общей цели. Если взаимопонимание между этими двумя силами относительно природы этой цели неполное, то результаты кооперации ожидаемо будут неудовлетворительными. Если для достиже-

ния своих целей мы используем механическую силу, в работу которой после приведения ее в действие не можем эффективно вмешаться, потому что ее операции настолько быстрые и необратимые, что мы не успеваем ничего сделать, прежде чем операция завершится, то лучше бы нам быть на сто процентов уверенными, что цель, заложенная в машину, именно та, которой мы действительно хотим достичь, а не просто ее красочная имитация.

19.4. Временные шкалы

До сих пор я рассматривал квазиэтические проблемы, вызванные одновременным действием машины и человека в совместном предприятии. Мы видели, что одна из основных причин опасаться катастрофических последствий применения самообучающейся машины – то, что действия человека и машины протекают в разных временных шкалах: машина гораздо быстрее человека, и все попытки действовать совместно наталкиваются на серьезные трудности. Проблемы того же рода возникают всякий раз, как два оператора управления с сильно различающимися временными шкалами действуют совместно, независимо от того, какая система быстрее, а какая медленнее. Это ставит перед нами гораздо более прямой этический вопрос: какие этические проблемы возможны, когда человек как отдельная личность действует в рамках контролируемого процесса, характеризующегося гораздо более медленной временной шкалой, например как часть политической истории или – и это основной предмет нашего любопытства – развития науки?

Отметим, что развитие науки – процесс управления и обмена информацией, имеющий конечной целью понимание и управление материей. В этом процессе 50 лет – все равно, что один день в жизни отдельного человека. Поэтому отдельный ученый должен работать как часть процесса, временная шкала которого настолько растянутая, что сам он сможет обозреть лишь очень малый его отрезок. И в этом случае взаимодействие между двумя частями сдвоенной машины оказывается трудным и ограниченным. Даже когда индивидуум верит, что наука способствует достижению тех человеческих целей, к которым он всей душой стремится, эта вера нуждается в постоянном пересмотре и переоценке, что возможно лишь отчасти. Ибо для отдельного ученого даже частичная оценка этой связи между человеком и процессом требует богатого воображения и прозорливого взгляда на историю, что трудно, изнурительно и достижимо лишь в ограниченных пределах. И если мы будем верны символу веры ученого, согласно которому неполное знание о мире и о нас самих все же лучше полного незнания, мы все равно ни в коей мере не сможем оправдать наивное допущение, что чем быстрее мы мчимся вперед, дабы поставить себе на службу новые силы, открывшиеся для нас, тем будет лучше. Мы всегда должны изо всех сил напрягать свои мыслительные способности, чтобы понять, куда могут завести наши вновь обретенные средства и методы.

20 Симбиоз человека и машины (1960)

Дж. К. Р. Ликлайдер

Дж. К. Р. Ликлайдер (1915–1990) отличается от большинства персонажей этой книги. Он не был ни инженером, ни математиком, ни философом. И тем не менее по счастливому стечению обстоятельств, сложившемуся в середине XX века, его видение помогло создать мир, в котором мы теперь живем.

На Ликлайдера неизменно ссылаются только по инициалам, потому что у него было прозвище «Лик». Он получил образование психолога и выполнил важные работы по слуховому восприятию, но затем, работая в МТИ во время холодной войны над проектированием систем противовоздушной обороны, начал размышлять о том, что теперь мы назвали бы «человеческим фактором» в информатике. На оператора ЭВМ возлагалась задача в доли секунды принимать решения, имеющие глобальные последствия, когда он видит перед собой данные о летящих ракетах и другую информацию от военной разведки. Как мог бы сам компьютер помочь человеку принимать наилучшие решения? Современные видеоигры, системы виртуальной реальности и видео котиков в интернете родились из такого желания поставить под контроль риск ядерного катаклизма. Когда центр психологических исследований перебрался в Гарвард, Ликлайдер ушел из МТИ и устроился в кембриджскую компанию Bolt Beranek and Newman (BBN), знаменитую своим опытом в области акустики, но в конечном итоге получившую контракты на изготовление первых шлюзов для сети ARPAnet, предшественницы интернета. Во время своей работы там Ликлайдер и написал оказавшую большое влияние статью, которую мы включили в этот сборник. Кое-чем она обязана воззрениям Ванневара Буша, и Ликлайдер действительно был знаком с Бушем по МТИ. Но к тому времени, когда эта статья была написана, уже существовали компьютеры, которые можно было запрограммировать для отображения интерактивной и динамической графики. Из BBN Ликлайдер в 1962 году

перешел в Управление перспективных исследовательских проектов (ARPA) Министерства обороны США, где способствовал становлению ARPANET.

Ликлайдер покинул правительственное учреждение и недолгое время работал в компании IBM, которую нашел недостаточно дальновидной. В 1966 году он вернулся в МТИ и оставался там почти до конца карьеры с еще одним небольшим перерывом на командировку в Министерство обороны, где руководил перспективными исследовательскими проектами. Он был забавным, красноречивым и скромным человеком. Я встречался с ним, когда в бытность свою студентом работал над гарвардской DEC PDP-1 и помогал отлаживать программу, которую он писал. Я даже не подозревал в нем того самого человека, чей провидческий дар сделал возможной работу, которой я занимался, – я думал, что это какой-то престарелый аспирант, – а он и не делал попыток просветить меня.

20.0. Краткое описание

Симбиоз человека и компьютера – это ожидаемый этап развития взаимодействия между людьми и машинами. Он будет включать тесную связь между людьми и электронными участниками делового сотрудничества. Основными целями являются: 1) открыть перед компьютерами возможность формулировать свои «мысли» по аналогии с тем, как сейчас они помогают в решении сформулированных задач, и 2) позволить людям и компьютерам совместно принимать решения и управлять сложными ситуациями без жестких зависимостей от предопределенных программ. В ожидаемом симбиотическом сотрудничестве подразумевается, что человек будет ставить цели, формулировать гипотезы, определять критерии и давать оценку, а вычислительные машины будут выполнять рутинную работу, без которой невозможно глубокое понимание и принятие решений в области науки и техники. Предварительный анализ показывает, что в симбиотическом сотрудничестве интеллектуальные операции будут выполняться гораздо эффективнее, чем это делает человек без помощников. Для достижения такого эффективного союза необходимо развивать систему разделения времени, компоненты памяти и ее организацию, языки программирования и устройства ввода-вывода.

20.1. Введение

20.1.1. Симбиоз. Фиговое дерево опыляется только насекомым *Blastophaga grossorun*, личинка которого живет в завязи дерева, где и получает пищу. Таким образом, дерево и насекомое находятся во взаимозависимом положении: дерево не может осуществлять процесс воспроизводства без насекомого; насекомое без дерева не сможет найти пропитание; вместе они органи-

зуют жизнеспособное, продуктивное и процветающее сотрудничество. Такое взаимодействие «живущих вместе в тесной связи двух организмов разных видов» называют симбиозом (Webster 1959).

«Симбиоз человека и компьютера» – это подкласс человеко-машинных систем, коих существует много. Однако в настоящий момент никакого симбиоза между человеком и компьютером нет. Цель данной статьи – представить концепцию и, хочется надеяться, оказать содействие развитию симбиоза человека и компьютера через анализ некоторых проблем взаимодействия между человеком и вычислительной машиной, обратив внимание на применимые принципы проектирования систем человек–машина и поставив некоторые вопросы, нуждающиеся в ответах, для которых необходимы исследования. Есть надежда, что в не столь отдаленном будущем человеческий мозг и компьютер будут очень тесно связаны между собой, а результат такого сотрудничества будет мыслить так, как ни один человеческий мозг до этого не делал, и оперировать данными способом, отличным от тех, которыми пользуются известные нам сегодня машины для обработки информации.

20.1.2. Между «механически улучшенным человеком» и «искусственным интеллектом». Человеко-машинный симбиоз как концепция существенно отличается от того, что Норт (North 1954) называл «механически улучшенным человеком». В человеко-машинных системах прошлого оператор отвечал за проявление инициативы, управление, интеграцию и задание критериев оценки. Механические части системы выступали только как продолжения, сначала человеческих рук, а потом и глаз. Конечно, в состав этих систем не входили «организмы разных видов, живущих вместе». Там присутствовал только один вид организма – человек, – а все остальное нужно было только для того, чтобы ему помогать.

Конечно, в каком-то смысле любая система, созданная человеком, предназначена для того, чтобы помогать ему или другим людям за пределами этой системы. Если мы рассмотрим человека-оператора внутри системы, то увидим, что за последние несколько лет в некоторых областях техники произошли фантастические изменения. «Механическое продолжение» открыло путь к замещению человека, к автоматизации, а на тех людей, что остаются, возлагается преимущественно задача помогать, а не получать помощь. В некоторых случаях, особенно в крупных компьютеризированных информационных системах и системах управления, операторы выполняют в основном те функции, автоматизировать которые оказалось невозможно. Такие системы (Норт назвал бы их «машинами, дополненными человеческими возможностями») не являются симбиотическими. Это «полуавтоматические» системы, т. е. такие, которые задумывались как полностью автоматизированные, но не смогли достичь поставленной цели.

Возможно, симбиоз человека и компьютера не является конечной парадигмой для сложных технических систем. Вполне вероятно, что на определенном этапе электронные или химические «машины» превзойдут человеческий мозг по большинству функций, которые сейчас мы считаем исключительно его прерогативой. Даже сейчас программа Гелернтера для доказательства теорем по планиметрии на IBM-704 решает задачу пример-

но с такой же скоростью, с какой студенты школы Бруклина, и совершает аналогичные ошибки (Gelernter 1959). На самом деле существует несколько программ для доказательства теорем, принятия решений, для игры в шахматы и распознавания образов – слишком много, чтобы все перечислять (Bernstein and Roberts 1958; Bledsoe and Browning 1959; Dinneen 1955; Farley and Clark 1954; Friedberg 1958; Gilmore and Savell 1959; Newell 1955; Newell and Shaw 1957; Newell et al. 1958; Selfridge 1958; Shannon 1950; Sherman 1959), – способных соперничать с интеллектуальными возможностями человека в ограниченном числе областей. А «универсальный решатель задач», созданный Ньюэллом, Саймоном и Шоу, может снять некоторые ограничения. В общем, лучше не спорить с (другими) энтузиастами искусственного интеллекта и допустить, что в отдаленном будущем мы уступим пальму первенства в области умственной деятельности машинам. Тем не менее в течение довольно длительного времени основные интеллектуальные открытия будут совершаться именно в процессе совместной работы людей и компьютеров. Многофункциональная исследовательская группа, изучающая будущие задачи, которые предстоит решить военно-воздушным силам в области исследований и разработок, подсчитала, что такой уровень развития искусственного интеллекта, при котором машины будут способны самостоятельно рассуждать или принимать решения военного значения, наступит не раньше 1980 года. Нам остается, скажем, 5 лет на разработку человеко-машинного симбиоза и 15 лет на его использование. Быть может, не 15, а 10 или 500, но эти годы в интеллектуальном плане будут самыми творческими и захватывающими в истории человечества.

20.2. Цели симбиоза человека и машины

Сегодняшние компьютеры предназначены главным образом для решения сформулированных заранее задач или для обработки данных в соответствии с предопределенными процедурами. Последовательность расчетов может изменяться в зависимости от результатов, полученных по ходу вычислений, но все возможные варианты должны быть определены загодя. (Если возникает непредвиденная ситуация, весь процесс останавливается в ожидании необходимой модификации программы.) Иногда требование предварительной формулировки или определения – не такой уж и существенный недостаток. Часто говорят, что программирование вычислительной машины заставляет четко мыслить, что это дисциплинирует сам процесс мышления. Если пользователь может заранее тщательно обдумать свою задачу, то симбиотическая связь с вычислительной машиной и не нужна.

Однако многие задачи, которые можно было бы продумать заранее, слишком сложны для этого. Их было бы проще и быстрее решить интуитивно понятным методом проб и ошибок, когда в сотрудничестве с компьютером выявлялись бы ошибки в рассуждениях или обнаруживались бы неожиданные повороты в процессе поиска решения. Другие задачи просто невозможно сформулировать без помощи вычислительной техники. Пуанкаре предугадал

чувство разочарования у важной группы потенциальных пользователей компьютеров, когда сказал: «Дело не в том, каков ответ, а в том, каков вопрос». Одной из главных целей человеко-машинного симбиоза является эффективное привлечение вычислительных машин к постановке технических задач.

Другая важная цель тесно связана с первой. Она заключается в том, чтобы эффективно включить вычислительные машины в процесс мышления, который должен происходить в режиме «реального времени», т. е. времени, которое течет слишком быстро, чтобы иметь возможность использовать компьютеры традиционным способом. Представьте, например, что вы пытаетесь управлять боем с помощью компьютера по следующему сценарию. Сегодня вы формулируете задачу. Завтра вы объясняете ее программисту. На следующей неделе компьютер тратит 5 минут на ассемблирование вашей программы и 47 секунд на вычисление ответа. Вы получаете лист бумаги длиной 20 футов, весь заполненный числами, но эти цифры предлагают не конечное решение, а лишь тактику, нуждающуюся в проверке посредством моделирования. Очевидно, что бой закончится еще до того, как начнется второй шаг его планирования. Чтобы рассуждать во взаимодействии с компьютером так же, как вы обдумываете задачу вместе с коллегой, чьи знания дополняют ваши, понадобится гораздо более тесная связь между человеком и машиной, чем подразумевается в примере выше. Сегодня такая связь еще невозможна.

20.3. Компьютер должен участвовать в формулировке задач и процессе мышления в реальном времени

В предыдущей части молчаливо предполагалось, что если бы машины для обработки данных можно было эффективно включить в мыслительный процесс, то функции, которые они способны выполнять, существенно улучшили или дополнили бы процесс мышления и решения задач. Это предположение требует обоснования.

20.3.1. Предварительный и неформальный анализ времени и движения в процессе технического мышления. Несмотря на наличие обширной литературы на тему мышления и решения задач, включая всесторонние исследования процесса изобретения, я не смог найти ничего похожего на анализ времени и движения в умственной деятельности человека, принимающего участие в научном или техническом проекте. Поэтому весну и лето 1957 года я посвятил попыткам отследить, что на самом деле делает человек со средним уровнем технического образования на протяжении того времени, которое он, как считается, посвящает работе. Осознавая недостаточность выборки, я все же решил выбрать самого себя на роль испытуемого.

Вскоре стало очевидно, что единственное, чем я занимался, – отчетность, и проект превратился бы в бесконечный регресс, если бы отчеты составлялись с той степенью детальности, которая предусматривалась в исходном плане. Не превратился. Тем не менее я получил картину своей деятельности, которая заставила меня задуматься. Возможно, я не являюсь типичным пред-

ставителем – надеюсь на это, но боюсь, что надежды тщетны.

Около 85 % времени, отведенного на «обдумывание», было потрачено на приведение себя в подходящее умонастроение для размышлений, принятия решений, изучения того, что мне нужно было знать. Гораздо больше времени ушло на поиск или получение информации, чем на ее осмысление. Многие часы ушли на построение графиков, и еще многие на инструктаж помощника, как это делать. Когда с графиками было покончено, взаимосвязи сразу стали очевидными, но прежде графики нужно было построить. В какой-то момент нужно было сравнить шесть экспериментально выведенных определений функции, связывающей разборчивость речи с отношением речь/шум. Все без исключения экспериментаторы использовали разные определения или способы измерения отношения речь/шум. Потребовалось несколько часов вычислений, чтобы привести данные к удобной для сравнения форме. Когда это было сделано, потребовалось всего несколько секунд, чтобы определить то, что мне было нужно.

Если вкратце описать результаты моих исследований, то оказалось, что время, отведенное для «обдумывания», уходило в основном на чисто механические действия, вполне доступные делопроизводителю: поиск, вычисления, построение графиков, преобразования, определение логических или динамических следствий из набора допущений либо гипотез, подготовка к принятию решения или осознанию. Кроме того, выбор, что стоит, а чего не стоит пытаться делать, в неприлично высокой степени определялся соображениями выполнимости рутинных задач, а не интеллектуальными способностями.

Основное предположение, на которое наводят описанные выше открытия, состоит в том, что операции, отнимающие большую часть времени и якобы имеющие целью обдумывание технических деталей, – это как раз те операции, которые машины могут выполнять лучше людей. Серьезные проблемы возникают из-за того, что эти операции должны применяться к разным переменным и в непредвиденных и постоянно изменяющихся последовательностях. И если их можно решить посредством создания симбиотической связи между человеком и машиной, способной быстро извлекать информацию и обрабатывать данные, то представляется очевидным, что такое взаимодействие существенно улучшит процесс мышления.

Тут уместно признать, что мы используем термин «компьютер» для обозначения широкого класса машин: вычислительных, обрабатывающих данные, а также информационно-накопительных и поисковых. Возможности техники данного класса возрастают каждый день. И поэтому делать общие заявления о возможностях класса довольно опасно. Быть может, столь же опасно делать общие заявления о возможностях людей. Тем не менее определенные генотипические различия в способностях людей и компьютеров действительно бросаются в глаза, и они имеют отношение к природе возможного симбиоза человека и компьютера и к потенциальной ценности его достижения.

Как уже не раз было сказано, люди – шумные, узкополосные устройства, но их нервная система обладает большим числом параллельных и одновременно активных каналов. По сравнению с людьми, вычислительные машины работают очень быстро и точно, но они могут выполнить лишь одну или не-

сколько элементарных операций за раз. Люди же в этом плане гибче и способны «программировать себя в зависимости от обстоятельств» на основе недавно полученной информации. Вычислительные машины настроены на решение одной конкретной задачи и ограничены своей «предварительно составленной программой». Для людей естественно говорить на избыточных языках, организованных вокруг односоставных объектов и упорядоченных действий, с использованием 20–60 элементарных символов. Для компьютеров «естественно» говорить на неизбыточных языках, обычно состоящих всего из двух элементарных символов и не имеющих представления ни об односоставных объектах, ни об упорядоченных действиях.

Для строгой корректности эти характеристики должны были бы включать множество квалификаторов. Тем не менее общая картина различий (и, следовательно, потенциальных дополнений), которые эти характеристики представляют, по сути своей верна. Вычислительные машины могут делать легко, быстро и хорошо множество вещей, трудных или невозможных для человека, но и люди способны легко, хорошо, хотя и не так быстро, делать множество вещей, трудных или невозможных для компьютеров. Это наводит на мысль, что симбиотическое взаимодействие, при условии успешной интеграции положительных характеристик людей и машин, имело бы огромную ценность. Конечно, при этом нам нужно преодолеть препятствие в виде разницы в скорости и языке.

20.4. Разделимые функции людей и компьютеров в предполагаемой симбиотической связи

Вполне вероятно, что вклады людей и машин во многие операции окажутся так тесно переплетены, что будет трудно аккуратно разделить их для анализа. Так было бы, если бы, например, при сборе данных для принятия решения и человек, и компьютер находили в своем прошлом опыте подходящие прецеденты и если бы компьютер затем предлагал план действий, согласующийся с интуитивным суждением человека. (В программах для доказательства теорем компьютеры находят прецеденты в опыте, а в системе SAGE предлагают план действий. И это не притянутый за уши пример.) Однако в других операциях вклады людей и машин до некоторой степени допускают разделение.

Конечно, именно люди будут ставить цели и определять мотивацию, по крайней мере на первых порах. Они будут формулировать гипотезы. Они будут задавать вопросы. Они будут продумывать механизмы, процедуры и модели. Они будут помнить, что имярек выполнил работу, возможно, имеющую отношение к делу, в 1947 году, ну, или вскоре после Второй мировой войны, и они будут способны предположить, в каких журналах эта работа могла быть опубликована. В общем, они будут вносить приближенный и несовершенный, но при этом главенствующий, вклад, и они будут определять критерии и производить оценку, вынося суждения о вкладе машин и задавая

общее направление размышлений.

Кроме того, люди будут разбираться с крайне маловероятными ситуациями, если таковые когда-нибудь возникнут. (В современных человеко-машинных системах это одна из важнейших функций человека-оператора. Суммарная вероятность таких крайне маловероятных ситуаций зачастую слишком велика, чтобы ей можно было пренебречь.) Люди будут восполнять упущения в решении задачи или в компьютерной программе в тех случаях, когда у компьютера нет подходящего для возникшей ситуации режима или процедуры.

Со своей стороны, оборудование для обработки информации будет преобразовывать гипотезы в допускающие проверку модели, а затем проверять модели на данных (которые оператор может приблизительно определить или пометить как релевантные, если компьютер предлагает их для одобрения). Машина будет отвечать на вопросы. Она будет имитировать механизмы и модели, выполнять процедуры и отображать результаты оператору. Она будет преобразовывать данные, строить графики (в тех разрезах, которые задаст оператор, или в нескольких разных разрезах, если оператор не уверен, чего он хочет). Машина возьмет на себя интерполяцию, экстраполяцию и трансформирование. Она будет преобразовывать статические уравнения или логические утверждения в динамические модели, так чтобы оператор мог исследовать их поведение. В общем и целом она займется рутинными, служебными операциями, заполняющими интервалы между принятием решений.

Дополнительно компьютер возьмет на себя роль машины для статистического вывода, теории принятия решений или теории игр и будет давать элементарные оценки предложенных планов действий в тех случаях, когда имеется достаточно материала для поддержки формального статистического анализа. Наконец, в его функции будет входить такой объем диагностики, сопоставления с образцами и оценки релевантности, на какой он способен, но в этих областях он, безусловно, будет играть вспомогательную роль.

20.5. Условия для реализации симбиоза между человеком и компьютером

Оборудования для обработки данных, молчаливо предполагавшегося в предыдущем разделе, пока не существует. Компьютерные программы еще не написаны. На пути между несимбиотическим настоящим и чаемым симбиотическим будущим есть еще несколько препятствий. Рассмотрим некоторые из них, чтобы лучше понять, что нужно и каковы шансы это обрести.

20.5.1. Несогласованность скорости людей и компьютеров. Любой современный большой компьютер слишком быстрый и слишком дорогой для совместного с человеком мышления в режиме реального времени. Очевидно, что ради эффективности и экономии компьютер должен разделять свое время между многими пользователями. Системы с разделением времени в настоящее время активно разрабатываются. Есть даже способы органи-

зации, не позволяющие пользователям «вмешиваться» во что-то, кроме их собственных программ.

Разумно предположить, что на горизонте 10–15 лет появится «мыслительный центр», который объединит в себе функции современных библиотек с прогнозируемыми достижениями в области хранения и поиска информации и симбиотическими функциями, описанными выше в этой статье. Картину легко мысленно расширить, включив в нее целую сеть таких центров, соединенных между собой широкополосными линиями связи; пользователи же будут подключаться к ним по арендованным линиям. В такой системе быстродействие компьютеров будет уравновешено, а стоимость гигантских объемов памяти и изощренных программ будет разделяться между многими пользователями.

20.5.2. Требования к аппаратуре памяти. Начиная размышлять о хранении сколько-нибудь заметной доли технической литературы в памяти компьютера, мы приходим к выводу о необходимости миллиардов бит и, если ситуация кардинально не изменится, о миллиардах долларов.

Первое, с чем приходится столкнуться, – невозможность хранить все научно-технические статьи в памяти компьютера. Мы можем сохранить части, которые можно кратко реферировать, – количественные данные и ссылки на литературу, но не все целиком. Книги принадлежат к числу самых удивительных творений человека, и их функциональное значение сохранится в контексте симбиоза человека и компьютера. (Хочется надеяться, что компьютер сможет ускорить поиск, доставку и возврат книг.)

Второй момент – тот факт, что очень важный раздел памяти будет постоянным: частично нестираемой, а частично опубликованной памятью. Компьютер сможет записывать в нестираемую память, а затем читать сколько угодно раз, но не сможет ее стереть. (Он может также перезаписывать, преобразовывая все нули в единицы, как бы отмечая то, что было записано ранее.) Опубликованная память будет памятью, доступной «только для чтения». Она включается в компьютер уже структурированной. Компьютер сможет обращаться к ней многократно, но не сможет изменять ее. Эти типы памяти будут приобретать все большую важность по мере увеличения самих компьютеров. Их можно сделать более компактными, чем память на магнитных сердечниках, на тонких пленках и даже на магнитных лентах, а стоить они будут гораздо меньше. Основная техническая проблема будет связана со схемами выборки.

Что касается других требований к памяти, то мы можем рассчитывать на дальнейшее развитие обыкновенных вычислительных машин для научных расчетов и бизнеса. В перспективе элементы памяти могут стать такими же быстрыми, как обрабатывающие (логические) элементы. Это оказало бы революционное воздействие на конструирование компьютеров.

20.5.3. Требования к организации памяти. В идее симбиоза человека и машины неявно подразумевается требование о том, что информация должна быть доступна как по имени, так и по образцу и что процедура доступа к ней должна быть гораздо быстрее последовательного поиска. По меньшей мере половина проблемы организации памяти, по-видимому, скрыта в про-

цедуре хранения. А большая часть оставшейся половины видится связанной с задачей распознавания образов в механизме хранения или на носителе. Подробное обсуждение этих проблем выходит за рамки настоящей статьи. Однако краткое описание многообещающей идеи «trie-памяти»[1] поможет оценить общую природу ожидаемых разработок.

Trie-память названа так ее изобретателем, Фредкиным (Fredkin 1960), потому что предназначена для ускорения информационного поиска и потому что эта ветвящаяся структура напоминает дерево (tree). В большинстве распространенных систем памяти хранятся функции от аргументов по адресам, определяемым аргументами. (В некотором смысле аргументы в них вообще не хранятся. Но в другом, более близком к реальности смысле в структуре памяти хранятся все возможные аргументы.) С другой стороны, в trie-памяти хранятся как функции, так и аргументы. Аргумент вносится в память первым, по одному символу за раз, начиная со стандартного начального регистра. В каждом регистре аргументов имеется по одной ячейке для каждого символа ансамбля (например, две для информации, закодированной в двоичной форме), и в каждой ячейке символа имеется место для хранения адреса следующего регистра. Аргумент хранится в последовательности адресов, каждый из который сообщает, где искать следующий. В конце аргумента находится специальный маркер «конца аргумента». Затем следуют указатели на функцию, которая может храниться одним из нескольких способов, причем наиболее эффективным часто является либо еще одна trie-структура, либо «списковая структура».

Схема trie-памяти неэффективна, когда объем памяти мал, но эффективность использования располагаемого объема памяти возрастает по мере увеличения этого объема. Перечислим привлекательные стороны этой схемы: 1) Процесс поиска очень прост. Получив аргумент, начать со стандартного начального регистра, содержащего первый символ, и выбрать адрес второго регистра. Перейти ко второму регистру и выбрать адрес третьего и т. д. 2) Если начальные символы двух аргументов совпадают, то для хранения этих символов используется одно и то же место в памяти. 3) Длины аргументов необязательно должны быть одинаковыми, и их не нужно задавать заранее. 4) Место в памяти не резервируется и не используется, пока аргумент не будет фактически сохранен. Trie-структура создается по мере добавления объектов в память. 5) Функцию можно использовать как аргумент другой функции, которая, в свою очередь, может являться аргументом третьей. Так, например, войдя в структуру с аргументом «matrix multiplication», можно найти в компьютере полную программу умножения матриц. 6) Исследуя память на заданном уровне, можно узнать, какие были сохранены объекты, похожие вплоть до этого уровня. Например, если ссылки на «Egan, J. P.» нет, то достаточно сделать один или два шага назад, чтобы найти «Egan, James». <…>

Описанные выше свойства включают не все желаемые, но благодаря им

---

[1] *Trie* – средний слог слова *retrieval* (поиск). По-другому называется «префиксным деревом». Произносится «три», но некоторые предпочитают произношение «трай», чтобы на слух отличить от tree (дерево). – *Прим. перев.*

компьютер становится ближе к операторам, которые, как и все люди, склонны обозначать вещи, называя их или указывая на них.

20.5.4. Языковая проблема. Принципиальная непохожесть человеческих и компьютерных языков может стать серьезным препятствием на пути к истинному симбиозу. Обнадеживает, однако, наблюдение за тем, какие гигантские шаги были сделаны в направлении создания интерпретирующих программ и особенно ассемблеров и компиляторов, таких как Fortran, цель которых – приблизить компьютеры к языковым формам, понятным человеку. В работе «Язык обработки информации» (Shaw et al. 1958) представлена еще одна линия сближения. А в языке Algol и родственных системах люди доказывают свою гибкость, приспосабливаясь к стандартным формулам представления и выражения, которые легко транслируются на машинный язык.

Однако для взаимодействия между людьми и компьютерами в режиме реального времени понадобится использовать дополнительный и довольно сильно отличающийся принцип обмена информацией и управления. Понять идею можно, сравнив инструкции, которые обычно обращены к разумным людям, с инструкциями, которые принято давать компьютерам. В последних точно описываются все шаги и их последовательность. А первые содержат какую-то информацию о побудительных мотивах или стимулах и критерий, по которому исполнитель может судить, когда задание можно считать выполненным. Короче говоря, инструкции компьютеру определяют порядок выполнения, а инструкции человеку – цели.

Людям естественнее и проще мыслить в терминах целей, а не порядка действий. Да, обычно они что-то знают о направлении путешествия или об общих принципах работы, но немного найдется таких, которые заранее составляют точный план поездки. Кто, например, отправится из Бостона в Лос-Анджелес с подробным описанием всего маршрута? Вместо этого, мы можем сказать, перефразируя Винера, что, направляясь в Лос-Анджелес, люди непрерывно пытаются уменьшить расстояние, оставшееся до попадания в облако смога.

К инструктированию компьютера посредством специфицирования целей ведут два пути. Первый – самоорганизующиеся программы решения задач путем восхождения на гору. Второй – объединение в реальном времени заранее запрограммированных сегментов и замкнутых подпрограмм, которые человек-оператор может указать и привести в действие просто по имени.

На первом пути имеются обнадеживающие исследовательские работы. Ясно, что, действуя в рамках необременительных ограничений предопределенных стратегий, компьютеры смогут со временем придумывать и упрощать собственные процедуры достижения поставленных целей. До сих пор достижения были не особенно впечатляющими – только «демонстрация принципа». Тем не менее эти работы могут иметь далеко идущие последствия.

Второй путь проще и, на первый взгляд, допускает более быструю реализацию, но пока пребывает в небрежении. Trie-память Фредкина предлагает многообещающую парадигму. Со временем мы, возможно, увидим серьезные попытки разработать компьютерные программы, которые можно связывать друг с другом, как слова и фразы в разговоре, дабы произвести

вычисление или управляющее воздействие, необходимое в данный момент. Сдерживают такие усилия разве что опасения, что они не дадут ничего особо ценного в контексте существующих компьютеров. Было бы неблагодарным занятием разрабатывать язык до того, как появятся вычислительные машины, способные осмысленно реагировать на него.

20.5.5. Оборудование ввода-вывода. Наименее передовой с точки зрения требований к симбиозу человека и компьютера видится та часть обработки данных, которая связана с оборудованием ввода-вывода, или, как это представляется человеку-оператору, – дисплеи и органы управления. Сразу же после этого утверждения уместно добавить пояснительные комментарии, потому что в инженерном отношении оборудование для высокоскоростного ввода и извлечения информации просто великолепно и потому что в таких исследовательских учреждениях, как лаборатория Линкольна, разработаны весьма изощренные методы отображения и управления. Но в общем и целом общедоступные компьютеры почти не предлагают средств человеко-машинного взаимодействия, более эффективных, чем электрическая пишущая машинка.

Средства отображения, по-видимому, находятся в несколько лучшем положении, чем органы управления. Многие компьютеры умеют выводить графики на экране осциллографа, а некоторые даже пользуются замечательными возможностями, символьными и графическими, которые дает знакопечатающая ЭЛТ (характрон). Но, насколько я знаю, никто даже близко не подошел к гибкости и удобству карандаша и блокнота или мела и доски, которыми люди пользуются в научных дискуссиях.

1) Встроенные в стол дисплеи и органы управления. Разумеется, для эффективного человеко-машинного взаимодействия необходимо, чтобы человек и компьютер могли рисовать графики и картинки и писать заметки и уравнения на одном и том же дисплее. Человек должен иметь возможность представить функцию компьютеру, пусть приближенно, но быстро, – нарисовав ее график. Компьютер должен прочесть написанное человеком, быть может, при условии что текст написан отчетливыми прописными буквами, и сразу же в позиции каждого рукописного символа вывести соответствующий ему символ определенным шрифтом. Имея такое устройство ввода-вывода, оператор быстро научится писать или печатать так, чтобы машина могла разобрать написанное. Он мог бы составлять инструкции или подпрограммы, записывать их в нужном формате и проверять перед окончательной отправкой в главную память компьютера. Он мог бы даже определять новые символы, как Гилмор и Сэйвелл (Gilmore and Savell 1959) проделали в лаборатории Линкольна, и представлять их компьютеру непосредственно. Он мог бы грубо набросать формат таблицы и позволить компьютеру аккуратно оформить ее. Он мог бы исправить машинные данные, инструктировать машину посредством блок-схем и вообще взаимодействовать с машиной почти так же, как с другим инженером, только «другой инженер» в этом случае был бы опытным чертежником, мгновенным вычислителем, мастером мнемонического запоминания и носителем многих других полезных качеств в одном лице.

2) Настенный компьютерный дисплей. В некоторых технических систе-

мах несколько человек несут ответственность за управление устройствами с пересекающимся поведением. Какая-то информация должна быть представлена одновременно всем, предпочтительно на общей сетке, чтобы можно было координировать их действия. Другая информация интересна только одному-двум операторам. Если бы вся информация отображалась на одном дисплее для всех, то получилась бы неразборчивая мешанина. Информация должна выводиться компьютером, потому что рисование вручную происходит слишком медленно и не поспевает за ситуацией.

Описанная только что проблема критична даже сейчас, и нет сомнений, что со временем ее критичность будет только возрастать. Некоторые проектировщики убеждены, что дисплеи с желаемыми характеристиками можно сконструировать с помощью импульсных источников света и проекционных экранов, основанных на принципе оптической модуляции.

По мнению большинства тех, кто размышлял над проблемой, большой дисплей следует дополнить индивидуальными устройствами отображения и управления. Они дали бы возможность операторам изменять изображение на настенном дисплее, не вставая со своих мест. Для некоторых целей было бы желательно, чтобы операторы могли взаимодействовать с компьютером с помощью вспомогательных дисплеев и, быть может, даже с помощью настенного. Практически реализуемой видится, по крайней мере, одна схема организации такого взаимодействия.

Большой настенный дисплей и ассоциированная с ним система, безусловно, имеют прямое отношение к симбиотической кооперации между компьютером и группой людей. Лабораторные эксперименты раз за разом показывали, что неформальная параллельная организация работы операторов, координирующих свои действия посредством обращения к большому ситуационному дисплею, имеет важные преимущества перед более широко употребительной организацией, когда операторы сидят за индивидуальными консолями и стараются согласовать свои действия при посредничестве компьютера. Это одна из нескольких проблем командной работы операторов, нуждающаяся в тщательном изучении.

3) Автоматическая генерация и распознавание речи. Насколько желательно и практически осуществимо речевое взаимодействие между операторами и вычислительными машинами? Этот двойной вопрос задают всякий раз при обсуждении сложных систем обработки данных. Инженеры, которые живут и работают с компьютерами, занимают осторожную позицию по поводу желательности такого взаимодействия. Инженеры, имеющие опыт в области автоматического распознавания речи, скептически настроены по поводу его практической осуществимости. Но тем не менее интерес к идее разговора с вычислительными машинами не угасает. В значительной мере он проистекает из осознания того, что вряд ли удастся оторвать генерала или президента корпорации от работы, чтобы обучить его печатать на машинке. Если мы хотим, чтобы топ-менеджеры когда-нибудь стали использовать машины напрямую, то имеет смысл предоставить более естественные средства коммуникации, пусть даже они будут стоить дорого.

Предварительный анализ стоящих перед президентом корпорации задач

и распорядка его дня показывает, что симбиотическая связь с компьютером может заинтересовать его только как развлечение. Ситуация в коммерческих делах развивается настолько медленно, что хватает времени на брифинги и конференции. Поэтому напрямую взаимодействовать с компьютерами в офисах компаний, скорее всего, будут только специалисты-компьютерщики.

С другой стороны, военачальник с большей вероятностью сталкивается с необходимостью принимать критические решения в крайне ограниченное время. Можно, конечно, чрезмерно драматизировать концепцию десятиминутной войны, но было бы опасно рассчитывать, что для принятия критического решения окажется больше десяти минут. Таким образом, по мере роста возможностей и сложности наземной среды и центров управления в военных системах с большой вероятностью может возникнуть реальная потребность в генерации и распознавании речи компьютерами. Конечно, если бы надежное и доступное оборудование уже было разработано, оно нашло бы применение.

Что до практической реализуемости, то генерация речи представляет не столь серьезные технические проблемы, как ее автоматическое распознавание. Коммерческий цифровой электронный вольтметр уже умеет вслух произносить показания, цифра за цифрой. На протяжении восьми или десяти лет Bell Telephone Laboratories, Королевский технический институт (Стокгольм), Научно-исследовательский институт связи (Кристчёрч), лаборатория Хаскинса и Массачусетский технологический институт в работах Dunn (1950), Fant (1959), Lawrence (1956), Cooper et al. (1952), Stevens et al. (1953) демонстрировали последовательные поколения автоматических генераторов разборчивой речи. Недавняя работа в лаборатории Хаскинса завершилась разработкой цифрового кода, пригодного для использования в вычислительных машинах, который позволяет автоматическому голосу произносить разборчивую связную речь (Liberman et al. 1959).

Практическая реализуемость автоматического распознавания речи сильно зависит от размера распознаваемого словаря и от разнообразия говорящих и акцентов, которые необходимо понимать. 98-процентная правильность распознавания естественно произносимых десятичных цифр была несколько лет назад продемонстрирована в Bell Telephone Laboratories и в лаборатории Линкольна (Davis et al. 1952, Forgie and Forgie 1959). Что касается масштабирования размера словаря, мы можем сказать, что автоматический распознаватель отчетливо произносимых букв и цифр почти наверняка можно построить на базе имеющихся знаний. Поскольку необученные операторы могут читать по меньшей мере с такой же скоростью, как обученные печатать, подобное устройство стало бы удобным инструментом почти в любом вычислительном центре.

Но для взаимодействия в реальном масштабе времени на действительно симбиотическом уровне, вероятно, был бы нужен словарь, содержащий примерно 2000 слов, например 1000 слов базового английского языка и 1000 технических терминов. И вот это уже трудная проблема. По общему мнению акустиков и лингвистов, построить распознаватель на 2000 слов прямо сейчас невозможно. Однако существуют организации, которые с радостью взя-

лись бы за разработку автоматического распознавателя такого словаря на горизонте пяти лет. Они, правда, оговаривают, что речь должна быть четкой, как при диктовке, и без необычного акцента.

Хотя подробное обсуждение методов автоматического распознавания речи выходит за рамки настоящего обзора, уместно будет отметить, что вычислительные машины играют преобладающую роль в разработке таких распознавателей. Они придали тот импульс, который объясняет нынешний оптимизм или, точнее, оптимизм, который сейчас наблюдается в некоторых кругах. Два-три года назад казалось, что до реализации автоматического распознавания словарей сколько-нибудь большого размера должно пройти не менее 10–15 лет, что для этого необходимо дождаться постепенного накопления знаний в области акустики, фонетики, лингвистики и психологических аспектов речевой коммуникации. Однако теперь многие видят перспективу ускорения сбора знаний с помощью компьютерной обработки речевых сигналов, и не так уж мало людей, полагающих, что изощренные компьютерные программы смогут хорошо обрабатывать речевые образы даже без солидных знаний о речевых сигналах и процессах. Сведение обоих соображений воедино позволяет снизить оценку времени, необходимого для достижения практически полезного уровня распознавания речи, до пяти лет – тех самых, о которых было сказано чуть выше.

21 Рекурсивные функции символических выражений и их вычисление машиной (1960)

Джон Маккарти

Джон Маккарти (1927–2011) изучал математику в Калифорнийском технологическом институте, а затем писал докторскую диссертацию в Принстоне. Во время преподавания в Дартмутском колледже в 1956 году Маккарти организовал летнюю школу по «Искусственному интеллекту» (название придумано им). В проспект конференции (McCarthy 1960), составленный совместно Маккарти, Шенноном, Марвином Мински (тогда младшим научным сотрудником в Гарварде) и исследователем из IBM Натаниэлем Рочестером, были включены следующие темы: (1) «Автоматические компьютеры» (как писать программы, моделирующие «высшие функции человеческого мозга»); (2) «Как запрограммировать использование языка компьютером»; (3) «Нейронные сети» (цитируя Мак-Каллока и Питтса); (4) «Теория шкалы вычислений» («теория сложности функций»); (5) «Самосовершенствование»; (6) «Абстракции» и (7) «Случайность и творчество». Все эти темы активно исследуются и сегодня!

Маккарти перешел в МТИ, где сыграл важную роль в рождении операционных систем с разделением времени (глава 23). Там он изобрел язык программирования Lisp как инструмент для символических рассуждений, которые, как он предвидел, станут ключом к прогрессу в области ИИ. Это был

дерзкий шаг – напрямую заимствовать аппарат лямбда-исчисления, которое Алонзо Чёрч разрабатывал как математической язык для решения проблемы разрешимости (Entscheidungsproblem) Гильберта. В своем первоначальном воплощении Lisp был чисто интерпретируемым языком; на создание компиляторов ушло несколько лет. Маккарти вынужден был разработать технику сборки мусора для управления памятью, иначе даже небольшие программы невозможно было выполнить на машинах того времени, имевших ограниченную память. Lisp не только использовался, но и оказался весьма влиятельным; Маккарти принимал участие в проектировании Algol 60, первого широко используемого языка общего назначения с поддержкой рекурсии, а Lisp оказал влияние на проектирование всех последующих языков функционального программирования.

В 1962 году Маккарти перешел в Стэнфорд, где основал Лабораторию искусственного интеллекта (SAIL), рассадник многочисленных авторитетных исследований по ИИ. Не только сам Маккарти, но и еще пятнадцать ученых, связанных с SAIL, были удостоены премии Тьюринга. На протяжении всей своей карьеры он являл почти уникальное сочетание глубоко укоренившегося уважения к формальным математическим и логическим основаниям науки с неуемным желанием писать работоспособный код, имитирующий различные аспекты человеческого мышления.

21.1. Введение

Система программирования Lisp (LISt Processor) была разработана для компьютера IBM 704 Группой искусственного интеллекта в МТИ. Система была создана, чтобы облегчить эксперименты с гипотетической системой Advice Taker, которую можно было запрограммировать для обработки как декларативных, так и императивных предложений и которая демонстрировала «здравый смысл» в ходе выполнения этих программ. Первоначальный проект (McCarthy 1961) Advice Taker был выполнен в ноябре 1958 года. Основным требованием было наличие системы программирования для манипуляции выражениями, представляющими формализованные декларативные и императивные предложения, а в качестве цели ставилась возможность дедуктивного вывода.

В ходе своего развития система Lisp прошла несколько стадий упрощения и в конечном итоге стала основываться на схеме представления частично рекурсивных функций определенного класса символических выражений. Это представление независимо от архитектуры IBM 704 или любой другой ЭВМ. Представляется целесообразным начать описание системы с класса выражений, именуемых S-выражениями, и функций, именуемых S-функциями.

В этой статье мы вначале опишем формализм для рекурсивного определения функций. Мы верим, что этот формализм имеет преимущества и в качестве языка программирования, и как средство развития теории вычислений. Далее мы опишем S-выражения и S-функции и затем универсаль-

ную S-функцию *apply*, которая, с теоретической точки зрения, может рассматриваться как универсальная машина Тьюринга, а с практической – как интерпретатор. Затем мы опишем представление S-выражений в памяти IBM 704 с помощью списковых структур, подобных тем, что использовали Ньюэлл и Шоу (Newell and Shaw 1957), а также программное представление S-функций. После этого мы упомянем основные возможности системы программирования Lisp для IBM 704. Затем перейдем к другому способу описания вычислений с помощью символических выражений и, наконец, опишем реализацию интерпретации блок-схем с помощью рекурсивных функций.

В других статьях мы надеемся описать некоторые из символических вычислений, для которых Lisp использовался, а также привести некоторые приложения нашего формализма рекурсивных функций к математической логике и к проблеме автоматического доказательства теорем.

21.2. Функции и определения функций

Нам понадобятся некоторые математические идеи и обозначения, касающиеся функций вообще. Большая их часть хорошо известна, но идея *условного выражения*, вероятно, нова, а использование условных выражений позволяет определять функции рекурсивно новым, удобным способом.

a. *Частичные функции*. Частичной функцией называется функция, определенная только на части своей области определения. Частичные функции естественно возникают при определении функций через вычисления, поскольку при некоторых значениях аргументов вычисление, определяющее значение функции, может не завершаться. Однако некоторые из наших элементарных функций будут определены как частичные.

b. *Пропозициональные выражения и предикаты*. Пропозициональным выражением называется выражение, принимающее значения T (истина) и F (ложь). Мы предполагаем, что читатель знаком с пропозициональными связками $\wedge$ («и»), $\vee$ («или») и $\neg$ («не»). Ниже приведены примеры типичных пропозициональных выражений:

$$x < y;$$

$$(x < y) \wedge (b = c);$$

$$x \text{ – простое число.}$$

Предикатом называется функция, вырабатывающая значения истинности T или F.

c. *Условные выражения*. Зависимость значений истинности от величин других видов выражается в математике через предикаты, а зависимость значений истинности от других значений истинности – через логические связки. Однако нотация для символического выражения зависимости величин других типов от значений истинности неадекватна, поэтому для описания этих зависимостей обычно используются английские слова и фразы в текстах, где прочие зависимости описываются символически. Например, функция $|x|$ обычно определяется словесно.

Условные выражения – это средство для выражения зависимости величин от пропозициональных величин. Условные выражения имеют вид $(p_1 \to e_1, \ldots, p_n \to e_n)$, где p – пропозициональное выражение, а e – выражение любого вида. Эту запись можно прочитать как «Если p_1, то e_1, иначе если p_2, то e_2, ..., иначе если p_n, то e_n», или как «из p_1 следует e_1, ..., из p_n следует e_n».

Приведем теперь правила, позволяющие установить, определено ли значение выражения $(p_1 \to e_1, \ldots, p_n \to e_n)$, и если да, то каково это значение. Проверяем значения p слева направо. Если p со значением T встретилось раньше любого p, чье значение не определено, тогда значением условного выражения становится значение соответствующего e (если оно определено). Если какое-либо неопределенное p встретилось раньше истинного p, или если все p ложны, или если e, соответствующее первому истинному p, не определено, то значение условного выражения не определено. Приведем примеры.

$$(1 < 2 \to 4, 1 \geqslant 2 \to 3) = 4$$
$$(2 < 1 \to 4, 2 > 1 \to 3, 2 > 1 \to 2) = 3$$
$$(2 < 1 \to 4, T \to 3) = 3$$
$$(2 < 1 \to \frac{0}{0}, T \to 3) = 3$$
$$(2 < 1 \to 3, T \to \frac{0}{0}) \text{ не определено}$$
$$(2 < 1 \to 3, 4 < 1 \to 4) \text{ не определено}$$

В определениях ниже представлены некоторые простейшие применения условных выражений:

$$|x| = (x < 0 \to -x,\ T \to x);$$
$$\delta_{ij} = (i = j \to 1,\ T \to 0);$$
$$sgn(x) = (x < 0 \to -1,\ x = 0 \to 0,\ T \to 1).$$

d. *Рекурсивное определение функций.* Используя условные выражения, мы можем без циклов определять функции с помощью формул, содержащих сами определяемые функции. Например:

$$n! = (n = 0 \to 1,\ T \to n \cdot (n - 1)!).$$

Используя эту формулу для вычисления 0!, мы получаем ответ 1, поскольку в этом случае ветка условного выражения, содержащая бессмысленное выражение $0 \cdot (0 - 1)!$, не проходится. Вычисление 2! в соответствии с этим определением протекает следующим образом:

$$\begin{aligned} 2! &= (2 = 0 \to 1,\ T \to 2 \cdot (2 - 1)!) \\ &= 2 \cdot 1! \\ &= 2 \cdot (1 = 0 \to 1,\ T \to 1 \cdot (1 - 1)!) \\ &= 2 \cdot 1 \cdot 0! \\ &= 2 \cdot 1 \cdot (0 = 0 \to 1,\ T \to 0 \cdot (0 - 1)!) \\ &= 2 \cdot 1 \cdot 1 \\ &= 2. \end{aligned}$$

Теперь приведем еще два приложения рекурсивного определения функций. Наибольший общий делитель $\gcd(m, n)$ двух натуральных чисел m и n вычисляется с помощью алгоритма Евклида. Алгоритм выражается следующим рекурсивным определением:

$$\begin{aligned}\gcd(m, n) = (&m > n \to \gcd(n, m),\\ &\mathrm{rem}(n, m) = 0 \to m,\\ &T \to \gcd(\mathrm{rem}(n, m), m)),\end{aligned}$$

где $\mathrm{rem}(n, m)$ – остаток от деления n на m.

Алгоритм Ньютона приближенного вычисления квадратного корня из числа a с начальным приближением x и условием остановки $|y^2 - a| < \varepsilon$ можно записать в виде:

$$\mathrm{sqrt}(a, x, \varepsilon) = \left(|x^2 - a| < \varepsilon \to x, T \to \mathrm{sqrt}\left(a, \frac{1}{2}\left(x + \frac{a}{x}\right), \varepsilon\right)\right).$$

Возможно также одновременное рекурсивное определение нескольких функций, и мы будем пользоваться этим, если потребуется. Нет гарантии, что вычисление, описываемое рекурсивным определением, когда-либо завершится. Например, вычисление $n!$ в соответствии с нашим определением завершится, только если n – неотрицательное целое число. Если вычисление не завершается, то функция должна считаться неопределенной при данных аргументах.

Пропозициональные связки сами могут быть определены с помощью условных выражений. Запишем:

$$\begin{aligned}p \wedge q &= (p \to q, T \to F);\\ p \vee q &= (p \to T, T \to q);\\ \sim p &= (p \to F, T \to T);\\ p \supset q &= (p \to q, T \to T).\end{aligned}$$

Нетрудно убедиться в правильности таблиц истинности для правых частей уравнений. Если учесть ситуацию, когда p или q могут быть не определены, то связки $\wedge$ и $\vee$ оказываются некоммутативными. Например, если p ложно, а q не определено, то в соответствии с определением выше $p \wedge q$ ложно, но $q \wedge p$ не определено. Для наших целей эта некоммутативность желательна, т. к. p вычисляется первым, и в случае, если p ложно, q не вычисляется. Если вычисление p не завершается, то мы никогда не перейдем к вычислению q. Далее мы будем использовать пропозициональные связки именно в таком смысле.

е. *Функции и формы.* Слово «функция» в математике, за исключением математической логики, обычно используют неточно и применяют ее к таким формам, как $y^2 + x$. Так как позже мы будем использовать в вычислениях выражения для функций, необходимо разграничить функции и формы и ввести нотацию для этого разграничения. Само разграничение и нотация для его

описания, от которой мы незначительно отклонимся, определены в работе Church (1941).

Пусть f – выражение, обозначающее функцию двух целых переменных. Запись $f(3, 4)$ должна иметь смысл, и значение этого выражения должно быть определено. Выражение $y^2 + x$ не удовлетворяет этому условию; $y^2 + x(3,4)$ не соответствует общепринятой нотации, и если мы попытаемся вычислить его, то будет не понятно, чему равен результат: 13 или 19. Чёрч называл выражение вида $y^2 + x$ формой. Форма может быть преобразована в функцию, если удастся определить соответствие между переменными, встречающимися в форме, и упорядоченным списком аргументов желаемой функции. Это достигается с помощью λ-нотации Чёрча.

Пусть $\mathcal{E}$ – форма от переменных $x_1, \ldots, x_n$, тогда $\lambda((x_1, \ldots, x_n), \mathcal{E})$ – функция n переменных, значение которой определяется подстановкой аргументов вместо переменных $x_1, \ldots, x_n$ в $\mathcal{E}$ именно в таком порядке с последующим вычислением полученного выражения. Например, $\lambda((x, y), y^2 + x)$ – функция двух переменных, и $\lambda((x, y), y^2 + x)(3, 4) = 19$.

Переменные, встречающиеся в списке переменных λ-выражения, являются связанными, подобно переменным интегрирования в определенном интеграле. То есть мы можем изменять имена связанных переменных в выражении функции без изменения значения выражения, при условии что выполняется одно и то же изменение для каждого вхождения переменной и переменные, бывшие ранее различными, не отождествляются. Таким образом, $\lambda((x, y), y^2 + x)$, $\lambda((u, v), v^2 + u)$ и $\lambda((y, x), x^2 + y)$ означают одну и ту же функцию.

Мы часто будем использовать выражения, в которых одни переменные связаны λ-выражением, а другие – нет. Такие выражения могут рассматриваться как определение функции с параметрами. Несвязанные переменные называются свободными переменными.

Подходящая нотация, различающая функции и формы, позволяет недвусмысленно трактовать функции от функций. Не хотелось бы слишком далеко отступать от темы, включая сюда примеры, но ниже мы будем использовать функции с функциями в качестве аргументов. <...>

21.3. Рекурсивные функции символических выражений

Сначала мы определим класс символических выражений в терминах упорядоченных пар и списков. Затем определим пять элементарных функций и предикатов и выстроим из них с помощью композиции, условных выражений и рекурсивных определений расширенный класс функций, к которому приведем ряд примеров. Затем мы покажем, как эти функции сами могут быть выражены в виде символических выражений, и определим универсальную функцию *apply*, которая позволит нам по выражению данной функции вычислять ее значение для заданных аргументов. И наконец, мы определим несколько функций с функциями в качестве аргументов и приведем несколь-

ко полезных примеров.

а. *Класс символических выражений.* Определим S-выражения (S – от symbolic, «символическая»). Они образуются путем использования специальных знаков

.
)
(

и бесконечного множества различимых атомарных символов. Для атомарных символов мы будем использовать строки заглавных латинских букв и цифр с включением одиночных пробелов. Примеры атомарных символов:

A
ABA
APPLE PIE NUMBER 3

Есть две причины для отказа от обычной математической практики использования одиночных букв для атомарных символов. Во-первых, в компьютерных программах часто требуются сотни различных символов, которые должны быть сформированы из 47 знаков, печатаемых компьютером IBM 704. Во-вторых, английские слова и фразы для обозначения атомарных сущностей удобны с точки зрения запоминания. Символы атомарны в том смысле, что их внутренняя структура как последовательности знаков игнорируется. Мы предполагаем только, что различные символы различимы.

Таким образом, S-выражения определяются следующим образом:

1. Атомарные символы являются S-выражениями.
2. Если e_1 и e_2 – S-выражения, то $(e_1 \cdot e_2)$ – также S-выражение.

Примеры S-выражений:

AB
(A · B)
((AB · C) · D)

Стало быть, S-выражение – это просто упорядоченная пара, членами которой могут быть атомарные символы или более простые S-выражения. Мы можем представить список произвольной длины в терминах S-выражений следующим образом. Список $(m_1, m_2, \ldots, m_n)$ представляется S-выражением

$$(m_1 \cdot (m_2 \cdot (\ldots (m_n \cdot \text{NIL}) \ldots))),$$

где NIL – атомарный символ, используемый для завершения списков.

Так как многие из символических выражений, с которыми мы имеем дело, удобно выражать в виде списков, введем нотацию списка, чтобы сократить некоторые S-выражения. Именно:

1) (m) заменяет $(m \cdot \text{NIL})$;
2) $(m_1, \ldots, m_n)$ заменяет $(m_1 \cdot (\ldots (m_n \cdot \text{NIL}) \ldots))$;
3) $(m_1, \ldots, m_n \cdot x)$ заменяет $(m_1 \cdot (\ldots (m_n \cdot x) \ldots))$.

Подвыражения можно сокращать аналогичным образом. Вот примеры та-

ких сокращений:

((AB, C), D) вместо ((AB · (C · NIL)) · (D · NIL));
((A, B), C, D · E) вместо ((A · (B · NIL)) · (C · (D · E))).

Так как мы рассматриваем выражения с запятыми как сокращения выражений, не включающих запятых, отнесем их все к S-выражениям.

b. *Функции S-выражений и выражения, их представляющие.* Теперь определим класс функций от S-выражений. Выражения, представляющие эти функции, записываются в традиционной функциональной нотации. Однако чтобы ясно отличать функции от S-выражений, мы будем использовать последовательности букв нижнего регистра для имен функций и переменных над множеством S-выражений. Также мы используем квадратные скобки и точку с запятой вместо круглых скобок и запятых, когда нужно обозначить применение функции к ее аргументам. Таким образом, мы пишем

car[x]
car[cons[(A · B); x]]

Любые S-выражения, встречающиеся в этих M-выражениях (метавыражениях), означают сами себя.

c. *Элементарные S-функции и предикаты.* Введем следующие функции и предикаты:

1. atom. atom[x] имеет значение T или F в зависимости от того, является x атомарным символом или нет. Таким образом:

   atom[X] = T
   atom[(X · A)] = F

2. eq. eq[x;y] определено тогда и только тогда, когда и x, и y атомарны. eq[x; y] = T, если x и y представляют собой один и тот же символ, и eq[x; y] = F в противном случае. Таким образом:

   eq[X; X] = T
   eq[X; A] = F
   eq[X; (X · A)] не определено.

3. car. car[x] определено тогда и только тогда, когда x – не атомарный символ. car[$(e_1 \cdot e_2)$] = e_1. Таким образом, car[X] не определено.

   car[(X · A)] = X
   car[((X · A) · Y)] = (X · A)

4. cdr. cdr[x] также определено, когда x – не атомарный символ. cdr[$(e_1 \cdot e_2)$] = e_2. Таким образом, cdr[X] не определено.

   cdr[(X · A)] = A
   cdr[((X · A) · Y)] = Y

5. cons. cons[x; y] определено для любых x и y. cons[e_1; e_2] = $(e_1 \cdot e_2)$. Таким образом:

   cons[X; A] = (X A)

cons[(X · A); Y] = ((X · A)Y)

Как нетрудно увидеть, car, cdr и cons удовлетворяют отношениям:

car[cons[x; y]] = x
cdr[cons[x; y]] = y
cons[car[x]; cdr[x]] = x в предположении, что x – не атомарный символ.

Имена «car» и «cons» будут иметь мнемоническое значение только при обсуждении представления системы на компьютере. Композиции car и cdr дают подвыражения данного выражения, начинающиеся в заданной позиции. Композиции cons образуют выражения заданной структуры из пар. Класс функций, которые можно образовать таким способом, довольно ограничен и не особо интересен.

d. *Рекурсивные S-функции*. Мы получим намного больший класс функций (фактически все вычислимые функции), допустив образование новых функций S-выражений с помощью условных выражений и рекурсивного определения.

Ниже приведено несколько примеров функций, определяемых таким способом.

ff[x]. Значением ff[x] является первый атомарный символ S-выражения x, не считая скобок. Таким образом:

$$ff[((A \cdot B) \cdot C)] = A$$

Имеем:

$$ff[x] = [atom[x] \rightarrow x; t \rightarrow ff[car[x]]]$$

[Примечание редактора: другие определения функций и предикатов, а также примеры опущены.]

<...>

f. *Универсальная S-функция apply*. S-функция *apply* обладает следующим свойством: если f – S-выражение для S-функции f′, а args – список аргументов вида (arg1, ..., argn), где arg1, ..., argn – произвольные S-выражения, то apply[f; args] и f′[arg1; ... ; argn] определены для одних и тех же значений arg1, ..., argn и равны, когда определены. Например:

λ[[x; y]; cons[car[x]; y]][(A, B); (C, D)]
= apply[(LAMBDA, (X, Y), (CONS, (CAR, X), Y)); ((A, B), (C, D))]
= (A, C, D)

<...>

g. *Функции с функциями в качестве аргументов*. Имеется ряд полезных функций, некоторые из аргументов которых являются функциями. Особенно они полезны при определении других функций. Одна из таких функций, maplist[x; f], имеет два аргумента – S-выражение x и аргумент f, являющийся функцией, которая преобразует S-выражения в S-выражения. Определим

$$maplist[x; f] = [null[x] \rightarrow NIL; T \rightarrow cons[f[x]; maplist[cdr[x]; f]]]$$

Полезность maplist иллюстрируется формулами частной производной по x выражений, включающих суммы и произведения x с другими переменными. S-выражения, которые мы будем дифференцировать, образуются следующим образом.

1. Атомарный символ является допустимым выражением.
2. Если $e_1, e_2, \ldots, e_n$ – допустимые выражения, то (PLUS, $e_1, e_2, \ldots, e_n$) и (TIMES, e_1, e_2, ..., e_n) также допустимы и представляют соответственно сумму и произведение e_1, e_2, ..., e_n.

Это, по сути, польская нотация для функций с тем отличием, что включение скобок и запятых позволяет использовать функции переменного числа аргументов. Примером допустимой функции является (TIMES, X, (PLUS, X, A), Y), что в традиционной алгебраической нотации записывается как X(X + A)Y.

Наша формула дифференцирования, дающая производную y по x, имеет вид:

```
diff[y; x] = [atom[y] → [eq[y; x] → ONE; T → ZERO];
    eq[car[Y]; PLUS] → cons[PLUS; maplist[cdr[y]; λ[[z]; diff[car[z]; x]]]];
    eq[car[y]; TIMES] → cons[PLUS; maplist[cdr[y]; λ[[z]; cons[TIMES;
      maplist[cdr[y]; λ[[w]; ~eq[z;w] → car[w]; T → diff[car[[w]; x]]]]]]]
```

Производная допустимого выражения, вычисляемая по этой формуле, равна:

```
(PLUS, (TIMES, ONE, (PLUS, X, A), Y), (TIMES, X, (PLUS, ONE, ZERO), Y),
    (TIMES, X, (PLUS, X, A), ZERO))

<...>
```

21.4. Система программирования LISP

Система программирования LISP позволяет использовать компьютер IBM 704 для вычисления символической информации в форме S-выражений. Она используется или будет использоваться для следующих целей:

1) написания компилятора для компиляции Lisp-программ на машинный язык;
2) написания программы для проверки доказательств в классе формальных логических систем;
3) написания программ для формального дифференцирования и интегрирования;
4) написания программ для реализации различных алгоритмов порождения доказательств в исчислении предикатов;
5) проведения некоторых инженерных расчетов, результаты которых выражаются формулами, а не числами;
6) программирования системы Advice Taker.

Основа системы – технология написания компьютерных программ для исполнения S-функций. Она будет описана в последующих разделах.

Помимо средства описания S-функций, существуют средства, позволяющие использовать S-функции в программах, записанных в виде последовательностей команд, как в языках Fortran (IBM 1956) или Algol (Perlis and Samelson 1958). В данной статье эти средства не рассматриваются.

a. *Представление S-выражений списковой структурой.* Списковая структура – это набор машинных слов, упорядоченный, как показано на рис. 21.1a или 21.1b. Каждое слово списковой структуры представлено одним из разделенных пополам прямоугольников на рисунке. *Левая* ячейка прямоугольника представляет *адресное* поле слова, а *правая* ячейка – *декрементное* поле. Стрелка от ячейки к другому прямоугольнику означает, что поле, соответствующее ячейке, содержит адрес слова, соответствующего другому прямоугольнику. [Примечание редактора: компьютер IBM 704 предоставлял удобные средства для манипулирования «Содержимым адресного поля регистра» (car) и «Содержимым декрементного поля регистра» (cdr). В каждое слово списковой структуры ранние реализации Lisp помещали два указателя этого слова (на другие слова списковой структуры) в правые биты, чтобы воспользоваться этими средствами.]

Подструктура может встречаться в нескольких местах списковой структуры, как на рис. 21.1b, но структура не может иметь циклы, как на рис. 21.1c.

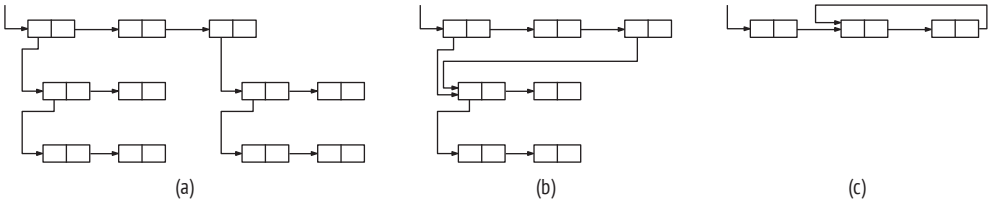


Рис. 21.1

Атомарный символ представлен в компьютере списковой структурой специальной формы, называемой *ассоциативным списком* символа. Адресное поле первого слова содержит специальную константу, которая позволяет программе определить, что это слово представляет атомарный символ. <...>

S-выражение x, не являющееся атомарным, представлено словом, адресная и декрементная части которого содержат адреса подвыражений car[x] и cdr[x] соответственно. <...>

Использование списковых структур для хранения символических выражений имеет следующие преимущества:

1) размер и даже количество выражений, с которыми программа будет вынуждена работать, невозможно предсказать заранее. Поэтому сложно организовать блоки хранения фиксированной длины для их размещения;
2) регистры могут быть возвращены в список свободной памяти, если они больше не нужны. Возвращение даже одного регистра в список полезно, однако если выражения хранятся линейно, то трудно использовать освобождающиеся блоки регистров нечетного размера;
3) выражение, встречающееся как подвыражение в нескольких выражениях, может быть представлено в памяти в одном экземпляре.

<…>

e. *Список свободной памяти*. В любой момент времени только часть памяти, зарезервированной для списковых структур, будет реально использоваться для хранения S-выражений. Остающиеся регистры (в нашей системе их изначально около 15 000) заносятся в простой список, называемый *списком свободной памяти*. Определенный регистр (FREE) в программе содержит адрес первого регистра в этом списке. Когда требуется слово для образования дополнительной списковой структуры, берется первое слово из *списка свободной памяти*, а число в регистре FREE меняется на адрес второго слова в списке свободной памяти. Пользователю нет нужды заботиться о программировании возврата регистров в список свободной памяти.

Этот возврат осуществляется автоматически примерно следующим образом (в данной статье мы сочли нужным дать упрощенное описание процесса). В программе существует фиксированный набор базовых регистров, которые содержат адреса списковых структур, доступных программе. Конечно, так как списковые структуры ветвятся, может быть вовлечено неопределенное количество регистров. Каждый регистр, доступный программе, доступен потому, что его можно достигнуть из одного или более базовых регистров по цепочке операций car и cdr. Если содержимое базового регистра изменяется, то может случиться так, что регистр, на который данный базовый регистр ранее указывал, больше не может быть достигнут по цепочке car–cdr ни из одного базового регистра. Такой регистр можно рассматривать как не используемый программой, потому что его содержимое больше не может быть найдено никаким способом; следовательно, его содержимое больше не представляет интереса, и желательно вернуть его в список свободной памяти. Это происходит следующим образом.

Пока программа не исчерпает свободную память, ничего не происходит. Когда требуется свободный регистр, а в списке свободной памяти ничего не осталось, запускается цикл высвобождения. Вначале программа находит все регистры, доступные из базовых регистров, и делает их знаки отрицательными. Для этого она стартует из каждого базового регистра и меняет знак всех регистров, которые могут быть достигнуты по цепочке car–cdr. Если программе в ходе этого процесса встречается регистр, знак которого уже отрицателен, это означает, что данный регистр уже был достигнут.

После того как знаки у всех доступных регистров были изменены, программа просматривает область памяти, зарезервированную для хранения списковых структур, возвращает все регистры, чьи знаки не были изменены на предыдущем шаге, обратно в список свободной памяти и делает знаки всех доступных регистров вновь положительными.

Этот процесс, так как он полностью автоматический, удобнее для программиста, чем система, в которой он сам должен отслеживать и стирать ненужные списки. Его эффективность зависит от недопущения исчерпания имеющейся памяти доступными списками. Процесс высвобождения занимает несколько секунд, поэтому, чтобы программа не тратила большую часть времени на высвобождение, он должен возвращать в список свободной памяти не меньше нескольких тысяч регистров. <…>

22 Усиление человеческого интеллекта: концептуальная модель (1962)

Дуглас К. Энгельбарт

Дуг Энгельбарт (1925–2013) прочитал работу Буша «Как мы можем мыслить» (глава 11), когда служил на южнотихоокеанском флоте в конце Второй мировой войны. Возможно, это звучит как научная фантастика, но всю свою карьеру он посвятил тому, чтобы облечь в плоть и кровь идеи Буша.

Проект «усиленного человеческого интеллекта» Энгельбарта выполнялся преимущественно в Стэнфордском исследовательском институте (SRI) в Менло-Парке, штат Калифорния. Проект охватывал сотрудничество между людьми и интерактивное общение людей и компьютеров и имел целью усилить интеллектуальные возможности человека. Этот выбор целей – часть долгого обдумывания проекта на ранней стадии. Работа Энгельбарта никогда не считалась основной, даже в институте SRI, славящемся своими инновациями, но попала под финансирование исследовательских работ, выделенное агентством ARPA, одним из отделений которого руководил Дж. К. Р. Ликлайдер. Как часто бывает с рискованными инвестициями в технологии, некоторые части проекта так ни во что и не вылились (например, клавиатура для одной руки, при работе с которой пять пальцев, каждый из которых нажимал на свою кнопку, могли располагаться в 31 позиции, – достаточно для

ввода всех букв латинского алфавита). Другие оказали огромное влияние (например, мышь). А третьи, например идея распределенных коллаборативных технологических процессов, сильно повлияли (или были открыты заново) на более поздние разработки.

Энгельбарт, с одной стороны, абстрагировал, а с другой – конкретизировал провидческую идею Ликлайдера, представленную в знаменитой демонстрации 1968 года, получившую название «мать всех демонстраций». Она состоялась на осенней совместной компьютерной конференции в Сан-Франциско и поразила аудиторию многочисленными интерактивными технологиями, которые теперь прочно вошли в наш быт: мышь, сети, видеконференции, гиперссылки, совместное редактирование текста, окна – и это еще не все.

Спустя десять лет SRI продал лабораторию Энгельбарта коммерческой компании, но там она не достигла процветания отчасти из-за особенностей характера Энгельбарта, а отчасти потому, что бум персональных компьютеров и сетевых технологий, каждый по-своему, изменили способ применения компьютеров. Несмотря на всю свою техническую подкованность, Энгельбарт был в первую очередь предан идее улучшения человека. Он был порождением 1960-х, а с наступлением 1970-х началась коммерциализация. Энгельбарт получил премию Тьюринга в 1997 году «за вдохновляющее предвидение будущего развития интерактивных вычислений и изобретение ключевых технологий, помогающих это предвидение реализовать».

22.1. Введение

Под «усилением человеческого интеллекта» мы понимаем увеличение способности человека находить подход к сложным ситуациям, достигать понимания с учетом своих потребностей и находить решения. В этом отношении под увеличением способностей понимается совокупность следующих элементов: ускоренное и улучшенное осмысление, возможность достигать полезного уровня понимания в ситуации, которая ранее была слишком сложна, принятие лучших решений с большей скоростью, а также умение находить решения задач, которые раньше казались неразрешимыми. А под «сложными ситуациями» мы понимаем профессиональные проблемы дипломатов, руководителей, социологов, биологов, физиков, юристов и дизайнеров – не важно, существует проблема на протяжении 20 минут или 20 лет. Мы не рассматриваем отдельные хитроумные трюки, которые помогают в конкретных ситуациях. Мы говорим о способах поведения в условиях всеобщей взаимосвязанности, когда догадки, метод проб и ошибок, неосязаемые мелочи и человеческое чутье обычно сосуществуют с впечатляющими концепциями, стандартизованными терминологией и нотацией, изощренными методами и электронными устройствами большой мощности.

Население мира и мировой валовой продукт растут быстрыми темпами, однако сложность стоящих перед человечеством проблем растет еще быст-

рее, а находить решения в условиях возрастающего темпа человеческой деятельности и все более глобального ее характера необходимо все быстрее и быстрее. Возможность усиления человеческого интеллекта в том смысле, в каком он определен выше, снискала бы полное одобрение воодушевленного общества, если бы удалось продемонстрировать разумный путь к этой цели и вероятные выгоды.

Данный отчет охватывает первую фазу программы, нацеленной на развитие инструментов приумножения человеческого интеллекта. В эти «инструменты» может входить множество вещей, и все они на деле оказываются лишь дополнениями к инструментам, разработанным и используемым в прошлом, для того чтобы помочь человеку применить свои чувственные, умственные и двигательные способности. Такая система очень важна для нашего общества, и, как и для большинства систем, лучший способ повысить ее действенность – рассматривать целое как совокупность взаимодействующих частей, а не изучать эти части по отдельности.

Такой системный подход к интеллектуальной эффективности человека не дает нам готовой концептуальной основы, подобной той, что имеется в давно оформившихся дисциплинах. До начала планирования исследовательской программы, рационально реализующей подобный подход, так чтобы за приемлемое время можно было бы извлечь практическую пользу, а также получить результаты, которые долго не потеряют актуальность, необходимо отыскать концептуальную основу, которая фокусировалась бы на важных элементах системы, на взаимоотношениях между ними, на типах их изменения, позволяющих улучшить качество ее функционирования, а также на многообещающих целях и методах исследования.

На первом (поисковом) этапе нашей программы мы разработали концептуальную модель, которая представляется удовлетворительной для текущих потребностей планирования этапа исследований. В § 22.2 описана суть этой модели, выведенная в результате применения нескольких различных подходов к анализу системы, состоящей из человека и средств для усиления его интеллекта.

Процесс разработки этой модели позволил сделать ряд важных выводов: что интеллектуальная эффективность, демонстрируемая данным лицом, вряд ли несет признаки ограниченности интеллекта – что существуют десятки дисциплин в технике, математике, физике, науках об обществе и жизни, которые могут внести вклад в улучшение средств усиления интеллекта; что от любого такого улучшения можно ожидать запуска цепочки связанных улучшений; что до тех пор, пока все вышеупомянутые дисциплины не прекратили развиваться и мы не исчерпали всех возможностей что-то почерпнуть из них, можно ожидать, что добавление улучшений в систему усиления человеческого интеллекта продолжится; что нет никаких причин думать, что усиление интеллектуальной эффективности личности, достигаемое в результате слаженного системно-ориентированного подхода, будет уступать расширению пределов географической доступности планеты человеку со времен перемещения на лошадях и на парусниках. <…>

Рассмотрим работу «усиленного» архитектора. Он сидит за рабочей станцией с экраном три на три фута; это его рабочая поверхность, она управля-

ется компьютером (его «помощником»), с которым он может взаимодействовать посредством клавиатуры и других устройств.

Он проектирует здание. Он уже представил в своем воображении основную планировку и формы конструкций, а теперь пытается перенести их на экран. Необходимые ему топографические данные уже введены, и он просит помощника показать перспективный вид строительной площадки на крутом холме с проходящей выше дорогой, с символическими изображениями деревьев, которые должны остаться на площадке, и точками привязки к различным служебным постройкам. Этот вид занимает две трети левой части экрана. С помощью «указателя» он отмечает две интересующие его точки, быстро проводит левой рукой над клавиатурой, и в правой трети экрана появляется высота обеих точек над уровнем моря и расстояние между ними.

Далее он проводит с помощью указателя и клавиатуры линию привязки. Постепенно экран начинает показывать результаты выполняемых им действий – на холме появляется аккуратный котлован, он немного корректируется, и еще немного... В следующий момент архитектор изменяет сцену на экране – просит показать вид на строительную площадку сверху, котлован по-прежнему присутствует. Потратив несколько минут на изучение, он вводит с клавиатуры список элементов, каждый раз проверяя, как очередной элемент выглядит на экране; эти элементы он изучит позже.

Не обращая внимания на изображение на экране, архитектор далее приступает к вводу последовательности спецификаций и данных: перекрытие из шестидюймовых плит, двенадцатидюймовые бетонные стены высотой восемь футов внутри котлована и т. д. Когда он закончит, на экране появится уточненная сцена. Конструкция обретает форму. Он изучает ее, подправляет, время от времени запрашивает у помощника информацию из справочника или каталога и вносит соответственные изменения. Часто он просит «помощника» показать свои рабочие спецификации и соображения, чтобы что-то вспомнить, внести изменения или добавления. Спецификации становятся все более детальной структурой с перекрестными связями, которая представляет наполняющиеся плотью мысли архитектора о плане здания.

Проведя плоскость здесь, искривленную поверхность там, подняв всю конструкцию примерно на пять футов, архитектор наконец получает грубый набросок внешнего вида здания, хорошо вписавшийся в окружающую среду, и может быть уверен, что этот вариант в основном совместим с материалами, которые предполагается использовать, и с назначением постройки.

Теперь он начинает вводить подробную информацию об интерьере. Тут важна способность помощника показать именно тот вид, который он хочет рассмотреть (слой интерьера или как конструкция будет выглядеть с проходящей сверху дороги). Он описывает дизайн отдельных деталей и смотрит, как они выглядят в конкретном помещении. Он хочет убедиться, что солнце, отражающееся от окон, не будет слепить водителя на дороге, а «помощник» производит вычисления, показывающие, что свет, отражающийся от одного из окон, будет сильно мешать водителям между 6:00 и 6:30 утра в середине лета.

Далее архитектор приступает к функциональному анализу. У него имеется перечень людей, которые будут находиться в этом здании, с почасовым описанием их действий. «Помощник» позволяет рассмотреть каждого из

них по очереди, изучить, как открываются и закрываются двери, где может понадобиться специальное освещение. Наконец, он поручает «помощнику» объединить все последовательности действий и отыскать, в каких местах здания будет возникать напряженное движение, где возможны заторы и где будет повышенный спрос на коммунальные услуги. Всю эту информацию (проект здания и ассоциированную с ним «мысленную структуру») можно сохранить на ленте и таким образом создать проектную документацию. Загрузив эту ленту в собственного помощника, другой архитектор, строитель или заказчик может маневрировать, не выходя за его рамки, чтобы уточнить представляющие для него интерес детали или идеи, а также добавлять замечания, которые будут включены в проектную документацию и могут быть использованы им или еще кем-то впоследствии.

В таком будущем взаимодействии человека, решающего задачу, и компьютера-помощника способность компьютера производить математические вычисления может быть использована всюду, где необходимо. Однако компьютер способен на большее, он может обрабатывать и отображать информацию, что будет весьма полезно человеку при выполнении таких нематематических действий, как планирование, организация, изучение и т. д. Любой, кто мыслит в терминах концепций, представленных символами (в форме ли английского языка, с помощью ли пиктограмм, формальной логики или математики), сможет использовать эти возможности с огромной пользой.

22.2. Концептуальная модель

22.2.1. Общие соображения. Искомая концептуальная модель должна показывать нам реальные возможности и проблемы, связанные с использованием современных технологий для оказания прямой помощи человеку, которому необходимо разобраться в сложной ситуации, выделять существенные факторы и разрешать проблемы. Чтобы приблизиться к этой цели, мы изучим, как индивидуумы достигают имеющегося на сегодняшний день уровня эффективности, и ожидаем, что это изучение откроет возможности для совершенствования. <...>

Любой процесс, не важно, связан он с мышлением или действием, состоит из подпроцессов. Рассмотрим в качестве примеров проведение черточки карандашом, написание буквы алфавита или составление плана. Проведение черточки нуждается в организации нескольких дискретных сокращений мускулов. Аналогично нанесение нескольких черточек и умозрительное представление буквы – сами по себе сложные процессы, являющиеся подпроцессами полного процесса написания буквы алфавита.

Хотя каждый подпроцесс и сам является полноправным процессом в том смысле, что состоит из более мелких подпроцессов, представляется бессмысленным опускаться до самого нижнего уровня этой иерархической структуры процессов. Похоже, невозможно сказать, существуют ли такие последние уровни (процессы, не допускающие дальнейшего разбиения) в физическом мире или это всего лишь ограничения способности человека к пониманию.

Да и в любом случае нет необходимости начинать с самого низа при обсуждении конкретной иерархии процессов. Никто не прибегает к совершенно уникальному процессу всякий раз, как сталкивается с чем-то новым. Вместо этого индивидуум начинает с группы базовых чувственных-умственных-двигательных возможностей и добавляет к ним некоторые из возможностей своих артефактов. Имеется лишь конечное множество таких базовых возможностей человека и артефактов, и только из них можно выбирать. Кроме того, даже сильно различающиеся процессы высокого порядка могут иметь общие подпроцессы сравнительно высокого порядка.

Когда человек пишет прозаический текст (процесс достаточно высокого порядка), он использует в качестве подпроцессов много процессов, являющихся частями других процессов высокого порядка, например планирование, композиция, диктовка. Но и сам процесс написания является подпроцессом многих других процессов еще более высокого порядка, например организации комитета, изменения политики и т. д.

Таким образом, каждый индивидуум создает некоторый арсенал возможностей, из которых выбирает то, что ему нужно, и адаптирует их для составления процессов, которые исполняет. Этот арсенал можно уподобить ящику с инструментами, и точно так же, как механик должен знать, что могут делать его инструменты и как ими пользоваться, так и работник умственного труда должен знать возможности своих инструментов и иметь хорошие методы, стратегии и эвристические правила их использования. Все возможности в арсенале индивидуума в конечном итоге сводятся к базовым возможностям его самого или его артефактов, а арсенал в целом представляет собой переплетенную иерархическую структуру (которую мы часто называем арсенальной иерархией).

В арсенале типичного индивидуума мы находим три общие категории возможностей. Во-первых, это возможности, реализуемые полностью внутри телесной оболочки человека, будем называть их чисто человеческими; во-вторых, это принадлежащие артефактам возможности для выполнения процессов без вмешательства человека, будем называть их чисто артефактными; и, в-третьих, это смешанные возможности, которые происходят из иерархий, содержащих возможности двух других видов.

Мы предполагаем, что наша система H-LAM/T (Human using Language, Artifacts, Methodology, in which he is Trained – система подъема способностей человека посредством языка, артефактов и методологии) имеет возможность и выполняет процесс при любом способе использования этого арсенала. Заглянем внутрь структуры процессов для составных частей LAM/T, чтобы лучше «прочувствовать» наши модели. Возьмем процесс написания важного меморандума. С этим процессом ассоциирована важная концепция – оформление информации в виде формального пакета и распространение ее среди определенных людей для рассмотрения; тип информационного пакета, связанный с этой концепцией, получил специальное название – меморандум. Уже сам язык системы несет на себе отпечаток этого процесса, именно концепция и ее название. <…>

22.2.2. Базовая перспектива. Индивидуумы, которые эффективно функционируют в нашей культурной среде, уже значительно «усилены». Базовые

человеческие способности к чувственному восприятию стимулов, к выполнению многочисленных умственных операций и к общению с внешним миром привлекаются к работе в нашем обществе в рамках некоторой системы – системы H-LAM/T – системы подъема способностей человека посредством языка, артефактов и методологии. Кроме того, мы подозреваем, что к повышению эффективности индивидуума в контексте его деятельности в нашем обществе следует подходить как к задаче системной инженерии – т. е. систему H-LAM/следует изучать как взаимодействующее целое с точки зрения подхода, ориентированного на синтез.

Этот взгляд на систему как на взаимодействующее целое сильно подкреплен рассмотрением арсенальной иерархии возможностей, которая построена из базовых составных частей в системе HLAM/T. Осознание того, что любое потенциальное изменение языка, артефакта или методологии важно только относительно его использования внутри процесса и что новая возможность, появившаяся в любом месте иерархии, может сделать практически полезным новое рассмотрение потенциально скрытых изменений во многих других частях иерархии – в языке, артефактах или методологии, – делает очевидной сильную взаимосвязь трех этих средств усиления.

Повышение эффективности использования индивидуумом его базовых возможностей создает проблему при перепроектировании изменяемых частей системы. Система активно участвует в непрерывных процессах (среди прочих) развития способности к постижению у индивидуума и решения проблем; оба процесса обусловлены мотивацией, целеполаганием и волей человека. Перепроектировать способность системы к выполнению этих процессов значит перепроектировать арсенальную иерархию полностью или частично. Чтобы перепроектировать конструкцию, мы должны узнать как можно больше из того, что известно об основных материалах и компонентах, применяемых в этой конструкции; помимо этого, мы должны научиться рассматривать, измерять, анализировать и оценивать ее в терминах функциональной целостности и назначения. В данном случае никакая из имеющихся аналитических теорий сама по себе недостаточна для анализа и оценки функционирования системы в целом; таким образом, для получения улучшенной системы необходимо использовать экспериментальные методы.

При таком перепроектировании необязательно добавлять или модифицировать очень сложные или формальные процессы. Практически все процессы, осуществляемые сегодня типичным человеком, – процессы, о которых он думает, когда мысленно представляет себе работу предстоящего дня, – это составные процессы того или иного вида, включающие внешнюю компоновку и манипулирование символами (текст, наброски, диаграммы, списки и т. д.). Многие процессы внешней компоновки и манипулирования (модификация, реорганизация) служат таким характерным для человека видам деятельности, как рассмотрение форм и связей с разных сторон с целью понять, что получается, многоходовое развитие идеи методом проб и ошибок или составление списка предметов для размышления с последующей реорганизацией и расширением по мере появления новых мыслей.

Технологии, которые существуют уже сейчас или появятся в ближайшем будущем, конечно, дают в руки нашим профессиональным «решателям проб-

лем» артефакты, необходимые им для дублирования и реорганизации текста перед глазами быстро и с минимумом усилий. Но даже в этом случае кажущееся незначительным улучшение могло бы привести к тотальным изменениям в арсенальной иерархии индивидуума, которые вылились бы в существенное повышение его общей эффективности. Обычно необходимое оборудование приходит на рынок медленно; неспешно и постепенно накопившиеся изменения создают рынок для радикально новых версий оборудования. Такой эволюционный процесс был типичен и для формирования роста наших арсенальных иерархий.

Но активные исследования, нацеленные на изучение и оценку возможных комплексных изменений в арсенальной иерархии, могли бы существенно ускорить эволюционный процесс. Исследования могли бы направить разработку новых артефактов в сторону действий, осмысленных в долгосрочной перспективе; в то же время индивидуумы, настроенные на работу в конкурентных условиях, хорошо реагирующие на продемонстрированные методы увеличения личной эффективности, создавали бы рынок для более радикальных инноваций в области оборудования. Можно ожидать, что такой направляемый эволюционный процесс окажется значительно быстрее традиционного.

В категорию «более радикальных инноваций» попадает цифровой компьютер как инструмент для персонального использования индивидуумом. Тут открываются не только перспективы большей гибкости при составлении и реорганизации текста и диаграмм прямо перед глазами индивидуума, но и многие другие возможности, которые можно интегрировать в арсенальную иерархию системы H-LAM/T.

22.2.3. Подробное обсуждение системы H-LAM/T

22.2.3.1. Источник интеллекта. Глядя на компьютерную систему, работающую над каким-то очень трудным заданием, человек на поверхности видит машину, которая способна исполнять чрезвычайно сложные процессы. Размышляя о том, что же является источником таких фантастических возможностей, неспециалист может наделить машину мистической способностью собирать и перерабатывать информацию с помощью воспринимающих и наделенных интеллектом искусственных думающих устройств. На самом же деле эти развитые возможности – результат очень умно организованной иерархии, поэтому в поисках источника интеллекта в системе мы опускались бы все ниже и ниже по уровням функциональной и физической организации, становящимся чем ниже, тем примитивнее.

Конкретно, мы можем начать сверху и перечислить основные уровни, которые увидели бы, последовательно разлагая на части функциональные элементы на каждом уровне в поисках «источника интеллекта». Программист сопровождал бы нас на нескольких уровнях – от одного до трех в зависимости от сложности процесса, выполняемого компьютером, – и, наверное, изображал бы организацию посредством структурной схемы. На первом уровне могли бы находиться функции, соответствующие предложениям проблемно-ориентированного языка (например, Algol или Cobol), благодаря которым достигаются конечные цели процесса в целом. На втором уровне меньшее

число функций было бы организовано в процессы, представленные предложениями на первом уровне. На третьем уровне, возможно, было бы видно, как элементарные машинные команды (точнее, процессы, которые ими представлены) организованы для реализации функций на втором уровне.

Дальше нас встретил бы конструктор машины и на принципиальной схеме организации компьютера показал бы (уровень 4), как различные аппаратные блоки (например, запоминающее устройство с произвольным доступом, арифметические регистры, сумматор, блок управления) организованы в систему, позволяющую выполнять последовательности команд на уровне 3. Конструктор логических схем мог бы затем провести для нас экскурсию по уровню 5, также с помощью принципиальных схем, и показать, каким образом такие элементы, как импульсные вентили, триггеры и схемы И, ИЛИ и НЕ, можно организовать в структуры, обеспечивающие работу функций уровня 4. На уровне 6 инженер-схемотехник показал бы нам диаграммы организации таких компонентов, как транзисторы, сопротивления, конденсаторы и диоды, в модульные структуры, необходимые для функционирования элементов уровня 5.

Физики разных направлений и конструкторы отдельных устройств могли бы провести нас по еще более глубоким уровням. Но довольно скоро мы пересекли бы границу между тем, что организовано человеком и природой, и в конечном итоге стали бы говорить о том, как некое физическое явление вытекает из организации субатомных частиц, а наша способность объяснить последующие уровни натолкнулась бы на недостаток нынешнего понимания этих материй.

Если теперь задаться вопросом, где же находится интеллект, то мы будем вынуждены признать, что он неуловимо распределен по иерархии функциональных процессов – иерархии, уходящей своим основанием в естественные процессы за пределами нашего понимания. Если и существует какая-то одна вещь, от которой зависит этот интеллект, то это, пожалуй, организация. Биологи и физиологи употребляют термин «синергизм» (Webster 1959), означающий «...усиливающий эффект взаимодействия двух или более факторов, характеризующийся тем, что их совместное действие превосходит сумму действий каждого их них...». Этот термин кажется непосредственно применимым к данной ситуации, так что можно было бы сказать, что синергизм – наиболее вероятный кандидат на роль фактического источника интеллекта.

На самом деле любое наблюдаемое нами явление – физическое, социальное или бытовое – по-видимому, можно вывести из поддерживающей его иерархии организованных функций (или процессов), в которой принцип синергизма придает более высокую феноменологическую сложность каждому последующему уровню иерархии. В частности, интеллект человека, вытекающий в конечном итоге из характеристик отдельных нервных клеток, несомненно является результатом синергизма.

22.2.3.2. Усиление интеллекта. В ходе этого исследования не раз предлагалось в шутку назвать то, что мы ищем, «усилителем интеллекта» (intelligence amplifier). (Этот термин приписывается У. Россу Эшби (Ashby 1952, 1956).) Поначалу он был отвергнут на том основании, что, на наш взгляд, единственная надежда – установить более точное соответствие между уже

имеющимся интеллектом человека и решаемыми задачами, а не сделать человека умнее. Но развитие идей, предложенных вниманию читателя в предыдущем разделе, показало нам, что этот термин все же применим к достижению поставленной нами цели.

Принятие термина «усиление интеллекта» не означает попытки увеличить естественный интеллект человека. Нам представляется, что он применим к нашей цели дополнить интеллект человека, в том смысле, что сущность, которая должна появиться в результате, будет проявлять большее количество того, что можно было бы назвать интеллектом, чем человек, лишенный дополнительной поддержки; мы усилим интеллект человека, организовав его интеллектуальные способности на верхних уровнях синергетической структуры. Обладать усиленным интеллектом будет результирующая система H-LAM/T, в которой дополнение человека LAM/T будет означать усиление его интеллекта.

Стремясь усилить наш интеллект, мы применяем принцип синергетической организации, который действовал в ходе естественной эволюции базовых человеческих способностей. При разработке наших средств дополнения мы сконструировали суперструктуру, которая является синтетическим расширением естественной структуры, на базе которой она построена. В некотором вполне вещественном смысле развитие «искусственного интеллекта» продолжается уже несколько столетий, о чем свидетельствует поступательная эволюция средств дополнения наших способностей.

22.2.3.3. Двухдоменные системы. Человек и артефакты – единственные физические компоненты системы H-LAM/T. Только их возможностями будут в конечном итоге определяться возможности системы. Этот вывод следует из ранее высказанного утверждения о том, что любой составной процесс системы в конечном итоге разлагается на чисто человеческие и чисто артефактные процессы. Таким образом, есть два отдельных домена деятельности внутри системы H-LAM/T: представленный человеком – там протекают все чисто человеческие процессы – и представленный артефактами – там протекают все чисто артефактные процессы. В любом составном процессе эти два домена взаимодействуют, что требует обмена энергией (в основном для целей информационного обмена). На рис. 22.1 изображена эта концепция двух доменов наряду с другими концепциями, обсуждаемыми ниже.

В ситуации, когда ведущий артефакт, с которым взаимодействует человек, представлен сложной машиной, на протяжении нескольких лет использовался термин «человеко-машинный интерфейс»; под этим понималась граница, через которую происходит обмен энергией между обоими доменами. Однако уже много веков – с тех пор, как люди стали использовать артефакты и выполнять составные процессы, – существует «интерфейс человек–артефакт».

Обмен через этот «интерфейс» имеет место, когда чисто человеческий процесс сопрягается с чисто артефактным. Очень часто эти сопряженные процессы проектируются исключительно для такого обмена, чтобы обеспечить функциональное соответствие между другими чисто человеческими и чисто артефактными процессами, скрытыми внутри своих доменов и выполняющими более важные вещи. Например, движения пальцами и запястьем (чисто

человеческие процессы) активируют движения клавиш пишущей машинки (сопряжение с чисто артефактными процессами). Но это только часть соответствия между более глубокими человеческими процессами, имеющими целью напечатать некоторое слово, и более глубокими артефактными процессами, которые приводят к появлению чернильного отпечатка на бумаге. <...>

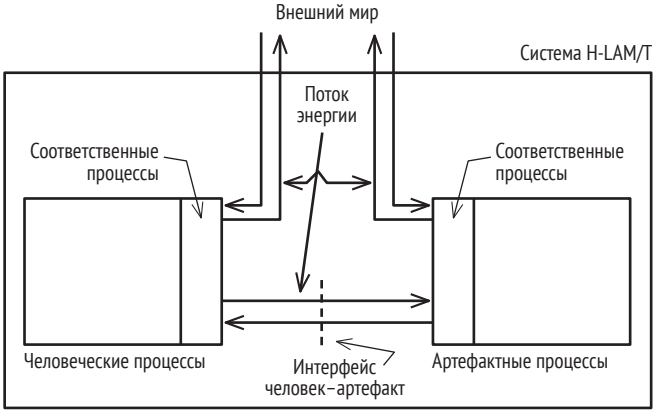


Рис. 22.1. Две стороны системы H-LAM/T

22.3. Примеры и обсуждение

22.3.2. Гипотетическое описание системы усиления на основе компьютера. <...> Рассмотрим некоторые конкретные возможности перепроектирования средств усиления с ориентацией на интеллектуальную поддержку человека, решающего проблемы. Мы решили представить те разработки языка и методологии, которые могут с пользой задействовать средства компьютеризованного оборудования, предназначенного для обработки строк и рисования изображений. Спектр таких возможностей быстро изменяется и расширяется по мере проведения научно-исследовательских работ, но мы хотим сделать то, что можем, – придать конкретики общим положениям, изложенным в § 22.2, попытаться донести до читателя наше ощущение богатства ожидающих нас возможностей и познакомить его с возможными направлениями исследований. <...>

22.3.2.1. Обстановка. Чтобы попытаться разделить с читателем наше отношение к заявленному тезису, несмотря на текущую ситуацию, мы представим картину усиления возможностей с помощью компьютера, описав, как могло бы выглядеть обсуждение, сопровождаемое демонстрацией, которую специально для вас организовал парень (по имени Джо), являющийся обученным и опытным пользователем такой системы усиления, созданной в результате экспериментальной исследовательской программы, до которой от настоящего момента еще несколько лет. Мы предполагаем, что вы по-

дошли к такой беседе с демонстрацией, обладая багажом знаний типа того, что можно почерпнуть из предшествующей части данного отчета, – т. е. что вы слышали или читали об общих положениях и знакомы с несколькими довольно примитивными примерами, но по-настоящему не прочувствовали, чем компьютеризованная система усиления может помочь человеку.

Джо понимает это и говорит, что сделает все, что в его силах, чтобы познакомить вас с концептуальной моделью, и для этого попытается пройти по узкой дорожке между излишней детализацией, чреватой утратой общего видения, и излишним обобщением, опасным тем, что вы не получите наглядного представления о том, что происходит. Он предлагает вам устроиться поудобнее и наблюдать за тем, как он будет выполнять какую-то типичную работу, после чего даст необходимые пояснения. Вам это не особенно по душе, потому что вы знаете, что он всего лишь собирается применить новые наработки в области языка и методологии к своим новым артефактам, а артефакты эти не особенно отличаются от того, что вы ожидали, – так зачем вам тут сидеть, притворяясь, что вы ничего такого в жизни не видали? Вам всего-то планируют показать, как «компьютер выполняет за него какие-то процессы манипулирования символами, чтобы он мог использовать более мощные концепты и приемы манипулирования концептами» – о чем вам уже много раз рассказывали.

Перед Джо бок о бок стоят два экрана, но один из них он, похоже, использует не так интенсивно, как другой. И экраны эти почти горизонтальны, они больше похожи на чертежную доску, чем на почти вертикальные дисплеи, которые вы себе почему-то вообразили. Но вы быстро понимаете, почему так сделано, – ведь он работает на поверхности дисплея так же сосредоточенно, как чертежник, склонившийся над своими чертежами, и было бы странно делать такую работу на вертикальной поверхности. Иногда Джо обеими руками стучит по клавиатуре – видимо, с большой скоростью вводит в компьютер данные. И вот что еще немного удивляет: каждая рука работает с клавишами, расположенными по разные стороны от окон на экране, так что руки разведены почти на два фута. Но понятно, что такая организация позволяет ему естественно находиться прямо над окнами, поэтому, когда он извлекает из воздуха световое перо (которое находится в нейтральном положении благодаря системе сочлененных кронштейнов и системе управления натяжением и сматыванием шнура), его рука по-прежнему находится на пути от клавиатуры к окну на экране. Закончив работать световым пером, он отпускает его, шнур сматывается, и перо возвращается в нейтральную позицию. Таким образом, чтобы вернуться к работе над окном, нужно затратить минимум усилий, движений и времени. То есть он без труда может переключаться с клавиатуры на световое перо и обратно любой рукой (для каждой руки предусмотрено свое световое перо), не поворачивая ни головы, ни корпуса и не принимая неудобных поз.

Похоже, Джо чаще всего держит одну руку на клавиатуре, а другой водит световым пером по поверхности дисплея. Именно в таком режиме изображения в окнах изменяются наиболее динамично. Еще один шок вы испытываете, осознав, как много всего происходит на экранах дисплеев. Вы задаетесь вопросом, почему не были к этому подготовлены, и вынуждены признать,

что общие положения, о которых вы слыхали, не имеют ничего общего с реальным погружением – фраза «новые методы манипулирования символами» повторялась часто, но перед вашими глазами не возникала картина быстрых и беспрепятственных действий Джо, изменяющего изображение на экране, и вы даже не думали, что имеющее практический смысл и гибкое воплощение идей случится так скоро.

Затем вы осознаете, что вообще не понимаете ни того, что он делает, ни большей части увиденного на экранах. Многие слова понятны, но еще больше таких, что, по-видимому, являются какими-то аббревиатурами. В те редкие моменты времени, когда изображение или его часть остаются неизменными достаточно долго, чтобы его можно было хоть немного изучить, вы почти никогда не видите ничего похожего на привычные предложения. До вас постепенно доходит, что в предложении слова перемешаны с какими-то другими символами и что различные части того, что должно было бы составлять законченное высказывание (именно так вы представляете себе предложение), просто расположены не последовательно с начала до конца, как вы ожидали. Но тут Джо внезапно очищает экран и поворачивается к вам с усмешкой, означающей, что период пассивного созерцания подошел к концу. И вы понимаете, что он прекрасно знает, что вам было необходимо это время, чтобы избавиться от своих ограниченных представлений и в полной мере осознать, что «иерархия возможностей» – действительно весьма насыщенная и полная жизни штука.

«Полагаю, вы заметили, что я использовал незнакомые обозначения, символы и процессы, чтобы проделать вещи, которые вам еще менее знакомы?» Вы уклончиво киваете – нет никаких причин признаваться, что вы даже не поняли, что с чем работало совместно, – а он продолжает: «Чтобы вы почувствовали, что происходит, я начну с обсуждения и демонстрации очень простых операций и обозначений, которыми пользовался. Уверен, вы читали о процессах и иерархиях возможностей. По прошлому опыту объяснения систем радикального усиления я знаю, что новые мощные возможности высокого уровня, интересные людям, – потому что это то, что все мы были бы рады улучшить, – не получается по-настоящему объяснить, не рассказав предварительно о новых и мощных возможностях, на которых они основаны. Речь идет о возможностях более низкого уровня, которые тоже являются новыми и непривычными, но которые мало кто назвал бы “мощными”. И тем не менее без них наши системы не были бы даже близко такими мощными, а представление человека о системе было бы довольно поверхностным без понимания этих базовых возможностей и иерархической структуры, которая поднимается от них к возможностям самого верхнего уровня». <…>

22.6. Выводы

Мы можем сделать три важных вывода, касающихся значимости и следствий из представленных выше идей.

Во-первых, любая возможность повысить эффективность использования интеллектуальных способностей людей, отвечающих за решение обществен-

ных проблем, заслуживает самого серьезного рассмотрения. Поскольку способность человека к решению проблем – вероятно, самый важный ресурс общества. Развитие и использование всех других претендентов на первое место критически зависит от этого ресурса. Любая возможность развития науки или искусства, которые могли бы прямым и значимым образом повлиять на продолжающееся совершенствование этого ресурса, вдвойне заслуживают серьезного отношения. Во-вторых, изложенные идеи следует рассматривать в обоих вышеозначенных смыслах; прямое совершенствование и «искусство совершенствования». Точнее, у возможностей есть долгосрочные последствия, но стремление к их реализации и первые плоды ждут нас уже сейчас. На наш взгляд, нет никакой нужды ждать, когда нам станет понятно, как устроены ментальные процессы у человека, необязательно ждать, когда мы научимся делать компьютеры умнее, больше или быстрее. Мы можем приступить к разработке мощных и экономически оправданных систем усиления на основе того, что знаем и имеем уже сейчас. Погоня за знаниями и желание совершенствовать машины не прекратятся в обозримом будущем, и мы всегда будем стремиться интегрировать результаты в «искусство» и улучшенные системы усиления – но, начав сейчас, мы получим не только направление таких исследований и стимул к ним, но и улучшим эффективность решения проблем и тем самым поможем эти исследования выполнять.

В-третьих, становится все более очевидно, что уже сейчас – чем раньше, тем лучше – необходимы действия в самых разных исследовательских сообществах, и действия агрессивные. Мы предлагаем концептуальную модель и план действий и рекомендуем внимательно рассмотреть их как основу для действия. Если после рассмотрения они будут признаны неприемлемыми, то, по крайней мере, следует предпринять серьезные и непрекращающиеся усилия, чтобы выработать более приемлемую концептуальную модель, в рамках которой рассмотреть всеобъемлющий подход, или чтобы выработать более приемлемый план действий, или то и другое.

Это открытый призыв к исследователям и всем тем, кто в конечном итоге мотивирует, финансирует или направляет их, обратить самое серьезное внимание на возможность развития динамической дисциплины, в контексте которой можно было бы рассматривать проблему повышения интеллектуальной эффективности в полном объеме. Эта дисциплина должна быть нацелена на организацию непрерывного цикла улучшений – улучшенного понимания проблемы, улучшенных средств разработки новых систем усиления и улучшенных систем усиления, которые могли бы обслуживать людей, занятых решением мировых проблем в целом и работающих над этой дисциплиной в частности. В конце концов, тратим же мы гигантские суммы на развитие дисциплин, нацеленных на понимание и обуздание ядерной энергии. Так почему же не подумать о разработке дисциплины, нацеленной на понимание и обуздание «нервной энергии»? В перспективе из двух этих целей могущество человеческого интеллекта – безусловно, важнейшая.

23 Экспериментальная система с разделением времени (1962)

Фернандо Корбато, Марджори Мервин Даггетт, Роберт К. Дейли

К середине 1950-х годов исследователи накопили достаточно опыта построения и использования электронных компьютеров, чтобы начать задумываться о различных путях эволюции компьютеров, которые позволили бы использовать их большему количеству людей для решения большего числа задач. В 1954 году на летней школе в МТИ произошел примечательный обмен мнениями между Грейс Хоппер и Джоном Бэкусом, который вскоре после того разработал язык программирования Fortran. «Д-р Грейс Хоппер поднимала вопрос о возможности параллельного использования нескольких малых компьютеров. Наибольшим спросом пользовались малые машины. <…> Она предвидела массовое производство малых машин, снабженных компилятором и библиотекой, отвечающей потребностям заказчиков. М-р Дж. У. Бэкус не соглашался с этой философией, исходя из соображений быстродействия компьютера; поскольку затраты на повышение скорости невелики, большой компьютер использовать дешевле, чем малый. <…> Джон Бэкус говорил, что благодаря разделению времени большой компьютер можно было бы использовать как несколько малых; нужно будет только предоставить каждому пользователю отдельную станцию для чтения» (Adams et al. 1954, стр. 16-1–16-2).

Хоппер и Бэкус согласились остаться при своих мнениях на том основании, что Хоппер в большей степени думала о коммерческих приложениях, а Бэкус – о научных. Но идея разделения времени уже родилась. Спустя несколько

лет группа в МТИ, возглавляемая Фернандо Корбато (1926–2019), полностью воплотила ее в жизнь.

Корбато защитил докторскую диссертацию по физике в МТИ в 1956 году и продолжил заниматься эксплуатацией вычислительного центра. Примерно в 1958 году Джон Маккарти, тогда преподававший в МТИ, предложил расширить возможности установленного в центре компьютера IBM, реализовав систему с разделением времени, о чем упоминал Ликлайдер (см. 271 этой книги). В 1961 году, уладив разногласия внутри МТИ и заручившись согласием IBM сотрудничать, Корбато, которому помогали программисты Марджори Мервин (родилась в 1928 году, впоследствии взявла фамилию мужа Даггетт) и Роберт К. Дейли, «сваяли» рудиментарную систему. Как гласит изустная история, целью было «убедить скептиков том, что задача не является невозможной, а заодно дать людям ощутить вкус интерактивной работы с компьютером. Для меня было удивительно тогда и удивительно до сих пор, что люди не могли представить себе, какое психологическое воздействие окажет появление интерактивного терминала. Вы можете до посинения писать на доске, а слушатели будут только приговаривать «Да, да, конечно, но зачем это нужно?». Мы, знаете ли, испробовали всякие аналогии, например: отправить матушке письмо и поговорить с ней по телефону. До сих пор помню, что осознание пришло к людям, только когда они увидели реальную демонстрацию. Вот тут-то они начали говорить: “Слушай, да она же отвечает. Вот это да! Ты печатаешь и получаешь ответ”» (Corbató and Norberg 1989).

Эта первая система получила название Compatible Time-Sharing System[1] (CTSS); имелась в виду «совместимость» между интерактивными и счетными заданиями, работавшими в фоновом режиме. Это была основа, впоследствии замененная системой Multics (MULTiplexed Information and Computing Service). МТИ продолжал разрабатывать эту систему десять лет, она была коммерциализирована, с ограниченным успехом, компанией GE, а затем Honeywell. Bell Labs тоже участвовала в разработке, но вышла из проекта, сочтя, что он становится слишком громоздким. Входивший в состав разработчиков Кен Томпсон положил уроки, вынесенные за время работы над Multics, в основание операционной системы, которую, беззлобно скаламбурив[2], назвал Unix (глава 37).

Из интересных аспектов этой статьи следует отметить математический анализ производительности и ретроспективную оценку ее влияния на ранние разработки компьютерных систем.

23.0. Краткое описание

Цель настоящей статьи – кратко обсудить потребность в разделении времени, некоторые проблемы реализации, экспериментальную систему с разделением времени, разработанную на современной вычислительной машине

---

[1] Совместимая система с разделением времени. – *Прим. перев.*

[2] MULTICS – много, UNIX – что-то одно. – *Прим. перев.*

IBM 7090, и, наконец, алгоритм планирования, предложенный одним из нас (Корбато), чтобы проиллюстрировать некоторые методы повышения и анализа пределов производительности такой системы.

23.1. Введение

За последние лет десять использования компьютеров были достигнуты большие успехи. В начале 1950-х годов задачи решались в основном путем конструирования и обслуживания оборудования; в середине 1950-х годов пришествие компиляторов существенно улучшило использование языков; а теперь, в начале 1960-х годов, мы являемся свидетелями третьего крупного изменения в области использования компьютеров: усовершенствования человеко-машинного взаимодействия посредством процесса разделения времени.

Значительная часть философии разделения времени, изложенной в этой статье, была разработана совместно с комитетом МТИ по предварительным исследованиям под председательством Х. Тигера, который изучал перспективные вычислительные потребности института, а затем с рабочим комитетом МТИ по компьютерам, возглавляемым Дж. Маккарти. Однако взгляды и выводы, представленные здесь, следует рассматривать исключительно как мнения авторов настоящей статьи.

Прежде чем двигаться дальше, следует дать более точное толкование термину «разделение времени». Он может означать как одновременное использование разных частей оборудования для решения разных задач, так и одновременное использование компьютера несколькими пользователями. Первое значение, часто называемое мультипрограммированием, направлено на повышение эффективности использования оборудования в смысле попытки добиться полной загрузки всех компонентов (Schmitt and Tonik 1959; Codd 1960; Heller 1961; Leeds and Weinberg 1961). Второе значение – то, что подразумевается здесь, – в основном связано с повышением эффективности работы людей, пытающихся использовать компьютер (Strachey 1959; Licklider 1960; Brown et al. 1962). Эффективность использования компьютера тоже следует принимать во внимание, но только с точки зрения общей полезности системы.

Побудительным мотивом к использованию компьютера в режиме разделения времени является медленность возможного в настоящее время взаимодействия человека с большими передовыми компьютерами. Его скорость изменилась не сильно (и зачастую в худшую сторону) за последние десять лет широкого распространения компьютеров (Teager and McCarthy 1959).

Отчасти причиной стало то, что как только решение элементарных задач на компьютере было освоено, сразу же возник интерес к более сложным задачам. И в результате, чтобы воспользоваться преимуществами более крупных и быстрых компьютеров, стали писать более крупные и сложные программы. Это неизбежно ведет к большему числу ошибок программирования и увеличению времени отладки. При использовании современных пакетных мони-

торов, как это делается на большинстве крупных компьютеров, устранение каждой ошибки занимает несколько часов, а то и целый день. Единственная альтернатива, доступная сегодня программисту, – попробовать отлаживать непосредственно на компьютере, но этот процесс ведет к очень непроизводительному расходованию компьютерного времени и серьезно осложняется недостатками имеющихся на сегодняшний день методов взаимодействия с консолью. Даже если консолью является пишущая машинка, обычно отсутствуют развитые программы запроса и ответа, которые жизненно необходимы для эффективного взаимодействия. Таким образом, желательно резко увеличить скорость взаимодействия программиста с компьютером без больших экономических потерь, а также сделать каждое взаимодействие более осмысленным, применив развитые и комплексные системные программы в помощь коммуникации между человеком и машинной.

Для решения этих проблем взаимодействия мы хотели бы, чтобы компьютер мог одновременно обслуживать несколько пользователей по аналогии с телефонным коммутатором. Каждый пользователь должен иметь возможность работать с консолью в удобном для себя темпе, не думая о том, что делают другие пользователи системы. Как минимум эта консоль может быть простой печатной машинкой, но в идеале хотелось бы иметь дисплей с постепенным обновлением и автоматической регенерацией изображения. В любом случае требования к передаче данных должны быть таковы, чтобы удаление консоли от самого компьютера не приводило ни к каким трудностям.

Базовая организация системы с разделением времени подразумевает, что много людей одновременно работают с компьютером с помощью консолей в виде пишущей машинки, а программа-супервизор последовательно выделяет каждой пользовательской программе короткий промежуток, или квант машинного времени. Эта последовательность, которая в простейшем случае представляет собой циклический перебор, должна повторяться достаточно часто, чтобы каждая пользовательская программа, хранящаяся в высокоскоростной памяти, получала квант, по крайней мере, один раз в течение времени типичной человеческой реакции (~0,2 с). Тогда каждый пользователь будет видеть, что компьютер реагирует даже на одиночные нажатия клавиш, для обработки которых требуется лишь тривиальное вычисление; в нетривиальных случаях пользователь видит постепенное увеличение времени отклика, пропорциональное сложности вычисления ответа, замедлению компьютера и общему числу пользователей. Однако должно быть ясно, что если имеется n пользователей, активно запрашивающих одновременное обслуживание, то каждый пользователь будет получать в среднем только $1/n$ эффективного быстродействия компьютера. В периоды интенсивного взаимодействия во время отладки программ это не должно быть серьезной помехой, потому что обычно объем вычислений, необходимых для реакции на отладочное действие, невелик по сравнению с потребностями в производственном режиме.

Такая система с разделением времени не только повысила бы нашу способность к традиционному программированию на один-два порядка, но и открыла бы несколько новых форм использования компьютеров. Многие научные и инженерные приложения можно было бы постепенно переписать

так, чтобы избавиться от программ, содержащих деревья решений, которые в настоящее время нужно определять заранее, а вместо этого задавать только конкретные ветви деревьев по мере необходимости. Еще одна важная область применения – обучающие машины, которые при всей своей вычислительной тривиальности могли бы естественно задействовать консоли системы с разделением времени, получив при этом дополнительный бонус – возможность использования более сложных и адаптивных обучающих программ. Наконец, как свидетельствует наличие многочисленных компьютеров для малого бизнеса, в бизнесе и в промышленности имеется множество приложений, в которых было бы выгодно иметь мощные вычислительные средства, установленные в изолированных местах, что позволило бы ограничиться постепенными инвестициями в приобретение консолей. Но важно понимать, что даже с учетом всех вышеперечисленных и новых приложений основное преимущество от ускорения программирования, которое несет разделение времени, получат вычислительные центры в университетах, исследовательских лабораториях и инженерных фирмах, где отладка программ является серьезной проблемой.

23.2. Проблемы реализации

Как уже отмечалось, прямолинейный план реализации разделения времени – исполнять пользовательские программы в течение небольших квантов вычислений, не отдавая никому приоритета, а просто перебирая программы циклически; как будет показано ниже, стратегия разделения времени может быть и более сложной, но и эта простая схема является приемлемым решением. Однако все равно остается много проблем; одни из них лучше решать на аппаратном уровне, а другие требуют изменения соглашений и практики программирования. Ниже перечислено несколько самых очевидных проблем.

23.2.1. Аппаратные проблемы

1. Различные пользовательские программы, одновременно находящиеся в оперативной памяти, могут вмешиваться в работу друг друга и программы-супервизора, поэтому должна быть какая-то форма защиты памяти, вступающая в действие при исполнении пользовательских программ.
2. В различные моменты времени супервизор разделения времени может выполнять одну и ту же программу из разных участков памяти. (Биты перемещения на этапе загрузки не помогут, потому что супервизор не знает, как переместить аккумулятор и т. д.) Динамическое перемещение всех операций доступа к памяти, которые выбирают инструкции или слова данных, – одно из эффективных решений.
3. Оборудование ввода-вывода может быть инициировано одним пользователем, а считывать слова в память в программе другого пользователя. [Примечание редактора: т. е. без адекватного механизма защиты памяти пользователь может повредить программу другого пользователя, «про-

читав» данные с устройства ввода, а затем сохранив их в памяти, занятой другой программой.] Избежать этого можно, если перехватывать все инструкции ввода-вывода, выданные пользовательской программой, при работе в режиме защиты памяти.
4. Большое резервное запоминающее устройство с произвольным доступом желательно для хранения файлов всех пользователей. Существующих дисков большой емкости, по-видимому, будет достаточно.
5. Супервизор разделения времени должен иметь возможность прервать пользовательскую программу по исчерпании кванта вычисления. Подойдет программно инициируемый одноходовой мультивибратор, который генерирует прерывание через фиксированное время.
6. Большая оперативная память (например, миллион слов) сильно упростила бы системное программирование, потому что программы разных активных пользователей, а также часто используемые системные программы, например компиляторы, программы обслуживания запросов и т. д., могли бы постоянно находиться в памяти.

23.2.2. Проблемы программирования

1. Программа-супервизор должна автоматически начислять пользователям плату за использование машинного времени. В общем случае плата начисляется по формуле или алгоритму, учитывающему потребление системных ресурсов, который должен включать такие факторы, как время вычислений, потребление высокоскоростной памяти, аренду места во вспомогательной памяти и т. д.
2. Программа-супервизор должна координировать весь ввод-вывод, поскольку нежелательно требовать, чтобы пользовательская программа оставалась в памяти на все время выполнения операций, ограниченных вводом-выводом. Кроме того, супервизор должен координировать использование центральных разделяемых высокоскоростных устройств ввода-вывода, обслуживающих всех пользователей, а также таймеров, дисков и т. д.
3. Доступные системные программы должны быть настолько мощными, чтобы пользователь мог думать о своей задаче, не отвлекаясь на детали кодирования или исправление опечаток. Таким образом, абсолютно необходимы компиляторы, программы обработки запросов, программа анализа аварийных дампов, загрузчики и хорошие редакторы.
4. В той мере, в какой это возможно, пользователям должна быть предоставлена максимальная гибкость программирования как в плане выбора языка, так и в плане отсутствия ограничений.

23.2.3. Проблемы использования

1. Непреднамеренно может быть запрошено слишком большое вычисление или очень длинный вывод на пишущую машинку, поэтому пользователю должен быть доступен специальный сигнал снятия задачи.
2. Поскольку физическое время – не то же самое, что время использования компьютера, супервизор должен держать каждого пользователя в курсе, так чтобы тот мог правильно судить о циклах и пр.

3. Отказы процессора, памяти и ленточного накопителя неизбежны. На базовые эксплуатационные вопросы типа «Какая программа сейчас работает?» должно быть возможно получить ответ; должны быть предусмотрены процедуры восстановления после ошибки.

23.3. Экспериментальная система с разделением времени для IBM 7090

После краткого описания желательной функциональности разделения времени уместно спросить, какого уровня функциональности можно достичь на существующем оборудовании. Чтобы хотя бы попытаться ответить на этот вопрос и исследовать все аспекты программирования и эксплуатации, была разработана экспериментальная система с разделением времени. Первоначально она была написана для IBM 709, но впоследствии перенесена на модель 7090.

Компьютер 7090 в вычислительном центре МТИ, помимо трех каналов с 16 ленточными накопителями, имеет четвертый канал со стандартным интерфейсом Direct Data Connection[1]. К этому интерфейсу подключен буфер оборудования реального времени и стойка управления, спроектированные и изготовленные под руководством Х. Тигера и его группы. [В работе Teager (1962) представлен другой подход к разработке системы с разделением времени для машины 7090 в МТИ.] К этой стойке подключены разнообразные устройства, но в используемых в настоящее время системах требуются только пишущие машинки типа флексорайтер. Также на 7090 установлены две специальные модификации (т. е. RPQ[2]): стандартный генератор времени частотой 60 Гц для тарификации и прерываний и специальный режим, обеспечивающий защиту памяти, динамическое перемещение и перехват всех попыток пользователей выполнить инструкции ввода-вывода.

В текущей версии системы время разделяется между четырьмя пользователями, три из которых работают в приоритетном оперативном режиме за своими пишущими машинками, а четвертый пассивно использует работающий в фоновом режиме монитор Fap-Mad-Madtran-BSS Monitor System, аналогичный монитору FORTRAN-Fap-BSS Monitor System (FMS), который используется большинством программистов центра и во многих других местах установки 7090.

Перечислим важные проектные особенности приоритетной системы. Она позволяет пользователям:

1) разрабатывать программы на языках, совместимых с фоновой системой;

---

[1] Прямое подключение к данным. – *Прим. перев.*

[2] RPQ (Request price quotation). Так в IBM традиционно называли компоненты, которые имеются в номенклатуре, но не включены в стандартный прейскурант. – *Прим. перев.*

2) разрабатывать частные файлы программ;
3) запускать сеансы отладки в том состоянии, в котором был закончен предыдущий сеанс;
4) устанавливать собственный темп работы, в котором почти нет бесполезной траты машинного времени.

Оперативная память выделяется таким образом, что каждый пользователь работает в старших 27 000 слов, а супервизор разделения времени постоянно занимает младшие 5000 слов. Чтобы избежать конфликтов при выделении памяти, защитить пользователей друг от друга и упростить первоначальную организацию в системе 709, только один пользователь находится в оперативной памяти в каждый момент времени. Однако благодаря наличию в модели 7090 специальных механизмов защиты памяти и перемещения начата разработка более сложных процедур выделения памяти. В любом случае, время выгрузки пользователей минимизировано за счет использования двухканального ленточного накопителя с совмещением чтения и записи по адресам, занятым программами двух пользователей.

Приоритетная система организована вокруг команд, набираемых пользователями на своих пишущих машинках, и частных программных файлов пользователей, которые в настоящее время (за отсутствием дисков) хранятся на отдельных магнитных лентах для каждого пользователя.

Для удобства формат частных ленточных файлов имитирует образы перфокарт и предусматривает заглавные карты с обозначением имени и класса; его можно записать или набить на автономном оборудовании. (Последнее предлагает также примитивную форму ввода-вывода большого объема.) К магнитным лентам система предъявляет следующие требования: семь лент для нормальной работы фоновой системы, системная лента для супервизора разделения времени, содержащая большую часть командных программ, а также лента с частными файлами и лента выгрузки для каждого из трех приоритетных пользователей.

Команды, набираемые пользователем, передаются супервизору разделения времени (а не его собственной программе), поэтому могут быть инициированы в любое время независимо от того, находится программа конкретного пользователя в памяти или нет. В силу тех же причин – необходимости координации – супервизор обрабатывает весь ввод-вывод приоритетных системных пишущих машинок. Команды состоят из сегментов, разделенных вертикальной чертой; первый сегмент содержит имя команды, все остальные – ее параметры. Каждый сегмент состоит из последних шести набранных символов (и начинается неявными 6 пробелами), так чтобы было проще исправить опечатки. Возврат каретки служит сигналом для исполнения команды. Когда супервизор получает команду, на машинке печатается слово «WAIT», а после завершения команды – слово «READY». (Ответы компьютера всегда печатаются цветом, отличным от цвета команд пользователя.) В процессе печати неполную командную строку можно игнорировать, набрав последовательность «отмены», состоящую из сигнала удаления, за которым следует возврат каретки. Уже начатую команду можно снять, набрав ту же последовательность «отмены». Кроме того, нежелательную печать можно прервать, завершив команду и вывод нажатием специальной кнопки «остановить вывод».

Использование приоритетной системы инициируется всякий раз, как пользователь завершает ввод команды на своей пишущей машинке и тем самым помещает ее в очередь ожидающих команд. По исчерпании каждого кванта супервизор разделения времени отдает высший приоритет инициированию ожидающих команд. Системные программы, соответствующие большинству команд, хранятся на специальной системной ленте супервизора, поэтому, чтобы избежать непроизводительного расходования машинного времени, супервизор продолжает выполнять последнюю пользовательскую команду, пока лента позиционируется для чтения требуемой команды. В этот момент данные активного пользователя выгружаются на его буферную ленту, программа команды считывается в память, переводится в рабочее состояние и инициируется как новая пользовательская программа. Однако прежде чем выделить квант вычислений новому пользователю, супервизор еще раз проверяет наличие в очереди ожидающей команды другого пользователя и, если необходимо, загодя начинает позиционирование ленты системных команд, пока обслуживается новый пользователь.

Если очередь ожидающих команд пуста, супервизор продолжает циклически перебирать приоритетные пользовательские программы, находящиеся в очереди работающих. Наконец, если пусты обе очереди, то в память загружается фоновая пользовательская программа, которая выполняется квант за квантом, пока в очереди не появится приоритетная программа.

Приоритетные пользовательские программы покидают очередь работающих двумя способами. Если программа дошла до конца, она может войти в супервизор таким образом, что уничтожит себя и переведет пользователя в завершенное состояние. Или же программа может быть переведена в спящее состояние из-за появления другой программы (или вручную, если пользователь ввел последовательность отмены). Спящее состояние отличается от завершенного тем, что пользователь еще может возобновить свою программу или исследовать ее.

Для пользовательского ввода-вывода используются пишущие машинки, и хотя у супервизора есть несколько буферных строк, может случиться, что ввод-вывод будет ограничен. Поэтому существует еще состояние ожидания ввода-вывода, аналогичное спящему, – супервизор автоматически переводит в него пользователя, когда возникают задержки ввода-вывода. Когда буферы оказываются близки к опустошению при выводе или к заполнению при вводе, пользовательская программа автоматически возвращается в состояние работающей, и таким образом удается избежать непроизводительной траты машинного времени. <…>

Хотя на данный момент опыт работы с системой крайне ограничен, есть первые признаки того, что программисты с удовольствием пользовались бы такой системой, если бы она была доступна. Теперь, когда имеется какой-никакой опыт эксплуатации системы, полезно подвести итог сделанным наблюдениям. [Примечание: первоначальный опыт эксплуатации был получен при использовании компьютера 709; из-за сложностей при переходе на новое оборудование мы не смогли начать использование системы на логически эквивалентном компьютере 7090 к 3 мая.] Сразу же можно отметить, что стоило пользователю привыкнуть к ответам компьютера, как даже задержки,

меньшие минуты, стали казаться ему невыносимо долгими – эффект такой же, как при общении с медленно разговаривающим собеседником. Кроме того, тот факт, что минимальной единицей общения между человеком и компьютером является строка, а не отдельный символ, является сдерживающим фактором – точно так же, как говорить по радиотелефону, когда нужно каждый раз нажимать кнопку, менее естественно, чем по обычному телефону. Поскольку, чтобы компьютер мог быстро реагировать на каждый введенный символ, нужно, чтобы в оперативной памяти постоянно находилась хотя бы рудиментарная ответная программа, самым прямолинейным решением в текущей системе было бы добавление оперативной памяти. Хотя бы один дополнительный банк памяти для супервизора разделения времени помог бы снять проблемы совместимости с программами, уже написанными для компьютеров 7090 с 32 000 слов.

По соображениям целесообразности, самыми слабыми частями существующей системы являются соглашения о вводе, редактировании пользовательских файлов, а также скорость взаимодействия и степень близости к компьютеру во время отладки. Поскольку в этих отношениях большую роль играют вкусы, привычки и психология пользователей, представляется, что для правильного решения потребуется провести много экспериментов и дать прагматическую оценку; ясно также, что рассматривать эти вопросы абстрактно нельзя, потому что от используемого языка программирования зависит применяемая техника. Безусловно, желательно использовать символические ссылки для адресов, имен программ и переменных; программы анализа аварийных дампов в символическом виде, программы трассировки и программы вывода дифференциальных дампов «до и после» должны играть важную роль в процедурах отладки.

При проектировании текущей системы большое внимание было уделено обеспечению независимости пользователей друг от друга. Однако было бы полезным расширением сделать так, чтобы это было необязательно. В частности, когда несколько консолей используются в управляемой компьютером группе, например для руководства предприятием или военной игры, при изучении группового поведения и, возможно, в обучающих машинах, было бы желательно, чтобы все консоли взаимодействовали с одной программой.

Еще одна область текущей системы, где напрашиваются усовершенствования, – обслуживание файлов, потому что применяемые сейчас ленты затрудняют удаление файлов пользовательских программ. Диски стали бы здесь хорошим подспорьем, как, впрочем, и в проблеме консолидации и диспетчеризации из центра большого объема ввода-вывода, генерируемого многочисленными пользовательскими консолями.

Наконец, есть ощущение, что было бы желательно устранить различие между приоритетными и фоновыми системами. Нынешний оператор компьютера играл бы роль фоновых пользователей и использовал бы операторскую консоль так же, как все прочие пользовательские консоли в системе: для монтирования и размонтирования магнитных лент по запросам супервизора, для получения инструкций по считыванию колоды карт на центральный диск и т. д. Аналогично приоритетный пользователь, довольный своей программой, мог бы с помощью своей консоли и супервизора поместить программу

в очередь фоновых работ, ожидающих выполнения. После реализации этих процедур различие между пользователями, работающими в режиме разделения времени и в фоновом режиме, исчезло бы, и пользователь компьютера мог бы выбирать тот режим, который ему удобен в данный момент, изменяя его по собственному желанию.

23.4. Многоуровневый алгоритм планирования

Каким бы ни был объем оперативной памяти – миллион слов или 32 000 слов, как в современном компьютере 7090, – неизбежно возникает проблема насыщения, когда суммарный размер программ активных пользователей превышает размер высокоскоростной памяти или когда активных пользователей слишком много и компьютер не может достаточно быстро реагировать на запросы с каждой консоли. Такое может случиться даже тогда, когда пользователей немного, если какие-то пользовательские программы чрезмерно требовательны по размеру или потребляемому времени. Выйти из затруднения можно, постулировав, что хороший дизайн системы должен предусматривать наличие процедуры насыщения, которая обеспечивает постепенную деградацию времени реакции и эффективной скорости выполнения больших долго работающих программ.

Чтобы пояснить общую проблему, на рис. 23.1 показан качественный график обслуживания пользователей в виде функции от *n*, количества активных пользователей. Под обслуживанием может пониматься любой из двух ключевых показателей: время реакции компьютера или умноженная на *n* физическая скорость вычислений. В любом случае существует некое критическое количество активных пользователей, *N*, после которого наступает насыщение. Если вблизи точки насыщения применяется простая стратегия циклического перебора всех пользователей, то внезапно в обслуживании наступает коллапс из-за резкого возрастания относительной доли времени, потребного для выгрузки программ во вторичную память, например на диск или барабан, и загрузки оттуда. Конечно, рис. 23.1 дает только качественное представление, потому что все зависит от диапазона размеров программ и временных показателей работы каждого пользователя.

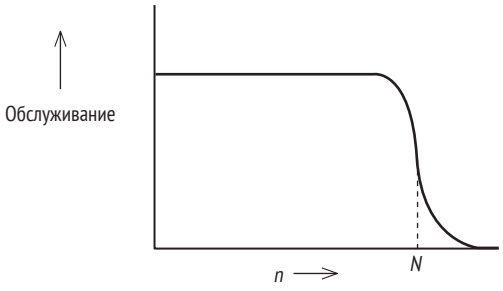


Рис. 23.1. Зависимость обслуживания от количества активных пользователей

Чтобы проиллюстрировать стратегию, позволяющую улучшить производительность системы с разделением времени в условиях насыщения, мы опишем многоуровневый алгоритм планирования. Анализ этого алгоритма дает широкие границы производительности системы. Основная идея многоуровневого алгоритма планирования – отнести каждую пользовательскую программу в момент поступления в систему для исполнения (или завершения ответа пользователю) к приоритетной очереди ℓ-го уровня. Первоначально программа попадает на уровень ℓ_0, соответствующий ее размеру и вычисляемый по формуле

$$\ell_0 = \left\lfloor \log_2 \left(\left\lfloor \frac{w_p}{w_q} \right\rfloor + 1 \right) \right\rfloor,$$

где w_p – число слов в программе, w_q – число слов, которое можно передать в высокоскоростную память из вспомогательной u обратно за один квант времени q, а скобки обозначают «целую часть». Как правило, квант, будучи базовой единицей времени, должен быть как можно меньше, но таким, чтобы относительная доля накладных расходов на переключение супервизора с одной программы на другую была невелика. В начале процесса супервизор исполняет программу, находящуюся в начале непустой очереди самого нижнего уровня ℓ, в течение времени, не превышающего 2^ℓ временных квантов, а затем, если программа не закончилась (т. е. не дала ответа пользователю), помещает ее в конец очереди уровня $\ell + 1$. Если не существует ни одной программы, поступившей в систему на уровне, меньшем ℓ, то этот процесс продолжается, пока очередь уровня ℓ не будет исчерпана; затем процесс итеративно начинается сначала на уровне $\ell + 1$, и теперь каждая программа работает в течение $2^{\ell+1}$ временных квантов. Если во время выполнения программы уровня ℓ в течение отведенных ей 2^ℓ квантов какая-то программа была помещена в очередь меньшего уровня ℓ', то программа текущего пользователя в начале ℓ-й очереди вытесняется, и процесс начинается заново с уровня ℓ'.

Аналогично, если программа размера w_p на уровне ℓ во время своей работы запрашивает изменение размера памяти у супервизора реального времени, то увеличенная (или уменьшенная) версия программы должна быть помещена в очередь уровня ℓ'', где

$$\ell'' = \ell + \left\lfloor \log_2 \left(\left\lfloor \frac{w''_p}{w_q} \right\rfloor + 1 \right) \right\rfloor.$$

И снова процесс начинается заново с пользовательской программы, находящейся в начале очереди наименьшего занятого уровня ℓ'.

В результате анализа описанного выше алгоритма мы приходим к нескольким важным заключениям, которые позволяют оценить производительность системы сверху.

1. *Вычислительная эффективность.* Поскольку программа всегда работает в течение времени, большего или равного времени вытеснения (т. е.

времени, необходимого для записи одной программы во вспомогательную память и чтения другой из вспомогательной памяти), вычислительная эффективность никогда не может опуститься ниже одной второй. (Очевидно, что эту долю можно настроить, изменив формулу для начального уровня ℓ_0.) По-другому интерпретировать эту границу можно, сказав, что реальная скорость вычислений, доступная одному из n активных пользователей, не хуже, чем если бы было $2n$ активных пользователей, все программы которых находились бы в высокоскоростной памяти.

2. *Время реакции*. Если максимальное число активных пользователей равно N, то отдельному пользователю программы заданного размера гарантируется время реакции

$$t_r \leq 2Nq\left(\left\lceil\frac{w_p}{w_q}\right\rceil+1\right),$$

т. к. худший случай имеет место, когда все конкурирующие пользовательские программы находятся на одном и том же уровне. Обратно, если t_r – произвольное гарантированное время реакции, то в предположении, что размер программы наибольший, максимально допустимое число активных пользователей ограничено сверху.

3. *Долгое выполнение*. Относительное время вытеснения при долгой работе может быть сделано сколь угодно малым. Этот вывод следует из того, что чем дольше программа работает, тем выше номер уровня, на который она переходит при соответственно меньшем относительном времени вытеснения. Важным свойством алгоритма является то, что долгие прогоны должны, по существу, доказывать, что они долгие, так чтобы неожиданно завершающиеся программы быстро обнаруживались. Чтобы число уровней было конечным, можно задать максимальный номер уровня L, так чтобы асимптотические накладные расходы на вытеснение составляли сколь угодно малую процентную долю, p:

$$L=\left\lceil\log_2\left(\left\lceil\frac{w_{pmax}}{pw_q}\right\rceil+1\right)\right\rceil,$$

где w_{pmax} – размер самой большой возможной программы.

4. *Многоуровневое и одноуровневое время реакции*. Время реакции для программ одинакового размера, поступающих в систему одновременно и исполняемых в течение нескольких квантов, не хуже, чем приблизительно удвоенное время реакции при использовании процедуры циклического перебора с одним квантом. Если имеется n программ одинакового размера, начавших работу в очереди уровня ℓ, то худшим будет случай, когда программа, находящаяся в конце очереди, готова ответить в первый же квант, выделенный уровню $\ell + j$. Если используется многоуровневый алгоритм, то полная задержка для программы в конце очереди по формуле суммы геометрической прогрессии равна:

$$T_m \sim q2^{\ell}\{n(2^j - 1) + (n-1)2^j\}.$$

Так как пользователь, находящийся в конце очереди, имел возможность выполнять вычисления в течение $2^{\ell}(2^j - 1)$ квантов, то эквивалентная задержка ответа для одноуровневого циклического перебора равна:

$$T_s \sim q2^{\ell}\{n(2^j - 1)\}.$$

Отсюда

$$\frac{T_m}{T_s} \sim 1 + \left(\frac{n-1}{n}\right)\left(\frac{2^j}{2^j - 1}\right) \sim 2,$$

и утверждение доказано. Следует отметить, что описанные выше условия с опущенными временами вытеснения, имеющие место, когда все программы остаются в высокоскоростной памяти, наименее благоприятны для многоуровневого алгоритма; если включить времена вытеснения в анализ, то отношение T_m/T_s может только уменьшиться и стать много меньше единицы. С помощью аналогичного анализа легко показать, что даже в неблагоприятных случаях, когда ни одна программа не выгружается, для программ в начале очереди, которые завершаются, как только закончатся временные кванты, при многоуровневом алгоритме время реакции оказывается в два раза меньше, чем при одноуровневом циклическом переборе (т. е. $T_m/T_s = ½$).

5. *Наивысший обслуживаемый уровень.* В многоуровневом алгоритме процедура установления уровня программы полностью автоматическая и зависит от производительности и размера программы, а не от заявлений (или надежд) каждого пользователя. По мере того как пользователь нагружает систему, качество обслуживания ухудшается постепенно, начиная с пользователей более высоких уровней, запускающих либо большие, либо долго работающие программы; однако, начиная с какого-то уровня, никакие программы выполняться не могут, потому что на более низких уровнях образовалось слишком много активных пользователей. Чтобы определить точку такого отсечения, рассмотрим N активных пользователей на уровне ℓ, каждый из которых работает в течение 2^{ℓ} квантов, завершается и снова входит в систему на уровне ℓ спустя время реакции пользователя t_u. Если никакого обслуживания на уровне $\ell + 1$ не должно быть, то время вычислений $Nq2^{\ell}$ должно быть больше или равно t_u. Таким образом, количество гарантированно активных уровней определяется неравенством:

$$\ell_a \leq \left\lfloor \log_2\left(\frac{t_u}{Nq}\right)\right\rfloor.$$

В пределе t_u могло бы быть столь же мало, как минимальное время реакции пользователя (~0,2 с), но ожидаемое значение будет на не-

сколько порядков больше вследствие статистики по большому числу пользователей.

В описанном выше многоуровневом алгоритме явно не учитывается время подвода головки или время задержки, которые проходят перед началом передачи на диск или барабан либо в обратном направлении, если эти устройства используются в качестве вспомогательной памяти (хотя формально коэффициент w_q мог бы содержать усредненное время). Алгоритм можно слегка модифицировать, что позволит избежать непроизводительного расходования указанного выше времени, – нужно просто продолжать выполнение последней пользовательской программы в течение стольких квантов, сколько нужно для вытеснения самого низкоприоритетного пользователя новым; т. к. обычно во вспомогательную память выгружаются только программы с более высоким номером уровня, продление времени работы старого пользователя, пока ищется новый, не должно сильно исказить базовый алгоритм.

Дополнительные усложнения возможны при наличии подходящего оборудования. Если компьютер оснащен каналами ввода-вывода, а скорость передачи во вспомогательную память и обратно низкая, то можно совместить чтение новых пользователей в высокоскоростную память и запись старых во вспомогательную, обслуживая в это время текущего пользователя. Эффект получается таким же, как при использовании барабана со 100-процентным коэффициентом мультиплексирования, но есть два подвоха: во-первых, никакой пользователь не может использовать всю доступную пользователям память, а во-вторых, процедура упреждающего поиска сбоит, если происходит непредвиденное изменение планирования (например, программа завершается или инициируется пользовательская программа с более высоким приоритетом).

Можно также усложнить выделение памяти, но, конечно, в случае, когда скорость передачи во вспомогательную память низкая, элементарной и желательной процедурой является объединение в один блок всех высокоприоритетных пользовательских программ, когда имеется достаточно фрагментированной неиспользуемой памяти для чтения новой пользовательской программы. Такая процедура присутствует на блок-схеме многоуровневого алгоритма планирования на рис. 23.2.

Следует также отметить, что на рис. 23.2 учтено только планирование программ в работающем состоянии и не учтено выделение памяти программами в спящем (или ожидающем завершения ввода-вывода) состоянии. Систематический метод обработки этого случая заключается в том, чтобы модифицировать алгоритм планирования, так чтобы программы, перешедшие в спящее состояние на уровне ℓ, попадали в очередь на уровне $\ell + 1$. Алгоритм планирования работает, как и прежде, спящие программы продолжают каскадное перемещение, но не выполняются, пока не достигнут начала очереди. Если требуется удалить какую-то программу из высокоскоростной памяти, то выбирается программа, находящаяся в конце непустой очереди с наибольшим номером уровня.

Наконец, очень интересно применить границы многоуровневого алгоритма планирования к современному компьютеру IBM 7090. Получены следующие приближенные значения:

q = 16 мс (в предположении накладных расходов на переключение 1 %);

w_q = 120 слов (в предположении одного диска IBM 1301 модели 2 без учета времени подвода головки или задержки);

$t_r \leqslant 8Nf$ с (в предположении, что программы занимают (32K) f слов);

$\ell_a \leqslant \log_2(1000/N)$ (в предположении, что t_u = 16 с);

$\ell_0 \leqslant 8$ (в предположении, что максимальный размер программы равен 32K слов).

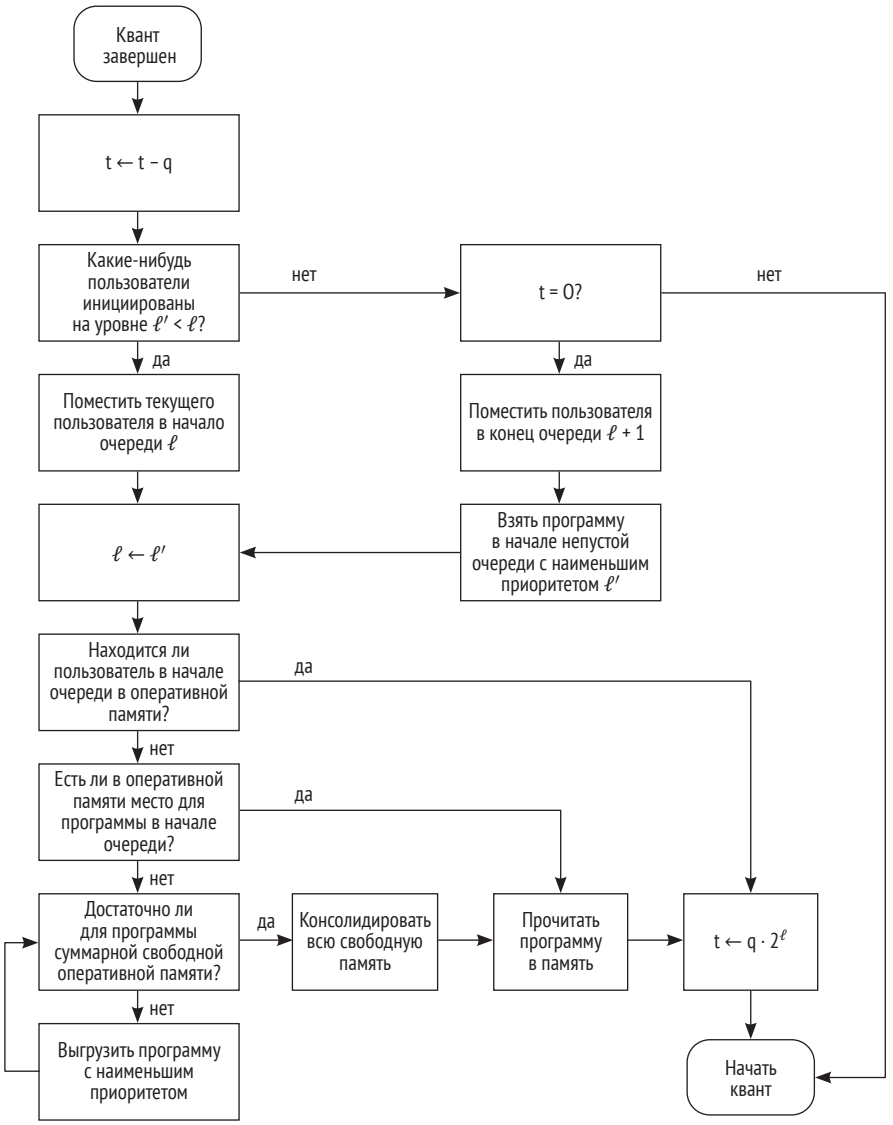


Рис. 23.2. Блок-схема многоуровневого алгоритма планирования

Приняв произвольный критерий, согласно которому программы, размер которых не превышает 32 000 слов, всегда должны получать какое-то обслуживание, или, что то же самое, $\max \ell_a = \max \ell_0$, получаем в качестве консервативной оценки, что N может быть равно 4 и что в худшем случае время реакции для тривиального ответа будет составлять 32 секунды.

Малое полученное значение N является прямым следствием малого значения w_q, которое, в свою очередь, обусловлено низкой скоростью передачи на диск. Эта скорость составляет лишь 3,3 % максимальной скорости мультиплексирования оперативной памяти. Интересно, что современные высокоскоростные барабаны большой емкости, такие, например, как в системе SAGE или в IBM Sabre, позволили бы добиться почти 100-процентного использования мультиплексора и, значит, увеличить w_q в 30 раз. Отсюда сразу вытекает, что время реакции, эквивалентное приведенному выше для диска, можно было бы обеспечить в 30 раз большему числу пользователей, т. е. 120 пользователям; однако общая вычислительная мощность при этом не изменилась бы.

В любом случае подходить к оценкам вычислительной мощности и времени реакции компьютера следует с большой осторожностью, потому что они критически зависят от функций распределения времени реакции пользователя, t_u, от размера пользовательской программы и от вычислительной мощности, запрашиваемой каждым пользователем. Прошлый опыт работы с традиционными системами программирования мало чем помогает, потому что функции распределения очень сильно зависят от систем программирования, доступных пользователям с разделением времени, а также от сформировавшихся привычек пользователей, которые будут изменяться постепенно.

23.5. Выводы

Итак, ясно, что современных компьютеров и оборудования достаточно, чтобы обеспечить режим разделения времени с умеренной производительностью для ограниченного числа пользователей. Есть несколько проблем, которые можно разрешить посредством тщательного проектирования оборудования, но также предстоит написать немало сложных системных программ, прежде чем систему с разделением времени можно будет признать адекватной. Важный аспект любого будущего компьютера с разделением времени заключается в том, что пока не завершено системное программирование, и в первую очередь супервизора разделения времени, компьютер абсолютно бесполезен. Таким образом, прежде чем заниматься проектированием и реализацией будущей системы, необходимо исследовать и понять все особенности систем с разделением времени на прототипе на базе имеющихся компьютеров, и только так можно будет добиться значительных успехов в организации и использовании компьютеров.

24 Sketchpad (1963)

Айвен Э. Сазерленд

Еще подростком Айвен Сазерленд (родился в 1938 году) обертывал учебники синьками, которые выбрасывал его отец, инженер-строитель. (Мать говорила, что семья не могла позволить себе обложки с символикой колледжа, которыми пользовались одноклассники.) Он изучал краткий, элегантный язык чертежей, разглядывая обложки своих учебников, когда в классе было скучно. Позже, учась на техническом факультете, он должен был не только читать чертежи, но и чертить самостоятельно, а вот сноровки и терпения для этого ему не хватало. Поэтому, когда он поступил в аспирантуру в МТИ и получил машинное время на компьютере TX-2, ему пришла в голову мысль взять в качестве темы для докторской диссертации написание программы, которая будет помогать в черчении. Сазерленд был знаком с провидческими предсказаниями Буша (глава 11) и Ликлайдера (глава 20), но ничего из того, что они навоображали, так и не было построено. Эта диссертация была написана под руководством Клода Шеннона и положила начало целой новой области: машинная графика и интерактивные вычисления.

Компьютер TX-2 был построен на транзисторах и в то время был самой мощной вычислительной машиной в мире. Он располагал памятью размером 64K 36-разрядных слов – в два раза больше, чем на любой другой машине. У него также имелся примитивный дисплей – просто круглая электронно-лучевая трубка и машинная инструкция, позволявшая высветить точку с заданными координатами. (Такие дисплеи не были чем-то новым, они уже использовались при конструировании EDVAC. См. анализ Ликлайдера ограничений технологий производства дисплеев в 1960 году на стр. 275.) Не было ни растра, ни возможности рисовать линии или отображать символ целиком; чтобы нарисовать линию, программа должна была вычислить координаты последовательных точек, а затем быстро подсветить эти точки одну за другой и повторять этот процесс, создавая иллюзию, что вся линия отображается сразу. Более сложные изображения можно было рисовать на экране таким же

образом, при условии что весь процесс занимает не дольше одной тридцатой секунды, иначе возникало раздражающее мелькание.

Для создания чертежной программы Сазерленду также было нужно устройство графического ввода. В TX-2 имелось световое перо – фотоэлемент, подключенный к компьютеру электрическим проводом. Если фотоэлемент располагался над частью изображения на экране, то программа могла определить его местонахождение, сопоставляя время, когда фотоэлемент стал активным, с позицией точки, отображаемой в этот момент. Рисуя крестик в этой позиции и проверяя, какие части крестика видны фотоэлементу в соседние моменты времени, программа могла определить направление движения светового пера.

Все эти приспособления были заменены лучшими в следующем десятилетии – волоконно-оптические кабели, планшеты, сенсорные экраны, растровые дисплеи, цветные дисплеи и т. д. Долговечным интеллектуальным вкладом Сазерленда была форма симбиоза между человеком и машиной – именно то, что предсказывал Ликлайдер. Sketchpad позволяла пользователю формулировать ограничения, например что прямые должны быть параллельны или пересекаться в конечных точках. Программа старалась удовлетворить этим ограничениям при перемещении или изменении формы объектов на экране. Таким образом, она понимала не только геометрию, но и топологию чертежа. В Sketchpad были встроены средства группировки, дублирования, вращения и перемещения объектов. Это была первая программа автоматизированного проектирования. А еще замечательнее тот факт, что применение в ней таких новаторских методов программирования, как удовлетворение ограничений и иерархии объектов, посеяло семена как непроцедурного, так и объектно-ориентированного проектирования систем.

Когда в 1963 году Сазерленд посетил вертолетную компанию Белла (Bell Helicopter Company) в Техасе, ему показали хитроумное приспособление, помогающее пилотам садиться ночью. Инфракрасная камера, смонтированная на днище вертолета, автоматически поворачивалась синхронно с движениями головы пилота, а изображение с нее через призмы проецировалось в поле его зрения. Сазерленд предложил заменить изображение с инфракрасной камеры изображением, сгенерированным компьютером, и так родилась первая система виртуальной реальности, построенная в 1968 году в Гарварде. Крепящийся на голове дисплей был подключен к потолку телескопическими трубками и позволял носителю видеть каркасный куб, парящий в пространстве, проходить сквозь него и вокруг него. Затем Сазерленд сконструировал графическую систему для первого практического полетного тренажера от компании Evans and Sutherland Computer Company, штат Юта. Когда в следующем десятилетии машинная графика достигла невероятного правдоподобия, Сазерленд переключился на проектирование сверхбыстрых электронных схем и до сих пор активно работает в этой области.

Еще одна деталь его биографии оказалась очень важна для развития информатики. Когда в 1964 году Дж. К. Р. Ликлайдер ушел из ARPA в бизнес, его заменил только-только защитивший диссертацию Сазерленд – во исполнение обязательств перед Министерством обороны, принятых на себя в обмен на получение стипендии по программе вневойсковой подготовки офицеров

резерва. Во время службы Ликлайдера и Сазерленда в ARPA финансирование перспективных инициатив стимулировало исследования в области информатики на всей территории США. И это тоже внесло свой вклад в формирование облика будущего.

24.1. Краткое описание

В системе Sketchpad рисование используется в качестве новаторской среды для взаимодействия с компьютером. Система включает программы ввода, вывода и вычислений, позволяющие ей интерпретировать информацию, рисуемую прямо на экране. Она применялась для создания электрических, механических, научных, математических и анимированных изображений; это система общего назначения. Особенно полезной Sketchpad оказалась в качестве вспомогательного средства для понимания процессов, например, стержневых конструкций, которые можно изобразить на рисунке. Sketchpad также упрощает построение часто повторяющихся или особенно точных чертежей и позволяет легко изменять чертежи, ранее построенные с ее помощью. Многие чертежи в этой диссертации были исполнены в Sketchpad.

Пользователь Sketchpad рисует прямо на дисплее компьютера «световым пером». Оно используется как для позиционирования частей чертежа на экране, так и для того, чтобы указать на часть, подлежащую изменению. Изменения управляются кнопками, например «стереть» или «переместить». Если не считать надписей, текстовая информация нигде не используется. Чертеж может состоять из отрезков прямых или дуг окружностей. Можно определять произвольные символы, составленные из отрезков прямых, дуг окружностей и ранее определенных символов. Пользователь может определить и использовать столько символов, сколько пожелает. Любое изменение в определении символа становится видно сразу, как только символ появляется на экране.

Sketchpad хранит явную информацию о топологии чертежа. Если пользователь переместит одну вершину многоугольника, то автоматически изменят положение соседние стороны. Если пользователь переместит символ, то все присоединенные к этому символу линии автоматически сдвинутся, оставаясь присоединенными. Топологические связи на чертеже автоматически создаются, когда пользователь строит его. Поскольку Sketchpad умеет распознавать топологическую информацию, вводимую на языке, совершенно естественном для пользователя, ее можно использовать как программу ввода для вычислительных программ, требующих топологических данных, например симуляторов электрических схем.

Sketchpad может самостоятельно перемещать части чертежа, чтобы удовлетворить новым условиям, заданным пользователем. Пользователь задает условия с помощью светового пера и кнопок. Например, чтобы сделать две прямые параллельными, он последовательно указывает на прямые световым

пером и нажимает кнопку. Сами условия отображаются на чертеже, чтобы их можно было стирать или изменять на языке светового пера. Произвольную комбинацию условий можно определить как составное условие и применить за один шаг.

В словарь Sketchpad легко добавлять совершенно новые типы. Поскольку условия могут включать все, что допускает вычисление, Sketchpad можно использовать для решения очень широкого круга задач. Например, Sketchpad применялась, чтобы найти распределение сил в элементах мостов со сквозными фермами, которые с ее же помощью и были вычерчены.

Чертежи, созданные Sketchpad, хранятся в памяти компьютера в виде специально спроектированной «кольцевой» структуры. Такая структура обеспечивает быструю обработку топологической информации вообще без поиска. Основные операции с кольцевой структурой в Sketchpad описаны ниже.

24.2. Введение

Система Sketchpad позволяет человеку быстро общаться с компьютером с помощью среды для рисования линий. До сих пор взаимодействие между людьми и компьютерами было медленным из-за необходимости сводить его к письменным предложениям, которые еще нужно напечатать; в прошлом мы писали буквы для того, чтобы общаться со своими компьютерами. Для многих типов взаимодействия, например при описании механической детали или соединений на электрической схеме, напечатанные предложения оказываются слишком громоздки. Система Sketchpad отказывается от ввода текста (кроме надписей на чертежах) в пользу рисунков и тем самым открывает новую область во взаимодействии человека с компьютером.

Решение реализовать чертежную систему отражало наше ощущение, что понять, какие средства окажутся полезными, можно, только попробовав их в деле. Но это решение не означало, что нужно тупо компьютеризировать обычные чертежные инструменты; исследовательский характер работы неявно предполагал, что следует изобрести простые новые средства, которые, будучи построены, окажутся полезными для широкого круга применений, в т. ч. непредвиденных – и это было бы лучше всего. Выяснилось, что черчение на компьютере кардинально отличается от черчения на бумаге не только точностью, простотой проведения линий и скоростью стирания, но и прежде всего возможностью перемещать части чертежа на экране, не стирая их. Если бы не была разработана работающая система, то наше мышление было бы слишком сильно зашорено многолетними привычками черчения на бумаге, и мы не смогли бы открыть многие полезные возможности, которые может предоставить компьютер.

По ходу работы было открыто и реализовано несколько простых и применимых к очень широкому кругу задач средств. Это возможность создания субизображений путем включения произвольных символов в чертеж, возможность определения ограничений, связывающих части чертежа любым допускающим вычисление способом, и возможность копировать определе-

ния, что позволяет строить сложные связи, комбинируя простые атомарные ограничения. В сочетании с возможностью указывать на части картинки световым пером субизображения, ограничения и копирование определений придают системе необычайную мощь. Как мы и надеялись в самом начале, система оказалась полезной для широкого спектра применений, и постоянно открываются новые непредвиденные пути ее использования.

Чтобы понять, на что система способна уже сегодня, рассмотрим ее применения для рисования гексагональной решетки, изображенной на рис. 24.1. Мы будем подавать специальные команды с помощью кнопок, включать и выключать функции с помощью переключателей, обозначать позицию и указывать на уже созданные части рисунка световым пером, поворачивать и увеличивать части рисунка с помощью поворотных ручек и наблюдать за процессом рисования на экране дисплея. Это оборудование, входящее в состав компьютера TX-2, созданного в лаборатории Линкольна (Clark et al. 1957), показано на рис. 24.2. Законченный рисунок можно напечатать на бумаге с помощью графопостроителя (EAI, 1959), показанного на рис. 24.3; именно так были созданы все рисунки в этой диссертации. На данном примере мы хотим продемонстрировать, как компьютер может помочь в рисовании; детали же того, как он осуществляет свои функции, будут изложены в следующих главах.

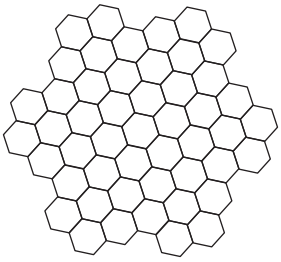


Рис. 24.1. Гексагональная решетка

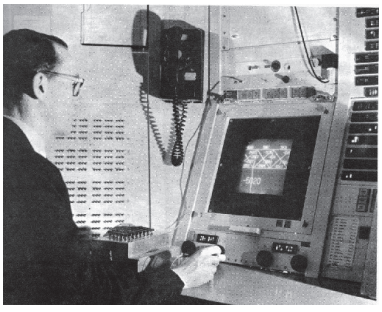


Рис. 24.2. Операционная область TX-2 – Sketchpad в действии. На экране видна часть моста. <...> Автор держит световое перо. Кнопки, предназначенные для управления функциями рисования, находятся на панели, лежащей перед автором. Часть банка переключателей видна позади автора. Размер и положение части общей картины, видимой на экране, задаются четырьмя черными ручками прямо над столом

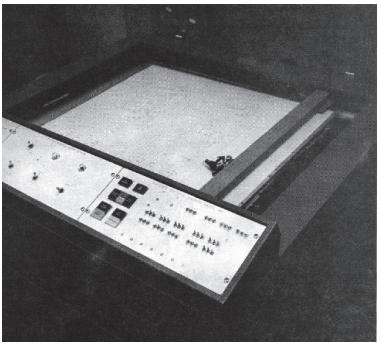


Рис. 24.3. Использование графопостроителя совместно со Sketchpad. Цифроаналоговая система управления заставляет графопостроитель проводить прямые линии и окружности под непосредственным управлением TX-2 или автономно с перфоленты

24.2.1. Вступительный пример. Если указать световым пером на экран дисплея и нажать кнопку «рисовать», то компьютер будет строить отрезок

прямой, который, как резинка, будет тянуться из начальной точки в точку, где в данный момент находится световое перо (рис. 24.4). Дополнительные нажатия кнопки порождают новые отрезки – всего шесть, достаточно для построения одного шестиугольника. Чтобы замкнуть фигуру, мы помещаем световое перо рядом с точкой, из которой начинали рисование, и конечная точка совмещается с начальной. Быстрая вспышка пера завершает рисование, оставив на экране неправильный замкнутый шестиугольник, показанный на рис. 24.5A.

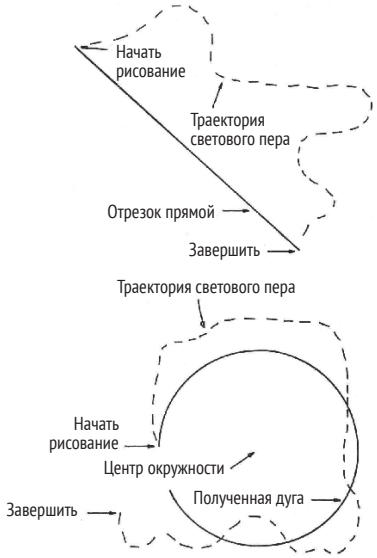


Рис. 24.4. Рисование прямой и окружности

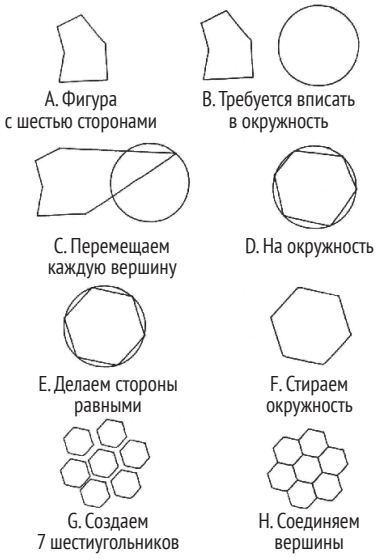


Рис. 24.5. Иллюстративный пример

Чтобы сделать шестиугольник правильным, мы можем вписать его в окружность. Для рисования окружности поместим световое перо в точку, где должен находиться центр, и нажмем кнопку «центр окружности», в результате на экране останется центральная точка. Теперь выберем точку на окружности (зафиксировав тем самым радиус) и снова нажмем кнопку «рисовать»; на этот раз будет рисоваться дуга окружности, длина которой контролируется световым пером, как показано на рис. 24.4. Затем переместим шестиугольник внутрь окружности, указав на одну из его вершин и нажав кнопку «переместить». Тогда вершина будет следовать за световым пером, ведя за собой два отрезка прямых. Указав на окружность и дав завершающую вспышку, мы показываем, что вершина должна лежать на окружности. Продолжая таким же образом, мы перемещаем все вершины на окружность, располагая их примерно на одинаковом расстоянии друг от друга, как показано на рис. 24.5D.

Мы указали, что все вершины шестиугольника должны лежать на окружности и оставаться на ней во время последующих манипуляций. Если мы еще

настаиваем на равенстве сторон, то будет построен правильный шестиугольник. Для этого укажем на одну сторону и нажмем кнопку «копировать», затем укажем на другую сторону и дадим завершающую вспышку. В этом случае кнопка копирует определение отрезков равной длины и применяет его к указанным отрезкам. По сути дела, мы сказали, что один отрезок должен иметь такую же длину, как другой. С помощью пяти таких команд мы добьемся, что все шесть отрезков будут иметь одинаковую длину. В итоге мы получим правильный шестиугольник, вписанный в окружность. Теперь окружность можно стереть, указав на любую ее часть и нажав кнопку «удалить». Получившийся шестиугольник показан на рис. 24.5F.

Чтобы создать гексагональную сетку, изображенную на рис. 24.1, мы хотим соединить много шестиугольников в вершинах, поэтому назначаем все шесть вершин нашего шестиугольника точками присоединения, для чего указываем на каждую из них пером и нажимаем кнопку. Теперь можно отложить базовый шестиугольник в сторону и начать работу с чистого «листа бумаги», изменив положение переключателя. На новом листе мы осуществляем сборку. Для этого нажатием кнопки создаем каждый шестиугольник в виде субизображения, располагая шесть шестиугольников вокруг находящегося в центре седьмого примерно так, как показано на рис. 24.5G. Каждое субизображение целиком можно позиционировать с помощью светового пера, повернуть или масштабировать с помощью ручек и зафиксировать в определенном положении, дав завершающую вспышку; их форма при этом остается неизменной. Если теперь указать на вершину одного шестиугольника, нажать кнопку и указать на вершину другого, то мы скрепим обе вершины вместе, поскольку они были назначены точками присоединения. Если присоединить обе вершины каждого внешнего шестиугольника к подходящим вершинам внутреннего шестиугольника, то все семь окажутся связанными, и компьютер позиционирует их, как показано на рис. 24.5H. Собранную таким образом группу шестиугольников можно рассматривать как символ. Группу как единое целое можно вызвать на другом «листе бумаги» и соединить с другими группами или отдельными шестиугольниками, создав очень большую сетку. Повторив элемент, изображенный на рис. 24.5H, семь раз, мы получим сетку, показанную на рис. 24.1. Построение такой сетки в Sketchpad занимает меньше пяти минут.

24.2.2. Интерпретация вступительного примера. В рассмотренном выше вступительном примере мы видели, как рисовать прямые и окружности и как перемещать части рисунка по экрану. Мы использовали световое перо как для того, чтобы позиционировать рисунок, так и для указания на его части. Например, мы указывали на окружность, чтобы стереть ее, а при рисовании шестого отрезка указывали на начало первого, чтобы замкнуть шестиугольник. Мы также видели в действии очень общие средства системы: *ограничение*, *субизображение* и *копирование определения*.

24.2.2.1. Субизображение. Исходный шестиугольник мог бы с тем же успехом быть чем-то еще: изображением транзистора, роликового подшипника, крыла самолета, буквы или целого рисунка для этого отчета. При желании можно нарисовать сколько угодно различных символов, выраженных в тер-

минах более простых символов, причем любой символ можно использовать сколько угодно раз.

24.2.2.2. Ограничение. Когда мы просили расположить вершины шестиугольника на окружности, мы пользовались базовой связью между частями изображения, встроенной в систему. В систему встроены следующие базовые связи (атомарные ограничения): сделать прямые вертикальными, горизонтальными, параллельными или перпендикулярными; поместить точки на прямые или окружности; расположить символы вертикально друг под другом или сделать их одинакового размера; связать символы с другими частями изображения, например точками либо отрезками. Новые типы ограничений программировать настолько просто, что набор атомарных ограничений был расширен с пяти до семнадцати, перечисленных в приложении А, всего за два дня; специализированные ограничения можно добавлять по мере необходимости. [Примечание редактора: приложение опущено.]

24.2.2.3. Копирование определения. Во вступительном примере мы просили сделать стороны шестиугольника одинаковыми по длине, для чего нажимали кнопку, указав на сторону. Тем самым мы воспользовались встроенной в систему возможностью копировать определение. Если бы мы определили составную операцию, например сделать два отрезка параллельными и равными по длине, то могли бы применить ее столь же просто. Количество операций, которые можно определить на основе базовых ограничений, применяемых к различным частям изображения, почти не ограничено. Новые полезные определения появляются регулярно; они могут быть такими простыми, как рисование горизонтальных прямых, или такими сложными, как рисование размерных линий со стрелками и числом, обозначающим длину линии. Средство копирования определения упрощает использование ограничений.

24.2.3. Выводы из вступительного примера. Как мы видели во вступительном примере, рисование в системе Sketchpad отличается от рисования карандашом на бумаге. Самое главное, чем отличается рисование в Sketchpad, – отсутствие следа, оставляемого грифелем на листе бумаги. Информация о том, как связаны между собой части чертежа, хранится в компьютере, равно как и информация о том, как выглядит рисунок. Поскольку части чертежа взаимосвязаны, он будет сохранять полезную форму даже при перемещении отдельных частей. Например, когда мы перемещали вершины шестиугольника на окружность, стороны, примыкающие к вершине, перемещались автоматически, так что шестиугольник оставался замкнутым. А потом, когда мы указали, что вершины шестиугольника должны лежать на окружности, они там и оставались на протяжении всех последующих манипуляций.

Именно возможность сохранять информацию о связи между частями чертежа и делает Sketchpad такой полезной. Например, конструкция, показанная на рис. 24.6, была нарисована в Sketchpad всего за несколько минут. К этой конструкции были применены ограничения, фиксирующие длины различных элементов. Предполагается, что поворот короткого центрального звена должен перемещать левый конец пунктирной линии по вертикали. Так

как точная информация о свойствах конструкции сохранена в Sketchpad, мы можем наблюдать движение всей конструкции при повороте короткого центрального звена. На значение числа на рис. 24.6 наложено ограничение – показывать длину пунктирной линии, сравнивая фактическое движение с вертикальной прямой в правой части чертежа. Можно заметить, что для всех положений конструкции длина пунктирной линии постоянна, т. е. конструкция действительно движется только по прямой. Другие примеры перемещения чертежей, выполненных в Sketchpad, приведены в последней главе.

Sketchpad хранит не только информацию о связи между различными частями чертежа, но и структуру использованных субизображений. Например, в хранимой информации о гексагональной решетке на рис. 24.1 указано, что она состоит из меньших элементов, которые сами состоят из отдельных шестиугольников. Если изменить главный шестиугольник, то изменится и вся гексагональная решетка. Но структура решетки, конечно же, сохранится. Например, если заменить базовый шестиугольник полуокружностью, то мгновенно появится сетка, напоминающая рыбью чешую (рис. 24.7).

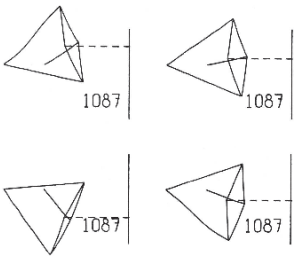


Рис. 24.6. Четыре положения конструкции. Число показывает длину пунктирной линии

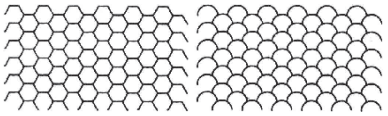


Рис. 24.7. Половинки шестиугольников и полуокружности, являющиеся элементами одной и той же решетки

Поскольку Sketchpad хранит структуру чертежа, этот чертеж явно указывает на сходство символов. Например, на электрической схеме все символы транзисторов созданы из одного главного чертежа транзистора. Если как-то изменить этот главный символ, то изменение сразу же распространится на все символы транзисторов без каких-либо дополнительных усилий. Самое главное – компьютер «знает», что в этом месте схемы имелся в виду «транзистор». Ему нет нужды интерпретировать набор линий, в котором мы легко узнаем символ транзистора. Так как Sketchpad хранит топологию чертежа, как мы видели при замыкании шестиугольника, то при рисовании в Sketchpad пользователь одновременно указывает и как схема выглядит, и какие в ней электрические соединения. То, что соединения схемы действительно хранятся, видно из того, что перемещение компонента автоматически перемещает всю связанную с ним разводку, чтобы сохранить правильность соединений. Чертежи схем, сделанные в Sketchpad, скоро будут использоваться как входные данные для симуляторов схем. Имея чертеж схемы, можно будет определить ее электрические характеристики.

24.2.4. Sketchpad и процесс проектирования. Построение чертежа в Sketchpad само по себе является моделью процесса проектирования. Положения точек и линий на чертеже моделируют переменные проекта, а геометрические ограничения на точки и линии моделируют проектные ограничения на значения переменных проекта. Способность Sketchpad удовлетворять геометрическим ограничениям на части чертежа моделирует умение хорошего конструктора удовлетворять всем проектным условиям, налагаемым ограничениями на материалы, стоимость и т. д. На самом деле, поскольку во многих областях конструкторы сами не производят ничего, кроме чертежа детали, проектные условия можно рассматривать как применяемые к чертежу детали, а не к самой детали. Если добавить такие проектные условия в словарь ограничений Sketchpad, то компьютер мог бы помочь пользователю не только красиво исполнить чертеж, но и выбрать правильную конструкцию.

24.2.5. Полезность на текущем этапе. В начале этого исследования никто не рисовал технические чертежи прямо на компьютерном дисплее, используя средства, хотя бы отдаленно похожие на те, что мы имеем сейчас, а потому никто и не знал, как это будет выглядеть. Теперь мы накопили примерно сто часов опыта фактического исполнения чертежей в работающей системе. Как показано в последней главе, методы черчения на компьютере нашли применение в различных задачах. По мере роста числа приложений стало ясно, что свойства чертежей, созданных в Sketchpad, делают их наиболее полезными в следующих четырех широких сферах применения.

24.2.5.1. Для внесения мелких изменений в имеющиеся чертежи. Описание каждого чертежа хранится в компьютере в форме, легко допускающей перенос на магнитную ленту. Со временем нарабатывается библиотека чертежей, части которой можно использовать в других чертежах, затратив на эту лишь малую долю времени, вложенного в создание первоначального чертежа. Поскольку чертеж, хранящийся в компьютере, может содержать явное представление проектных условий в виде ограничений, изменение важной части вручную автоматически повлечет за собой соответствующие изменения в связанных с ней частях.

24.2.5.2. Для лучшего научного или технического понимания операций, которые можно описать графически. Описание чертежа, хранящееся в системе Sketchpad, – нечто большее, чем собрание статических частей, прямых, кривых и т. д. Чертеж может содержать явные утверждения о связях между частями, так что при изменении одной части становятся очевидны вызванные этим изменения в других частях. Как мы видели на рис. 24.6, отрезкам можно приписать свойство фиксированной длины и таким образом изучать механические конструкции, наблюдая траектории одних частей при перемещении других. На рис. 24.7 мы видели, что любое изменение в определении субизображения сразу же отражается на внешнем виде этого субизображения, где бы оно ни встретилось. Внося такие изменения, можно понять связи между сложными наборами субизображений. Например, можно изучить, как изменение базового элемента кристаллической структуры отражается на кристалле в целом.

24.2.5.3. Как топологическое устройство ввода для симуляторов схем и т. д. Поскольку кольцевая структура хранения в Sketchpad отражает топологию схемы или диаграммы, она может служить входными данными для многих программ моделирования сетей или электрических схем. Дополнительные усилия, необходимые для перечерчивания всей схемы в Sketchpad, могут с лихвой окупиться, если путем симуляции нарисованной схемы удастся установить ее свойства.

24.2.5.4. Для чертежей с высоким уровнем повторения. Способность компьютера воспроизводить любой нарисованный символ в любом месте одним нажатием кнопки и рекурсивно включать субизображения в состав субизображений упрощает построение чертежей, состоящих из огромного числа деталей примерно одинаковой формы. Большой интерес к такому применению проявляют люди, занятые разработкой запоминающих устройств и логических микросхем, где нужно генерировать сразу много элементов с помощью фотографических процессов. Легко можно получить эталонные чертежи для повторяющихся структур. И в этом случае способность распространить изменение в одном элементе повторяющейся структуры сразу на все такие подэлементы позволяет изменять элементы матрицы, не перечерчивая ее целиком. <…>

24.3. История Sketchpad

<…> Если бы мне пришлось выполнять эту работу снова, я начал бы с нуля, уже твердо зная, что наличие общей структуры, разделение подпрограмм на универсальные, применимые к частям рисунка любого типа, и специальные, применимые только к конкретным типам частей, а также неограниченная применимость функций (например, все должно быть перемещаемым) с лихвой окупили бы все усилия, потраченные на их достижение. Я искренне восхищаюсь людьми, которые все время твердили мне про это, но я должен был, спотыкаясь, сам пройти по тропинке, описанной в этой главе, чтобы лично во всем убедиться. Остается надеяться, что будущие исследователи либо сумеют сразу оценить всю мощь общности и стремиться к ней всеми силами, либо будут иметь мужество пройти по той же дорожке, что и я, и так достичь понимания. <…>

У этой системы есть потенциал, о котором мы еще даже помыслить не можем. Богатство возможностей, которые несет с собой функция копирования определения, и новые типы ограничений, которые легко добавить в систему для специальных целей, позволяют предположить, что будущие приложения принесут с собой целую новую область знаний о применениях системы. Так, о примерах проектирования мостов, приведенных в конце этой статьи, изначально никто не думал. Конечно, у системы имеются ограничения. В последней главе предложен ряд усовершенствований: одни из них – просто мелкие изменения, другие – крупные добавления, которые, возможно, изменят характер системы целиком. Надеюсь, что будущее далеко превзойдет мои усилия.

25 Упаковка большего числа компонентов на интегральной схеме (1965)

Гордон Мур

Транзистор был одним из величайших изобретений XX века. Небольшие, ненагревающиеся, надежные и потребляющие очень мало энергии, транзисторы к середине 1950-х годов стали заменять электровакуумные лампы во многих приложениях. Первые карманные радиоприемники на батарейках появились примерно в 1954 году; это был потрясающий шаг вперед по сравнению с проводными настольными радиоприемниками с их светящимися лампами. Вскоре стало понятно, что транзисторы можно использовать не только как усилители, но и как переключатели, и это послужило стимулом для проектирования электронных компьютеров типа TX-2.

Но пока еще никто не видел признаков настоящей надвигающейся революции: миниатюризации. В начале 1960-х годов появилась возможность массово производить электронные схемы целиком методом фотолитографии. Поначалу на кремниевых кристаллах умещалась всего горстка транзисторов, но их число быстро росло. В 1965 году журнал «Electronics» – коммерческий, а не научный – попросил Гордона Мура (1929–2023) дать прогноз развития полупроводниковой техники на следующее десятилетие. Эта статья стала его ответом. В то время Мур занимал пост директора по исследованиям и разработкам в компании Fairchild Semiconductor и, опираясь на собственные наблюдения за совершенствованием проектирования и изготовления микросхем, предсказал, что число компонентов на кристалле будет удва-

иваться ежегодно. (Позже он понизил прогноз, ограничившись удвоением каждые два года.)

Этот прогноз вскоре стал восприниматься как вызов; он, безусловно, не является и никогда не был «законом», как его прозвали. Поразительно, что прогноз оставался верным на протяжении тридцати или сорока поколений электронной техники, хотя с самого начала было очевидно, что в конечном итоге размеры транзисторов и их межсоединений станут сравнимы с размерами молекул, из которых они состоят. Когда эта точка стала совсем близко, «закон» обобщили, интерпретировав его как увеличение быстродействия, или вычислительной мощности, достигаемое не более плотной упаковкой компонентов, а какими-то другими способами.

Как бы там ни было, оригинальная статья Мура является на удивление провидческой тем, что привлекла внимание к проблеме выхода годных в производственном процессе, к неизбежному появлению варианта электронного компьютера, ориентированного на потребительский рынок и пр.

В 1968 году Мур и Роберт Нойс основали корпорацию Intel, до сих пор являющуюся одним из крупнейших производителей полупроводников в мире.

Будущее интегральной электроники – это будущее самой электроники. Преимущества интеграции вызовут к жизни широкое распространение электроники и послужат проникновению этой науки во многие новые области.

Интегральные схемы породят такие чудеса, как домашние компьютеры – или, по крайней мере, терминалы, соединенные с центральным компьютером, – автоматические средства управления автомобилями и персональное переносное оборудование связи. Чтобы уже сегодня в нашу жизнь вошли электронные наручные часы, не хватает только дисплея.

Но самый большой потенциал открывается в производстве крупных систем. В телефонной связи применение интегральных схем в цифровых фильтрах позволит разделять каналы в аппаратуре мультиплексирования. Интегральные схемы будут также переключать телефонные цепи и выполнять обработку данных.

Компьютеры станут более мощными и будут совершенно иначе организованы. Например, память, состоящая из интегральных компонентов, может быть распределена по всей машине, а не сосредоточена в центральном блоке. Кроме того, благодаря повышенной надежности интегральных схем можно будет конструировать более крупные обрабатывающие блоки. Машины, аналогичные сегодняшним, будут стоит дешевле и окупаться быстрее.

25.1. Настоящее и будущее

Под интегральной электроникой я понимаю все те разнообразные технологии, которые сегодня называются микроэлектроникой, а также те, результатом которых является функциональность, поставляемая пользователю

в виде неразборных блоков. Исследование этих технологий началось в конце 1950-х годов. Ставилась цель миниатюризировать электронное оборудование, чтобы разместить все более сложные функции в ограниченном пространстве и с минимальным весом. Было предложено несколько подходов, в т. ч. технология микромодулей для отдельных компонентов, тонкопленочные структуры и полупроводниковые интегральные схемы.

Все подходы быстро развивались и сближались, заимствуя идеи и методы друг у друга. Многие исследователи полагают, что будущее – за сочетанием различных подходов. Сторонники полупроводниковых интегральных схем уже вовсю используют улучшенные характеристики тонкопленочных резисторов, нанося пленки непосредственно на активную полупроводниковую подложку. Приверженцы тонкопленочных технологий разрабатывают хитроумные методы размещения активных полупроводниковых приборов на пассивных тонкопленочных решетках.

Оба подхода хорошо зарекомендовали себя и используются в оборудовании уже сегодня.

25.2. Утверждение

Сегодня интегральная электроника заняла свое законное место. Ее применение стало почти обязательным в новых системах военного назначения, поскольку предъявляемые многими из них требования к надежности, размеру и весу достижимы только с помощью интеграции. Такие программы, как Аполлон, программа пилотируемого полета на Луну, продемонстрировали надежность интегральной электроники, показав, что целые схемы безотказны так же, как лучшие отдельные транзисторы.

Большинство компаний в области производства коммерческих компьютеров проектируют или уже находятся на ранних стадиях производства машин на базе интегральных схем. Эти машины дешевле «традиционных» электронных компьютеров, а их производительность выше.

В разнообразных приборах, особенно в быстро растущей доле тех, где применяются цифровые методы, начинают использовать интеграцию, потому что это позволяет уменьшить стоимость производства и проектирования.

Применение аналоговых интегральных схем пока еще ограничено преимущественно военными целями. Такие интегральные функции дороги, а их ассортимент недостаточен для того, чтобы заместить значительную долю аналоговой электроники. Но уже есть первые применения в коммерческом секторе, особенно в оборудовании, где необходимы малоразмерные усилители низкой частоты.

25.3. Надежность важна

Почти во всех случаях интегральная электроника продемонстрировала высокую надежность. Даже при текущем уровне производства – низком по

сравнению с дискретными компонентами – она открывает возможность для снижения стоимости систем, а во многих системах удалось также достичь повышения производительности.

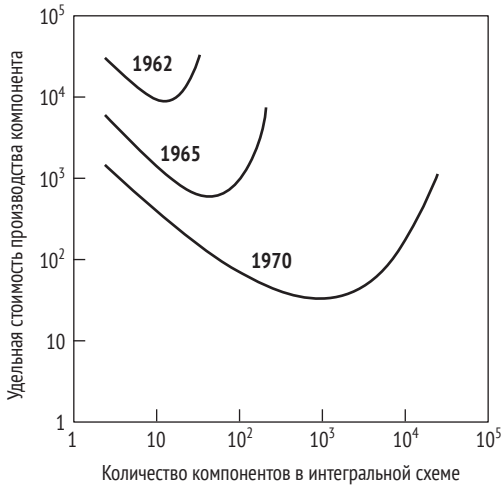
Интегральная электроника будет способствовать распространению электронных технологий в обществе, беря на себя многие функции, которые в настоящее время выполняются плохо или не выполняются вовсе. Главные ее преимущества – низкая цена и существенно более простой дизайн – станут воздаянием за массовое производство дешевых функциональных модулей.

В большинстве применений полупроводниковые интегральные схемы будут преобладать. Полупроводниковые приборы – единственно приемлемые из имеющихся в настоящее время кандидатов на роль активных элементов интегральных схем. Пассивные полупроводниковые элементы тоже выглядят привлекательно благодаря потенциально низкой цене и высокой надежности, но их можно использовать, только если точность не является первоочередным требованием.

Кремний, по-видимому, останется основным материалом, хотя в отдельных приложениях будут использоваться и другие. Например, арсенид галлия будет важен в интегральных схемах, работающих в условиях СВЧ-излучения. Но кремний будет преобладать на низких частотах, потому что уже созданы технологии на его основе и на основе его двуокиси и потому что он встречается в изобилии и относительно дешев в качестве исходного материала.

25.4. Затраты и кривые

Снижение стоимости – одно из основных достоинств интегральной электроники, и его значимость будет только возрастать по мере развития технологии в направлении производства все более функциональных схем на единой полупроводниковой подложке. Для простых схем удельные затраты приближенно обратно пропорциональны числу компонентов – результат эквивалентности количества полупроводника в эквивалентном пакете, содержащем больше компонентов. Но по мере добавления компонентов уменьшение выхода годных более чем компенсирует увеличение сложности, что приводит к росту удельных затрат. Таким образом, на каждом этапе развития технологии существует минимальная стоимость. В настоящее время она достигается, когда схема включает 50 компонентов. Но этот минимум быстро растет, тогда как вся кривая затрат убывает (см. график). Если заглянуть на пять лет в будущее, то кривая затрат позволяет предположить, что минимальной удельной стоимости компонента можно ожидать при 1000 компонентов в схеме (при условии что такие схемы можно будет производить в умеренных количествах). Ожидается, что в 1970 году стоимость производства одного компонента будет в десять раз ниже, чем сейчас.



Сложность при минимальной стоимости компонентов возрастала примерно вдвое за год (см. график). Конечно, можно ожидать, что на протяжении короткого промежутка времени такой темп сохранится, а то и возрастет. В более длительной перспективе уверенность в сохранении темпа возрастания сложности меньше, но нет причин полагать, что она не сохранится примерно на том же уровне в течение, по крайней мере, десяти лет. Это означает, что к 1975 году число компонентов на интегральной схеме будет равно 65 000. Я думаю, что такую большую схему можно будет создать на одной пластине.

25.5. Двухмиллидюймовые квадраты

При тех допусках по размерам, которые уже применяются при производстве интегральных схем, изолированные высокопроизводительные транзисторы можно размещать, если расстояние между центрами равно двум тысячным дюйма. Такой двухмиллидюймовый квадрат может также содержать резистор на несколько килоом или несколько диодов. Это позволяет разместить как минимум 500 компонентов на одном линейном дюйме или четверть миллиона в квадрате со стороной один дюйм. Таким образом, для размещения 65 000 компонентов потребуется всего четверть квадратного дюйма.

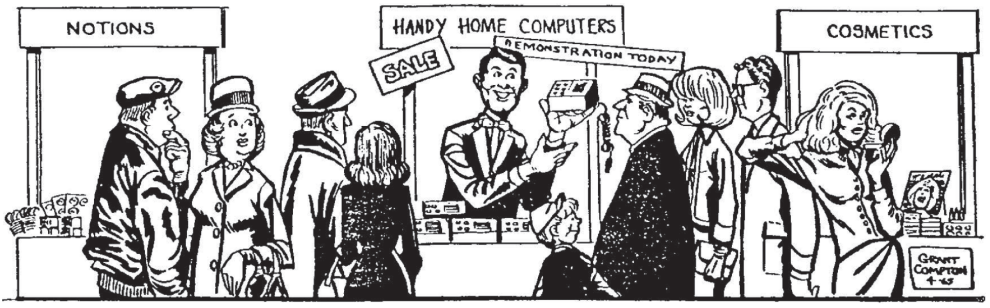
На современных кремниевых пластинах диаметром обычно дюйм или больше достаточно места для такой структуры, если компоненты удастся упаковать плотно, не тратя места на межсоединения. Это реально, поскольку уже полным ходом идут исследования с целью увеличить достигнутый уровень сложности за счет использования многослойных рисунков металлизации со слоями, разделенными диэлектрическими пленками. Такой плотности компонентов можно достичь применением уже имеющихся оптических методов,

не прибегая к более экзотическим технологиям, например с использованием электронного луча, хотя и они изучаются для создания структур еще меньшего размера.

25.6. Увеличение выхода годных

Не существует принципиальных препятствий для доведения выхода годных до 100 %. В настоящее время затраты на упаковку настолько превышают затраты на саму полупроводниковую структуру, что нет никакого стимула увеличивать выход годных, но он может быть поднят в той мере, в какой это экономически оправдано. Не существует барьеров, сопоставимых с соображениями термодинамического равновесия, которые часто ограничивают выход химической реакции; не нужно даже проводить фундаментальные исследования или заменять существующие процессы. Необходимы только инженерные усилия.

На заре развития интегральной электроники, когда выход годных был крайне низок, такие стимулы существовали. Сегодня при изготовлении обыкновенных интегральных схем выход годных сравним с уровнем, характерным для отдельных полупроводниковых приборов. Точно так же можно сделать изготовление бóльших матриц экономичным, если это окажется желательным по другим причинам.



25.7. Проблема тепловыделения

Можно ли будет отводить тепло, выделяемое десятками тысяч компонентов на одном кремниевом кристалле?

Если бы мы смогли уменьшить объем стандартного быстродействующего компьютера до объема, занимаемого самими компонентами, то, наверное, он ярко пылал бы при текущем уровне рассеяния энергии. Но с интегральными схемами такого не произойдет. Поскольку интегральные электронные структуры двухмерные, близко к каждому центру тепловыделения на них имеется поверхность, доступная для охлаждения. Кроме того, энергия необходима в основном для приведения в действие различных линий и ем-

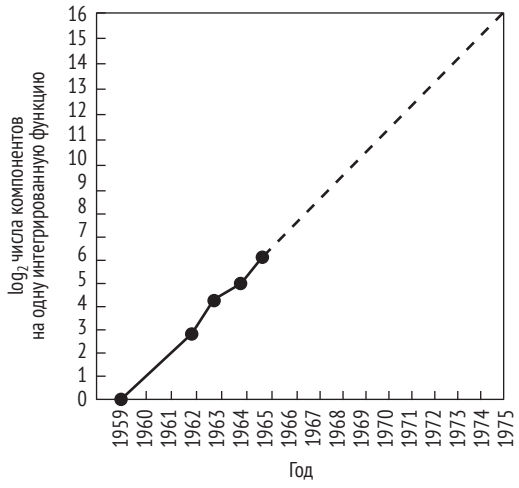
костей в системе. При условии что функция ограничена небольшой площадью на пластине, величина емкости, которая должна быть задействована, четко ограничена. [Примечание редактора: тепловыделение стало проблемой, потому что это предположение перестало быть верным.] На самом деле с уменьшением размеров на интегральной структуре становится возможно эксплуатировать структуру на более высокой скорости при том же тепловыделении на единицу площади.

25.8. Судный день

Очевидно, мы сможем построить такое оборудование, плотно набитое компонентами. Но дальше нужно задать себе вопрос, при каких условиях нам следует это делать. Общие затраты на то, чтобы сделать конкретную систему работоспособной, должны быть минимизированы. Для этого мы должны амортизировать расходы на проектирование и изготовление на несколько идентичных экземпляров или развить гибкие методы разработки сложных устройств, чтобы не нести непропорционально больших затрат на создание отдельной матрицы. Быть может, новые процедуры автоматизации проектирования позволят перевести логическую диаграмму в техническую реализацию без каких-то специальных инженерных усилий.

Быть может, экономически выгоднее будет создавать крупные системы из меньших функциональных блоков, которые упаковываются по отдельности и соединяются между собой. Доступность крупных модулей в сочетании с проектированием и конструированием функционала должна дать производителю больших систем возможность проектировать и конструировать самое разное оборудование быстро и экономично.

25.9. Аналоговые схемы



Интеграция не изменит аналоговые системы так же радикально, как цифровые. И все же при изготовлении аналоговых схем будет достигнута значительная степень интеграции. Отсутствие конденсаторов и индуктивностей большого номинала – самое серьезное препятствие на пути внедрения интегральной электроники в аналоговые схемы.

По самой своей природе такие элементы требуют запасания энергии в некотором объеме. Для достижения высокого Q необходимо, чтобы объем был большим. Несовместимость большого объема с интегральной электроникой очевидна из самого смысла терминов. Можно ожидать, что определенные резонансные явления, такие, например, как в пьезоэлектрических кристаллах, найдут применения к функциям настройки, но от емкостей и конденсаторов нам какое-то время никуда не деться.

Будущий интегральный усилитель высокой частоты вполне мог бы состоять из интегральных каскадов усиления, обладающих высокими характеристиками при минимальной стоимости, перемежающихся относительно большими элементами настройки.

Другие аналоговые функции претерпят значительные изменения. Согласование и отслеживание похожих компонентов в интегральных структурах позволит проектировать дифференциальные усилители со значительно улучшенными характеристиками. Применение тепловой обратной связи для стабилизации интегральных структур с точностью до доли градуса позволит конструировать осцилляторы с устойчивостью кристаллической структуры.

Даже в сверхвысокочастотной области структуры, включенные в определение интегральной электроники, будут приобретать все большую значимость. Способность изготавливать и собирать компоненты, размер которых мал по сравнению с длиной волны, позволит использовать конструкции с сосредоточенными параметрами, по крайней мере на низких частотах. В настоящее время трудно предсказать, насколько интенсивным будет проникновение интегральной электроники в СВЧ-область. Успешная реализация таких устройств, как, например, антенны с фазированной решеткой, в которых используется множество интегрированных источников СВЧ-энергии, могло бы совершить революцию в области радиолокаторов.

26 Решение задачи параллельного управления программой (1965)

Эдсгер Дейкстра

Эдсгер Дейкстра (1930–2002) был блестящей и неоднозначной фигурой. Даже завершение предложения «Дейкстра был голландским...» вызывает трудности. Он был убийственно язвительным мыслителем и писал первоклассные статьи о компьютерном программировании. Но он не любил выражения «компьютерная наука» (computer science), предпочитая ему «наука вычислений» (computing science) в тех случаях, когда вообще пользовался каким-нибудь термином. Он отвергал разветвление вычислений на теорию и инженерную практику; он настаивал на элегантности и ясности всего, к чему прикасался. «Искусство программирования – это искусство организации сложности, овладения разнообразием и избегания сопровождающего его хаоса настолько эффективно, насколько это возможно», – писал он (Dijkstra 1972). И если для этого требовалось больше мозгов, что ж, не каждый рожден для такой работы. «Не я виноват в том, что компетентное программирование, которое я вижу как интеллектуальную способность, оказывается слишком трудным для "среднего программиста" – не следует совершать ошибку, отказываясь от какой-то хирургической техники, потому что она не по зубам цирюльнику в парикмахерской за углом» (Dijkstra 1975).

На протяжении всей своей карьеры Дейкстра пытался переместить центр внимания с машины, исполняющей программы, на абстрактные рассуждения, стоящие за вычислениями. Получив математическое и физическое

образование, он призывал методично обдумывать проблемы программирования и осмыслять их математически, прежде чем писать первую строку кода. Он внес важный вклад во многие аспекты этой области, сначала как программист-практик в компьютерную индустрию, а затем как профессор Технологического университета Эйндховена, Нидерланды, и Техасского университета в Остине. С его именем связан элегантный алгоритм нахождения кратчайших путей между всеми парами вершин графа (Dijkstra 1959). Он также внес важный вклад в изучение компиляторов, операционных систем (см. главу 28), распределенных систем и методов программирования (см. главу 29).

Эта работа, наверное, является первой научной статьей по алгоритмическим проблемам – в отличие от языковых конструкций – в области конкурентного программирования (встречаются также названия «параллельное программирование» и «мультипрограммирование»). Эта статья – образец логической строгости, хотя код отнюдь не назовешь самодокументированным. Она положила начало целой новой области исследований – конкурентным вычислениям. Операционные системы и системы баз данных в особенности обладают тем свойством, что конкурентные участки кода выполняются очень много раз при самых различных условиях, и потому любая ошибка, которая может случиться, рано или поздно случится. Дейкстра в этой работе указывает путь к использованию ясных рассуждений для предотвращения ошибок синхронизации. Он вводит термины «критическая секция» и «взаимное исключение» и доказывает, что его код удовлетворяет требованиям, которые сегодня мы назвали бы безопасностью и живучестью (Lamport 2015).

Ратуя за строгое мышление, Дейкстра с такой же категоричностью отвергал работы, не отвечавшие его стандартам, – включая целые исследовательские программы. Невозможно в полной мере оценить его непререкаемое требование логической ясности, не поняв его отношения к работе, этому требованию не удовлетворявшей. Когда в 1985 году его спросили о будущем искусственного интеллекта, он ответил: «Можно ли исследовать что-то, наукой не являющееся? Я считаю, что пытаться использовать машины для подражания человеческому способу рассуждения глупо и опасно. Глупо, потому что если взглянуть на человеческий способ рассуждения без прикрас, он покажется довольно скверным; даже самые квалифицированные математики – мыслители так себе. <…> Любой успешный проект ИИ по самой своей природе станет кастрацией машины». Когда его спросили об огромном росте интереса к компьютерной науке со стороны студентов, он ответил: «Если вы спрашиваете меня, много студентов или мало, отвечу: *на порядок больше, чем нужно*. С точки зрения науки, надо бы сильно проредить. Оставьте 2 % самых талантливых, и пусть они занимаются делом» (van Vlissingen and Dijkstra, 1985).

О важности делать программы более удобными для использования: «Пользователь компьютера, которого мысленно представляют себе при разработке компьютерных продуктов, – не реальный человек из плоти и крови, а литературный персонаж <…>. Он глуп, противится образованию, если вообще способен к образованию, и ненавидит, когда от него требуют интеллектуальных усилий в любой форме, он не способен восхищаться прекрасным, поскольку

не обучен ценить красоту. Целые разделы компьютерной науки парализованы видением такого кретина в роли своего типичного пользователя».

26.1. Введение

В этой статье приводится решение проблемы, вопрос о разрешимости которой, насколько известно автору, оставался открытым по крайней мере с 1962 года. Статья состоит из трех частей: проблема, решение и доказательство. Хотя на первый взгляд постановка задачи может показаться академической, автор уверен, что всякий, кто знаком с логическими проблемами, возникающими в связи со взаимодействием компьютеров, оценит значимость того факта, что эту проблему все-таки можно решить.

26.2. Проблема

Начнем с того, что рассмотрим *N* компьютеров, и пусть каждый из них выполняет процесс, который для наших целей можно считать циклическим. В каждом цикле имеется так называемая «критическая секция», и компьютеры следует запрограммировать так, чтобы в любой момент времени в критической секции находился только один из этих *N* циклических процессов. Чтобы добиться такого взаимного исключения выполнения в критической секции, компьютеры могут взаимодействовать друг с другом посредством общей памяти. Запись слова в эту память или неразрушающее чтение слова из нее – неделимые операции, т. е. когда два или более компьютеров пытаются одновременно взаимодействовать (читать или записывать) с одной и той же ячейкой общей памяти, взаимодействия будут происходить поочередно, но в неизвестном порядке.

Решение должно удовлетворять следующим требованиям.

(a) Решение должно быть симметрично относительно всех *N* компьютеров; следовательно, мы не вправе вводить статические приоритеты.
(b) Не делается никаких предположений об относительном быстродействии *N* компьютеров; мы даже не вправе предполагать, что скорость их работы не изменяется во времени.
(c) Если какой-то компьютер остановится вне своей критической области, это не должно приводить к блокировке остальных.
(d) Если более одного компьютера собираются войти в критическую область, должно быть невозможно задать для них такие конечные скорости, при которых решение о том, какой компьютер войдет в свою критическую область первым, будет отложено на бесконечное время. Иными словами, конструкции, в которых последовательность «После вас»–«После вас»–блокировка возможна, хотя и маловероятна, не должны считаться правильными решениями.

Мы призываем амбициозного читателя сделать здесь паузу и попытаться решить задачу самостоятельно, потому что это представляется единственным способом в полной мере прочувствовать каверзные следствия из того факта, что каждый компьютер может запрашивать лишь одно одностороннее сообщение в каждый момент времени. И только это заставит читателя осознать, до какой степени эта проблема далека от тривиальной.

26.3. Решение

Общая память состоит из: «**boolean array** *b*; *c*[1 : *N*]; **integer** *k*».

Целое *k* удовлетворяет условию $1 \leqslant k \leqslant N$, *b*[*i*] и *c*[*i*] записываются только *i*-м компьютером, но могут быть прочитаны всеми остальными. Предполагается, что все компьютеры начинают работу далеко от критической области, а все вышеупомянутые булевы массивы инициализированы значением **true**; начальное значение *k* несущественно.

Программа для *i*-го компьютера ($1 \leqslant i \leqslant N$) имеет вид:

```
“integer j;
Li0 : b[i] := false;
Li1 : if k ≠ i then
Li2 :     begin
              c[i] := true;
Li3 :         if b[k] then k := i;
              go to Li1
          end
      else
Li4 :     begin
              c[i] := false;
              for j := 1 step 1 until N do
                  if j ≠ i and not c[j] then go to Li1
          end;
      критическая секция;
      c[i] := true; b[i] := true;
      остальная часть цикла, в которой разрешена остановка;
      go to Li0»
```

26.4. Доказательство

Для начала заметим, что решение безопасно в том смысле, что никакие два компьютера не могут оказаться в своей критической области одновременно. Ибо единственный способ войти в критическую секцию – выполнить составное предложение *Li*4 без обратного перехода к *Li*1, т. е. когда все прочие *c* оказались равными **true** после того, как собственное *c* сделано равным **false**.

Во второй части доказательства мы должны показать, что не может случиться последовательности «После вас»–«После вас»–блокировка, т. е. когда ни один компьютер не находится в своей критической секции, из циклящихся компьютеров (т. е. тех, которые переходят обратно на метку *Li*1) по крайней мере одному – а значит, ровно одному – будет разрешено войти в его критическую секцию в должное время.

Если *k*-й компьютер не входит в число циклящихся, то *b*[*k*] будет равно **true** и все циклящиеся компьютеры найдут, что $k \neq i$. В результате один или несколько из них найдут, что в точке *Li*3 булева величина *b*[*k*] равна **true**, и, следовательно, один или более решат выполнить присваивание «*k* := *i*». После первого присваивания «*k* := *i*» величина *b*[*k*] становится равной **false**, и ни один новый компьютер не может еще раз решить присвоить *k* новое значение. После того как все присваивания *k* произведены, *k* будет указывать на один из циклящихся компьютеров, и его значение некоторое время не будет изменяться, а именно до тех пор, пока *b*[*k*] не станет равным **true**, т. е. пока *k*-й компьютер не завершит работу в своей критической секции. Как только значение *k* перестанет изменяться, *k*-й компьютер перейдет в состояние ожидания (с помощью составного предложения *Li*4), пока все остальные *c* не станут равны **true**, но это обязательно произойдет, если еще не произошло, потому что все остальные циклящиеся компьютеры обязаны установить свои *c* в **true**, когда обнаруживают, что $k \neq i$. И это, как полагает автор, завершает доказательство.

27 Элиза – компьютерная программа для изучения взаимодействия между человеком и машиной на естественном языке (1966)

Джозеф Вейценбаум

Джозеф Вейценбаум (1923–2008) – немецкий еврей, бежавший вместе с семьей в США, когда ему было 13 лет. После изучения математики и вычислительной техники в государственном университете Уэйна он начал работать преподавателем на факультете компьютерных наук в МТИ. Там в 1964 году он написал первый пример того, что сегодня мы назвали бы чат-ботом, – программы, которая «знает» очень мало, но создает иллюзию разговора, манипулируя фразами собеседников. Он был удивлен, что люди так увлекались диалогом с этой простой программой, как будто это человек. Известен курьезный факт – его собственная ассистентка, лучше кого-либо другого знавшая, что никакого человека, отвечающего на ее смутные мечтания, нет и в помине, просила его выходить из комнаты, когда беседовала с Элизой

(Weizenbaum, 1976, стр. 6). Как будто думала, что он подслушивает ее личные разговоры.

Элиза стала сенсацией отчасти и потому, что в 1966 году разделение времени было в новинку – настолько, что в этой статье Вейценбаум счел необходимым объяснить, что это такое, читателям журнала «Communications of the ACM». Впервые люди без технического образования начинали пользоваться компьютерами, а программисты – писать программы, предназначенные не в последнюю очередь для того, чтобы поразвлечь обыкновенных людей.

Но Элиза также затронула что-то, спрятанное глубоко в душе человека. Программа названа в честь Элизы Дулитл, персонажа пьесы Бернарда Шоу «Пигмалион», по которой в 1956 году на Бродвее был поставлен мюзикл «Моя прекрасная леди». В 1964 году вышел фильм по мотивам мюзикла. В пьесе Шоу Элиза была неграмотной лондонской продавщицей цветов, а профессор Генри Хиггинс, лингвист, «перепрограммировал» ее, научив изображать из себя аристократку. Хиггинс влюбился в (почти) полностью преобразившуюся Элизу, так же, как в греческом мифе Пигмалион влюбился в статую женщины, изваянную им из слоновой кости.

На самом деле греческий миф о Пигмалионе ближе к Элизе, чем современная пьеса, поскольку повествует об оживлении неживого. В оригинале боги ответили на молитвы Пигмалиона и вдохнули душу в его статую. Это лишь один из западных мифов о неодушевленном предмете, вызванном к жизни в образе человека (см. стр. 20).

В детстве став свидетелем расчеловечивания человеческих существ, Вейценбаум был глубоко обеспокоен тем, как легко люди обманываются, и скептически относился к научной программе своих коллег очеловечить машины. Всю свою жизнь он резко критиковал искусственный интеллект; его самая важная работа, в которой он сделал попытку раз и навсегда разделить человека и машины, называется «Computer Power and Human Reason» (Weizenbaum, 1976)[1]. Она не убедила сторонников ИИ, а по мере отработки технологий понимания и синтеза речи и моделирования эмоций продолжились споры о том, где компьютеры *должны* – а не *могут* – заменить человека.

Элиза – программа, работающая под управлением системы с разделением времени MAC в MTИ, которая открывает возможность для разнообразных бесед человека с компьютером на естественном языке. Входные предложения анализируются на основе правил разбора, активируемых появлением ключевого слова во входном тексте. Для генерирования ответов применяются правила сборки, ассоциированные с соответствующими правилами разбора. Основные технические проблемы, с которыми справляется Элиза, – это: (1) выявление ключевых слов, (2) выделение минимального контекста, (3) выбор подходящих преобразований, (4) генерирование ответов в отсутствие ключевых слов и (5) обеспечение возможности редактирования «сценариев» Элизы. В конце этой статьи обсуждаются некоторые психологи-

[1] *Вейценбаум Дж.* Возможности вычислительных машин и человеческий разум / пер. с англ.; под ред. А. Л. Горелика. М.: Радио и связь, 1982.

ческие вопросы, относящиеся к принятому в Элизе подходу, а также будущее развитие.

27.1. Введение

Говорят, что объяснять – значит оправдываться. Эта максима нигде не уместна так, как в области компьютерного программирования, особенно так называемого эвристического программирования и искусственного интеллекта. Ибо в этих сферах программисты принуждают машины вести себя диковинным образом, зачастую настолько, что даже самый опытный наблюдатель может впасть в заблуждение. Но как только с программы сброшена маска, как только механизм ее работы объяснен на языке, достаточно простом для понимания, магия трещит по швам; остается простой набор процедур, каждая из которых не таит в себе ничего непостижимого. Наблюдатель говорит себе «я бы и сам мог написать такое». И с этой мыслью переставляет программу с полки, помеченной «интеллектуальные», на полку для курьезов, годящихся лишь для обсуждения с людьми, менее просвещенными, чем он сам.

Цель этой статьи как раз и состоит в подобной переоценке программы, которую мы собираемся «объяснить». Мало найдется программ, нуждающихся в этом сильнее.

27.2. Программа Элиза

Элиза – программа, делающая возможным диалог с компьютером на естественном языке. Текущая реализация работает в системе с разделением времени MAC в МТИ. Она написана на языке MAD-SLIP (Weizenbaum, 1963) для IBM 7094. Название выбрано специально, чтобы подчеркнуть, что она постепенно улучшается в ходе общения с пользователями, потому что ее языковые способности могут быть развиты «учителем». Как и Элизу из пьесы «Пигмалион», ее можно заставить выглядеть более цивилизованной, однако соответствие внешнего вида и реальности остается на совести драматурга.

Для наших целей MAC достаточно охарактеризовать как систему, которая позволяет человеку пользоваться всеми возможностями компьютера, работая с удаленной пишущей машинки. У оператора создается иллюзия, будто он единственный пользователь вычислительного комплекса, хотя на самом деле вместе с ним могут «разделять время» и другие. Важно здесь только то, что компьютер может читать сообщения, набранные на пишущей машинке, и отвечать, пользуясь тем же устройством. Время между получением сообщения компьютером и появлением ответа зависит от программы, управляющей диалогом, и таких параметров системы MAC, как количество пользователей, общающихся в данный момент с системой. Впрочем, эти параметры обычно вносят такой малый вклад в общее время ответа, что в ходе диалога с компьютером никогда возникает совсем уж нестерпимых задержек.

При беседе с Элизой пользователь вводит какое-нибудь предложение или несколько предложений на естественном языке, соблюдая обычную пунктуацию и грамматическую структуру. Нельзя только использовать вопросительный знак, потому что он интерпретируется системой MAC как символ удаления строки. Предложение пользователя завершается двойным возвратом каретки, после чего управление переходит к Элизе. Элиза анализирует предложение пользователя, генерирует ответ и печатает его. Управление возвращается пользователю. Типичный диалог может выглядеть следующим образом[1].

Все мужчины одинаковы.
КАКИМ ОБРАЗОМ
Они всегда достают нас так или иначе.
НЕ МОГЛИ БЫ ПРИВЕСТИ КОНКРЕТНЫЙ ПРИМЕР
Ну вот, мой дружок заставил меня прийти сюда.
ВАШ ДРУЖОК ЗАСТАВИЛ ВАС ПРИЙТИ СЮДА
Он говорит, что я всегда кисну.
МНЕ ГРУСТНО СЛЫШАТЬ, ЧТО ВЫ ВСЕГДА КИСНЕТЕ
Это так. Я несчастна.
ВЫ ДУМАЕТЕ, ЧТО ПРИХОД СЮДА ПОМОЖЕТ ВАМ ПЕРЕСТАТЬ БЫТЬ НЕСЧАСТНОЙ
Мне нужна помощь, уж это точно.
КАКОЕ ЗНАЧЕНИЕ ИМЕЛА БЫ ДЛЯ ВАС ПОМОЩЬ ЕСЛИ БЫ ВЫ ЕЕ ПОЛУЧИЛИ
Быть может, я смогла бы научиться ладить с матерью.
РАССКАЖИТЕ МНЕ ПОДРОБНЕЕ О СВОЕЙ СЕМЬЕ
Моя мать заботится обо мне.
КТО ЕЩЕ В ВАШЕЙ СЕМЬЕ ЗАБОТИТСЯ О ВАС
Мой отец.
ВАШ ОТЕЦ
Вы чем-то напоминаете моего отца.
В ЧЕМ ВЫ ВИДИТЕ СХОДСТВО
Вы не особенно настойчивы, но я думаю, что вы не хотите, чтобы я это заметила.
ЧТО ЗАСТАВЛЯЕТ ВАС ДУМАТЬ ЧТО Я НЕ ОСОБЕННО НАСТОЙЧИВА
Вы со мной не спорите
КАК ВЫ ДУМАЕТЕ ПОЧЕМУ Я С ВАМИ НЕ СПОРЮ
Вы меня боитесь.
ВАМ НРАВИТСЯ СЧИТАТЬ, ЧТО Я ВАС БОЮСЬ
Мой отец всех боится.
ЧТО ЕЩЕ ПРИХОДИТ ВАМ НА УМ, КОГДА ВЫ ДУМАЕТЕ ОБ ОТЦЕ
Агрессия.
ЭТО КАК-ТО СВЯЗАНО С ТЕМ ФАКТОМ, ЧТО ВАШ ДРУЖОК ЗАСТАВИЛ ВАС ПРИЙТИ СЮДА

Заглавными буквами напечатаны ответы машины.

---

[1] В оригинале диалог ведется по-английски, Элиза не понимала русского языка (хотя ее и можно было ему научить). Но эта техническая деталь не помешает восприятию приведенного ниже диалога. – *Прим. перев.*

Главная процедура программы очень проста: текст читается, и в нем ищется ключевое слово. Если такое слово найдено, то предложение преобразуется в соответствии с правилом, ассоциированным с данным словом, если это не бессодержательное замечание. При некоторых условиях извлекается результат предшествующего преобразования. Вычисленный или извлеченный текст печатается.

Конечно, детали описанной выше процедуры существенно сложнее. Например, с ключевыми словами может быть связан РАНГ, или числовой приоритет. Процедура чувствительна к таким числам в том смысле, что отбросит ключевое слово, уже найденное при просмотре текста слева направо, в пользу слова с более высоким рангом. Кроме того, процедура распознает запятые и точки в качестве ограничителей. Если такой знак встретился и ключевое слово уже было найдено, то весь последующий текст удаляется из входного сообщения. Если ключевое слово еще не было найдено, то удаляется фраза или предложение слева от ограничителя (и сам ограничитель). В результате преобразованию всегда подвергается только одна фраза или предложение.

Ключевые слова и ассоциированные с ними правила преобразования составляют СЦЕНАРИЙ для конкретного класса диалогов. Важным свойством Элизы является то, что сценарий – это данные, а не часть самой программы. Поэтому Элиза не ограничена определенным множеством распознаваемых образцов или ответов и даже каким-то одним языком. Существуют сценарии Элизы (на данный момент) не только для английского, но также для валлийского и немецкого языков.

Ниже перечислены фундаментальные технические проблемы, занимающие Элизу.

1. Идентификация «самого важного» ключевого слова во входном сообщении.
2. Идентификация некоторого минимального контекста, окружающего выбранное ключевое слово; например, если ключевым словом является «you», то за ним следует слово «are» (в этом случае, вероятно, высказывается какое-то утверждение).
3. Выбор подходящего правила преобразования и, конечно, применение самого преобразования.
4. Предоставление механизма, позволяющего Элизе отвечать «разумно», когда входной текст не содержит ключевых слов.
5. Предоставление механизма редактирования и особенно расширения сценария на уровне написания сценария.

Конечно, имеются обычные ограничения, диктуемые необходимостью экономно использовать машинное время и память.

Центральным вопросом, очевидно, является обработка текста, а его главной частью – концепция *правила преобразования*, которое, как было сказано, ассоциировано с определенными ключевыми словами. Механизмы, отнесенные к категории «правило преобразования», – это ряд функций на языке SLIP, которые служат (1) для разбора строки данных в соответствии с некоторым критерием и (2) для сборки разобранной строки в соответствии с некоторыми спецификациями. <…>

27.3. Обсуждение

На момент написания этой статьи единственные сколько-нибудь серьезные сценарии Элизы заставляют ее отвечать примерно так, как психотерапевты (последователи Роджерса). Элиза работает оптимально, когда ее собеседника предварительно проинструктировали «разговаривать» с ней (посредством пишущей машинки, конечно) так, как он разговаривал бы с психиатром. Такой режим был выбран, потому что беседа с психиатром – один из немногих примеров категоризованного диадического общения на естественном языке, при котором один из участников волен притворяться, что почти ничего не знает о реальном мире. Если, например, говорящий скажет психиатру «я отправился в долгую прогулку на катере», а тот ответит «расскажите мне о катерах», не следует предполагать, что он ничего не знает о катерах, на самом деле он лишь хочет задать направление последующему разговору. Важно отметить, что это предположение самого говорящего. Реалистично оно или нет – совершенно другой вопрос.

В любом случае его психологическая полезность несомненна, потому что у говорящего поддерживается впечатление, что его слышат и понимают. Далее говорящий отстаивает это впечатление (вполне возможно, иллюзорное), наделяя собеседника любыми знаниями об обстановке, возможностью проникновения вглубь проблемы и способностью рассуждать. Но опять-таки все это – вклад, привнесенный *говорящим* в беседу. Он проявляется дедуктивно в *интерпретации* предлагаемых ответов говорящим. Чисто технически, с точки зрения программирования, сценарий Элизы в форме беседы с психиатром имеет то преимущество, что устраняется необходимость хранить явную информацию о реальном мире.

Человек-собеседник, как уже было сказано, многое делает для того, чтобы облечь ответы Элизы в одежды правдоподобия. В разговоре двух людей говорящий делает определенные (быть может, излишне щедрые) предположения о своем собеседнике. Коль скоро сохраняется возможность интерпретировать ответы последнего, не вступая в противоречие с этими предположениями, представление говорящего о собеседнике остается неизменным и, главное, нескомпрометированным. Ответы, которые с трудом поддаются такой интерпретации, могут привести к расширению образа собеседника, к дополнительным рациональным объяснениям, благодаря которым в дальнейшем становятся разумны более сложные интерпретации ответов.

Если, однако, такие рациональные объяснения становятся слишком громоздкими и даже противоречивыми, то весь образ может рассыпаться и будет заменен другим («А ведь он не такой умный, как я думал»). Если собеседником является машина (различие между машиной и программой здесь несущественно), то идею *достоверности* вполне можно заменить идеей *правдоподобия*.

Взяв Элизу за основу, можно ставить эксперименты, в которых участник находит правдоподобным, что ответы, появляющиеся на его пишущей машинке, исходят от человека, сидящего за таким же инструментом в другой комнате. Как должен быть написан сценарий, чтобы поддерживать доверие

к этой идее на протяжении длительного времени? Как можно систематически понижать характеристики Элизы, чтобы прийти к контролируемым и предсказуемым порогам доверия у участника? Какую роль во всем этом играет первоначальный инструктаж участника? С другой стороны, предположим, что участнику сообщено, что он общается с машиной. К каким выводам относительно машины он придет в результате опыта диалога с ней? Некоторых участников было очень трудно убедить, что Элиза (при использовании имеющегося на данный момент сценария) – не человек. Это поразительная форма теста Тьюринга. Как следовало бы спланировать эксперимент, чтобы сделать его более строгим и не вызывающим сомнений?

Вообще, вопрос доверия (со стороны человека) к выводу машины заслуживает внимательного исследования. Все чаще важные решения принимаются на основе данных, выведенных компьютером. Человек, отвечающий за окончательное толкование того, «что говорит машина», мало чем отличается от собеседника Элизы, который постоянно сталкивается с необходимостью выносить суждения о степени достоверности. Как минимум Элиза показывает, как просто создать и поддерживать иллюзию понимания и сопереживания, а как следствие, и мнение о том, что она заслуживает доверия. Здесь таится определенная опасность.

Представление о том, что текущий сценарий Элизы не содержит никакой информации о реальном мире, не совсем верно. Например, правила преобразования, благодаря которым ввод

Меня все ненавидят

преобразуется в вывод

Не могли бы вы назвать кого-то конкретного

и другие в том же духе основаны на вполне определенных гипотезах о мире. В каком-то смысле весь сценарий является моделью некоторых аспектов мира. Написание сценария – вариант программирования и обладает всеми особенностями программирования, прежде всего ясно показывает точку, где программист перестает владеть предметом.

Значительная часть той элегантности, которую можно приписать Элизе, связана с тем фактом, что Элиза поддерживает иллюзию понимания при минимуме механизмов. Но есть границы расширяемости «понимательной» способности Элизы, и это функция самой программы, а не поданного ей сценария. Любой учитель знает, что верным критерием понимания является не способность испытуемого продолжать разговор, а способность делать правильные выводы из услышанного. Чтобы компьютерная программа могла это сделать, она должна по крайней мере иметь возможность сохранять избранные части входных данных. Элиза отбрасывает все входные данные, кроме тех немногих, что были преобразованы посредством механизма ПАМЯТИ. [Примечание редактора: небольшое количество входных предложений сохраняется в структуре данных ПАМЯТЬ, чтобы можно было вернуться к тому, что пользователь упоминал ранее, когда диалог затухает.] Конечно, проблема не только в недостатке памяти. На самом деле суть ее выражается употребленным выше словом «избранные». В своей текущей ипостаси Элиза считает

одной из главных целей скрыть отсутствие понимания. Но чтобы побудить собеседника предлагать входные данные, из которых она могла бы выбрать информацию, способствующую «излечению», она должна обнаружить свое непонимание. Переключение между двумя целями – сокрытием и обнаружением непонимания – по-видимому, является необходимым условием, при котором программа типа Элизы может лечь в основу эффективной системы общения между человеком и машиной на естественном языке.

Таким образом, одной из целей улучшенной программы Элиза является создание системы, которая уже имеет доступ к хранилищу информации о некоторых аспектах реального мира и которая посредством диалога с людьми может обнаружить как то, что ей известно (т. е. вести себя как поисковая система), так и то место, где ее знания заканчиваются и должны быть пополнены. Хочется надеяться, что пополнение ее знаний также будет прямым следствием опыта, приобретенного в ходе бесед. Именно перспектива бесед такой программы со многими людьми, в ходе которых она учится у каждого чему-то новому, и дает надежду на то, что в итоге она может стать интересным и даже полезным собеседником.

Немного иную промежуточную цель можно сформулировать, сказав, что Элиза должна быть наделена способностью постепенно строить модель своего собеседника. Например, если собеседник упомянул, что он не женат, а позже начинает говорить о своей жене, то Элиза должна уметь сделать осторожный вывод, что он либо вдовец, либо разведен. Конечно, он мог просто что-то перепутать. В итоге Элиза должна уметь строить структуру доверия (пользуясь терминологией Абельсона) к собеседнику и на этой основе обнаруживать в его словах рациональные объяснения, противоречия и т. д. Диалоги с такой Элизой часто заканчивались бы спорами. Важные шаги на пути осуществления этих целей уже сделаны. Самый заметный из них – работа Абельсона и Кэрролла по моделированию структур доверия (Abelson and Carroll 1965).

Сценарий, положенный в основу большей части этого обсуждения, имеет выраженную психологическую ориентацию. Причина уже была названа. Но есть опасность, что этот пример разойдется с тем, что был призван иллюстрировать. Полезно помнить, что программа Элиза – не более чем транслятор в том техническом смысле, который принят в программировании. Горн (Gorn 1964) в работе по языковым системам пишет:

> Если дан язык, который уже обладает семантическим содержанием, то транслятор, даже если он понимает только синтаксис, порождает соответствующие выражения на другом языке, которым мы можем приписать «смыслы» (быть может, несколько – транслятор может и не быть взаимно однозначным), «семантические намерения» исходных выражений; сочтем ли мы результат непротиворечивым, полезным или тем и другим вместе, – это, конечно, совсем другой вопрос. Вполне возможно, что таким методом на синтаксически одном и том же целевом языке можно приписать несколько полезных смыслов каждому выражению. <…>

Поразительно, как точно его слова описывают Элизу. «Заданный язык» – английский, равно как и «другой» язык, на котором порождаются выражения.

В принципе, заданным языком мог бы быть тот вариант английского, на котором формулируются школьные задачи по алгебре, а другим языком – машинный код, позволяющий конкретному компьютеру «решать» поставленные задачи. (См. программу Боброу STUDENT [Bobrow 1964].)

Цель предшествующих замечаний – сорвать с Элизы остатки покрова волшебства, который был наброшен на нее в связи с применением, носящим оттенок психологии. Если посмотреть холодным взглядом, то Элиза – не более чем транслятор в смысле Горна, но специально сконструированный в расчете на хорошую работу с текстом на естественном языке.

28 Структура системы мультипрограммирования THE (1968)

Эдсгер Дейкстра

Сведение об авторе приведены в главе 26, где было продемонстрировано мастерство Дейкстры в математическом и логическом осмыслении вычислений. Но Дейкстра был также инженером, очень опытным системным проектировщиком и разработчиком. Этот успешный в плане дизайна и разработки проект упрочил репутацию Дейкстры как специалиста по информатике в полном смысле слова; он послужил стимулом для огромного числа научных работ по доказуемой правильности, безопасности и надежности операционных систем.

В приложении представлены семафоры, весьма важная абстракция управления, заключенная в форму, удобную для общих рассуждений о разнообразных машинно зависимых методах, применявшихся в то время для блокировки и разблокировки ресурсов в конкурентных программах.

28.1. Введение

В ответ на недвусмысленный призыв предлагать статьи по «актуальным исследованиям и разработкам» я представляю отчет о состоянии работ по мультипрограммированию на факультете математики в Технологическом университете Эйндховена, Нидерланды.

Располагая очень ограниченными ресурсами (а именно группой из шести человек, работающих в среднем по полдня) и желая внести вклад в искусство

проектирования систем – включая все стадии планирования, конструирования и верификации, – мы столкнулись с проблемой, как получить необходимый опыт. Для решения этой проблемы мы приняли три следующих руководящих принципа:

1. Выбирать проект настолько передовой, насколько можно себе представить, и настолько амбициозный, насколько можно это обосновать, в надежде, что рутинная работа будет сведена к минимуму; противиться всякому побуждению включить в систему такие расширения, которые приведут лишь к чисто количественному увеличению общего объема работы.
2. Выбирать машину с хорошими базовыми характеристиками (например, системой прерываний, чтобы всей душой полюбить ее поистине чудесные свойства); в дальнейшем стараться как можно дольше не включать в рассмотрение те конкретные свойства конфигурации, для которой создается система.
3. Осознавать тот факт, что опыт ни в коем случае не ведет автоматически к знанию и пониманию; другими словами, предпринимать сознательные усилия, для того чтобы извлечь как можно больше из своего предшествующего опыта.

Таким образом, я попытаюсь не ограничиваться тем, что и как мы сделали, а рассказать еще и о том, чему мы научились.

Мне хочется закончить введение двумя краткими замечаниями об условиях работы, просто для полноты картины. Далее я не буду акцентировать внимание на этих моментах.

Первое замечание состоит в том, что скорость разработки заметно снижается, если люди работают по полдня и имеют также и другие обязанности. Коэффициент снижения составляет, как минимум, 4, а может быть, и того больше. Каждый человек в отдельности теряет время и энергию на смену вида деятельности, а группа в целом теряет темп принятия решений, поскольку, когда возникает необходимость в обсуждении, его часто приходится откладывать до тех пор, пока все необходимые люди не соберутся вместе.

Другое замечание заключается в том, что члены группы (в основном математики) раньше были хорошими студентами университета, проучившимися от 5 до 8 лет, и имели степень магистра или доктора. Я отмечаю это потому, что, по крайней мере в моей стране, интеллектуальный уровень, требуемый для проектирования систем, обычно сильно недооценивается. Я более чем когда-либо убежден, что это очень трудная работа и что любые попытки привлекать к ней не самых лучших обречены либо на провал, либо на скромный успех при очень высоких затратах.

28.2. Инструментарий и цель

Система разрабатывалась для голландской машины EL X8 (N.V. Electrologica, Rijswijk (ZH)). Характеристики конфигурации следующие:

1. Оперативная память: время цикла 2.5 мкс, 27 бит, 32K в настоящее время.
2. Магнитный барабан на 512K слов, 1024 слова на дорожке, время оборота 40 мс.
3. Механизм косвенной адресации, хорошо приспособленный к реализации стека.
4. Хорошая система управляемой периферии и управления прерываниями.
5. Потенциально большое число каналов с низкой пропускной способностью, десять из которых используются (3 считывателя с перфоленты со скоростью 1000 символов/с, 3 ленточных перфоратора со скоростью 150 символов/с, 2 телетайпа, графопостроитель и строчный принтер).
6. Отсутствие многих громоздких, но часто встречающихся возможностей.

Основной целью системы является бесперебойная обработка непрерывного потока пользовательских программ для нужд университета. Мультипрограммная система была выбрана, имея в виду следующие цели: (1) уменьшение оборотного времени для коротких программ; (2) экономное использование периферийных устройств; (3) автоматическое управление вспомогательной памятью в сочетании с экономным использованием центрального процессора; (4) экономическая целесообразность использования машины для тех приложений, которым нужна только гибкость ЭВМ общего назначения, но (как правило) не нужна ни высокая пропускная способность, ни вычислительная мощность.

Система не задумывалась как система коллективного доступа. В ней нет общей базы данных, с помощью которой независимые пользователи могли бы общаться друг с другом; они только совместно используют конфигурацию и библиотеку процедур (которая включает в себя транслятор ALGOL 60, расширенный для работы с комплексными числами). Система не рассчитана на пользовательские программы, написанные на машинном языке. <...>

28.3. Отчет о состоянии

Мы допустили несколько обычных мелких ошибок (например, уделили слишком много внимания устранению того, что в действительности не было узким местом) и две крупные.

Наша первая крупная ошибка состояла в том, что мы слишком долго ограничивали внимание «идеальным вычислительным центром»; пока мы думали, как лучше всего распорядиться одним из периферийных устройств, оно сломалось, и мы столкнулись с серьезными проблемами. Решение проблемы этой «патологии» потребовало больше энергии, чем мы ожидали, и часть наших трудностей была прямым следствием нашего хитроумия на ранних этапах, т. е. сложности ситуации, в которую система могла бы себя завести. Если бы мы уделяли внимание патологии на более ранней стадии проекти-

рования, то наши правила управления, безусловно, оказались бы не столь изощренными.

Вторая крупная ошибка заключалась в том, что мы планировали и программировали основную часть системы, мало задумываясь о том, как будем ее отлаживать. Я отказываюсь считать своей заслугой то, что эта ошибка не имела серьезных последствий. Кто-то мог бы возразить, что это запоздалая мысль.

Как капитан команды я имел обширный опыт (начиная с 1958 года) работы с базовым программным обеспечением для обработки прерываний реального времени и на собственном горьком опыте знал, что в результате невоспроизводимости момента прерывания программная ошибка может представать в ложном свете и казаться случайным сбоем оборудования. Этого я ужасно боялся. Питая опасения относительно возможности отладки, мы решили быть настолько внимательными, насколько это возможно, и – предотвращение лучше, чем исправление! – постараться предотвратить внесение ошибок в программу.

Это решение, продиктованное страхом, является основой того, что я расцениваю как главный вклад группы в искусство проектирования систем. Мы поняли, что возможно разработать изящную мультипрограммную систему таким образом, что ее логическая правильность может быть доказана априори, а реализация будет допускать исчерпывающее тестирование. Во время тестирования были выявлены только тривиальные ошибки кодирования (встречавшиеся с частотой одна ошибка на 500 команд); для локализации каждой из них потребовалось не более 10 минут (классической) проверки на машине, а для исправления – примерно столько же. На момент написания статьи тестирование еще не закончено, но результирующая система гарантированно не будет содержать ошибок. Когда система будет внедрена, мы не будем испытывать постоянный страх, что система сбойнет в какой-то маловероятной ситуации, например в результате несчастливого совпадения двух или более критических условий, поскольку мы доказали правильность системы со строгостью и ясностью, необычными для большинства математических доказательств.

28.4. Обзор структуры системы

28.4.1. Выделение памяти. В классической машине фон Неймана информация идентифицируется адресом ячейки памяти, ее содержащей. Начиная думать об автоматическом управлении вспомогательной памятью, мы были хорошо знакомы с системой (а именно Gier Algol), в которой вся информация идентифицировалась адресом на барабане (как в классической машине фон Неймана) и в которой функция оперативной памяти сводилась лишь к тому, чтобы сделать информацию «постранично» доступной.

Мы пошли по другому пути, и, как оказалось, не зря. В нашей терминологии проведено строгое различие между единицами памяти (мы называем их «страницами», существуют «оперативные страницы» и «барабанные страни-

цы») и соответственными информационными единицами (из-за отсутствия лучшего слова мы называем их «сегментами»); сегмент точно помещается в страницу. Для сегментов мы создали совершенно независимый механизм идентификации, в котором количество возможных идентификаторов сегментов значительно больше, чем общее количество страниц в основной и вспомогательной памяти. Идентификатор сегмента дает быстрый доступ к так называемой «сегментной переменной» в оперативной памяти, значение которой показывает, является сегмент пустым или нет, и если он не пуст, то в какой странице (или страницах) может быть найден.

Следствием этого подхода является тот факт, что если информационный сегмент, размещенный в оперативной памяти, должен быть выгружен на барабан, чтобы освободить оперативную страницу для другого использования, то нет необходимости возвращать сегмент на ту же барабанную страницу, где он находился первоначально. И эта свобода действительно используется: из всех свободных барабанных страниц выбирается та, для которой время задержки минимально. Другое следствие – полное отсутствие проблемы выделения памяти на барабане: например, не было никаких причин требовать, чтобы программа занимала на барабане последовательные страницы. В мультипрограммной среде это очень удобно.

28.4.2. Выделение процессора. Мы ясно понимали, что в одном последовательном процессе (например, выполняемом последовательным автоматом) логическое значение имеет только временная последовательность различных состояний, но не фактическая скорость выполнения последовательного процесса. Поэтому мы организовали систему как совокупность последовательных процессов, относительные скорости которых не определены. Каждой пользовательской программе, поступающей в систему, соответствует последовательный процесс, каждому периферийному устройству ввода соответствует последовательный процесс (буферизирующий входные потоки синхронно с выполнением команд ввода), каждому периферийному устройству вывода соответствует последовательный процесс (дебуферизирующий выходные потоки синхронно с выполнением команд вывода); кроме того, имеется «контроллер сегментов», связанный с барабаном, и «интерпретатор сообщений», связанный с клавиатурой консоли.

Это дало нам возможность проектировать всю систему в терминах абстрактных «последовательных процессов». Их гармоничное взаимодействие регулируется посредством явных предложений взаимной синхронизации. С одной стороны, эта явная взаимная синхронизация была необходима, поскольку мы не делаем никаких предположений об отношении скоростей; с другой стороны, она возможна, поскольку «вре́менная задержка выполнения другого процесса» никогда не может вступить в противоречие с внутренней логикой задержанного процесса. Фундаментальным следствием этого подхода – явной взаимной синхронизации – является то, что гармоничное взаимодействие набора таких последовательных процессов может быть установлено с помощью дискретных рассуждений; еще одно следствие заключается в том, что вся гармоническая совокупность взаимодействующих последовательных процессов не зависит от реального числа доступных им

процессоров, при условии что доступные процессоры могут переключаться с одного процесса на другой.

28.4.3. Иерархия системы. Вся система имеет четкую иерархическую структуру.

На уровне 0 лежит ответственность за выделение процессора одному из процессов, для которых логически допускается динамическое выполнение (например, ввиду явной взаимной синхронизации). На этом уровне обрабатывается прерывание от генератора реального времени, введенное для того, чтобы не допустить монополизации процессора одним процессом. На этом уровне введены правила приоритетов для достижения быстрой реакции системы, когда это необходимо. Здесь реализована наша первая абстракция: выше уровня 0 реальное число совместно используемых процессоров уже не имеет значения. На более высоких уровнях мы обнаруживаем только деятельность различных последовательных процессов, реальные процессоры теряют свою идентичность и исчезают из общей картины.

На уровне 1 мы имеем так называемый «контроллер сегментов», последовательный процесс, синхронизированный с прерываниями от барабана и с последовательными процессами верхних уровней. На уровне 1 лежит ответственность за учет использования ресурсов вспомогательной памяти. На этом уровне реализована наша следующая абстракция: на всех более высоких уровнях информация идентифицируется в терминах сегментов, реальные же страницы памяти теряют свою идентичность и исчезают из общей картины.

На уровне 2 находится «интерпретатор сообщений», управляющий выделением клавиатуры консоли, с помощью которой ведется диалог между оператором и процессами верхних уровней. Интерпретатор сообщений работает синхронно с оператором. Когда оператор нажимает клавишу, машине отправляется символ вместе с сигналом прерывания, объявляющим о поступлении этого символа. Печать же символа осуществляет команда вывода, генерируемая машиной под управлением интерпретатора сообщений. (С точки зрения аппаратуры, консольный телетайп рассматривается как два независимых периферийных устройства: входная клавиатура и выходной принтер.) Когда один из процессов открывает диалог, он идентифицирует себя в открывающей фразе диалога для удобства оператора. Если же диалог открывает оператор, то он должен идентифицировать процесс, к которому обращается, в открывающей фразе диалога, т. е. эта открывающая фраза должна быть интерпретирована до того, как станет известно, к какому процессу диалог адресуется! В этом и состоит логическая причина введения отдельного последовательного процесса для консольного телетайпа, отраженная в его названии – «интерпретатор сообщений».

Выше уровня 2 все выглядит так, будто каждый процесс имеет собственную диалоговую консоль. Тот факт, что все они совместно используют одну и ту же физическую консоль, транслируется в ограничение на ресурсы вида «только один диалог в каждый момент времени», которое удовлетворяется благодаря взаимной синхронизации. На этом уровне реализована следующая абстракция: на всех более высоких уровнях реальный консольный телетайп теряет свою идентичность. (Если бы интерпретатор сообщений не находился на более высоком уровне, чем контроллер сегментов, то реализовать его можно

было бы только путем резервирования для него постоянного места в оперативной памяти; поскольку словарь диалога может быть большим (если операторы захотят получить заковыристые приветственные сообщения), это повлечет за собой слишком высокий спрос на постоянную оперативную память. Поэтому словарь, на котором выражаются сообщения, хранится в сегментах, т. е. информационных единицах, которые могут находиться также и на барабане. Именно из-за этого интерпретатор сообщений находится уровнем выше контроллера сегментов.)

На уровне 3 мы находим последовательные процессы, связанные с буферизацией входных потоков и дебуферизацией выходных потоков. На этом уровне осуществляется следующая абстракция, а именно абстракция фактических периферийных устройств. Они выделяются на нем как «логические коммуникационные устройства», в терминах которых работают еще более высокие уровни. Последовательные процессы, связанные с периферийными устройствами, находятся уровнем выше интерпретатора сообщений, потому что они должны быть в состоянии вести диалог с оператором (например, в случае обнаружения сбоя). Ограниченное количество периферийных устройств также выступает в роли ограничения на ресурсы для процессов верхних уровней, которое удовлетворяется путем их взаимной синхронизации.

На уровне 4 мы находим независимые программы пользователей, на уровне 5 – оператора (не реализован нами).

Структура системы была описана так подробно, для того чтобы был понятен следующий раздел.

28.5. Опыт проектирования

Стадия концептуального планирования заняла много времени. В этот период родились концепции в тех терминах, в которых система была описана выше. Кроме того, мы научились искусству рассуждений, позволяющих выводить из требований то, как процессы должны влиять друг на друга с соблюдением взаимной синхронизации, чтобы удовлетворить этим требованиям. (Требования состояли в том, что информация не должна использоваться прежде, чем она будет произведена, что ни одно внешнее устройство не должно быть выделено двум задачам одновременно и т. д.) Наконец, мы научились искусству рассуждений, позволяющих доказывать, что поведение во времени совокупности взаимно синхронизированных процессов действительно удовлетворяет всем требованиям.

Стадия конструирования была традиционной, возможно даже старомодной: обычный машинный код. Перепрограммирование из-за изменения спецификаций было редким явлением – обстоятельство, которое следует приписать в основном «паровому методу». Тот факт, что первые две стадии заняли больше времени, чем планировалось, отчасти был компенсирован задержкой в предоставлении нам машины.

На стадии верификации мы на короткое время получали машину в свое полное распоряжение, в это время мы работали с «голой» машиной без како-

го-либо программного обеспечения для отладки. Система тестировалась, начиная с уровня 0, и каждый новый уровень (или его часть) добавлялся только после того, как предыдущий уровень был полностью протестирован. Каждый сеанс тестирования содержал поверх тестируемой (части) системы ряд тестовых процессов двойного назначения. Во-первых, они должны были переводить систему в различные релевантные состояния, а во-вторых, проверять, что система продолжает реагировать в соответствии со спецификациями.

Не стану отрицать, что конструирование этих тестовых программ потребовало значительных интеллектуальных усилий: убедиться, что не пропущено никакое «релевантное состояние» и что тестовые программы генерируют все такие состояния, – непростая задача. Обнадеживает то, что (как мы теперь знаем!) это возможно.

Этот факт был одним из счастливых следствий иерархической структуры!

Тестирование уровня 0 (генератор реального времени и выделение процессора) подразумевало наличие над ним ряда тестовых последовательных процессов, которые совместно проверяют, что при любых обстоятельствах процессорное время делится между ними в соответствии с правилами. После того как это было установлено, были реализованы последовательные процессы как таковые.

Тестирование контроллера сегментов на уровне 1 означало, что все «релевантные состояния» могут быть сформулированы в терминах последовательных процессов, генерирующих (в разных комбинациях) спрос на страницы оперативной памяти, – ситуация, которую можно создать посредством явной синхронизации тестовых программ. На этом этапе существование генератора реального времени – хотя он все время генерировал прерывания – было настолько неощутимым, что один из тестировщиков даже забыл о его существовании!

К этому моменту мы уже реализовали корректную реакцию на (взаимно несинхронизированные) прерывания от генератора реального времени и барабана. Если бы мы не ввели раздельные уровни 0 и 1 и не разработали терминологию (довольно «абстрактных» последовательных процессов), в которой существованием прерываний от генератора времени можно пренебречь, а пытались бы вместо этого создать неиерархическую конструкцию, в которой центральный процессор непосредственно реагировал бы на любую, сколь угодно странную, временную последовательность этих двух прерываний, то количество «релевантных состояний» могло бы увеличиться до такой степени, что протестировать их исчерпывающим образом оказалось бы невозможно. (Не говоря уже о том, что мы вряд ли смогли бы сгенерировать все такие ситуации, потому что не могли контролировать скорость работы барабана и генератора времени.) <...>

28.6. Заключение

В части верификации программы я не представил ничего существенно нового. При тестировании объектов общего назначения (будь это часть обо-

рудования, программа, машина или система) мы не можем проверить его во всех возможных случаях: для компьютера это означало бы подачу на вход всех возможных программ! Следовательно, мы должны тестировать его на наборе релевантных примеров. Решение о том, что релевантно, а что нет, невозможно принять, коль скоро механизм рассматривается как черный ящик; иными словами, мы должны идти от внутренней структуры тестируемого механизма. Представляется, что на проектировщике лежит ответственность за конструирование механизма именно таким – т. е. хорошо структурированным – образом, чтобы на каждом этапе процедуры тестирования число релевантных состояний было настолько малым, чтобы их можно было перебрать целиком, и чтобы то, что тестируется, было настолько заметно, что пропустить какую-то ситуацию попросту было бы невозможно. Я представил обзор нашей системы, потому что, на мой взгляд, это прекрасный пример формы, которую могла бы принять такая структура.

Должен с сожалением сказать, что, по моему опыту, производители промышленного программного обеспечения реагируют на систему со смешанными чувствами. С одной стороны, они склонны полагать, что мы построили некую модель, а с другой стороны, сомневаются в том, что использованные нами приемы применимы за пределами оранжерейной атмосферы университета, и полагают, что успех сопутствовал нам только благодаря скромным масштабам всего проекта. Я вовсе не призываю недооценивать организационные способности, необходимые для управления значительно более масштабной работой, которой занято много больше людей, но рискну предположить, что чем крупнее проект, тем важнее структурирование! Иерархия из пяти логических уровней на практике может оказаться мелковата, особенно если проектировать систему более осмысленно, чем это делали мы, поставив целью сделать программное обеспечение легко адаптируемым к (возможно, кардинальным) расширениям конфигурации. <…>

28.7. Приложение

28.7.1. Синхронизирующие примитивы. Явная взаимная синхронизация параллельно выполняющихся последовательных процессов реализована с помощью так называемых «семафоров». Это специальные целочисленные переменные, выделяемые в той среде, в которой выполняются процессы; они инициализируются (значениями 0 или 1) до того, как стартуют сами параллельные процессы. После такой инициализации параллельные процессы могут обращаться к семафорам только с помощью двух весьма специфических операций: синхронизирующих примитивов. По историческим причинам они названы *P*-операцией и *V*-операцией.

Процесс, скажем «*Q*», который выполняет операцию «*P*(sem)», уменьшает значение семафора «sem» на 1. Если результирующее значение этого семафора неотрицательно, то процесс *Q* может продолжать выполнение следующего предложения, но если результирующее значение отрицательно, то процесс *Q* останавливается и помещается в список ожидания, связанный с этим сема-

фором. До будущего уведомления (т. е. до *V*-операции над тем же семафором) динамическое продолжение процесса *Q* логически запрещено, и процессор ему выделяться не будет (см. выше «Иерархия системы», уровень 0).

Процесс, скажем «*R*», который выполняет операцию «*V*(sem)», увеличивает значение семафора «sem» на 1. Если результирующее значение этого семафора положительно, то данная *V*-операция больше ничего не делает; если же значение этого семафора неположительно, то один из процессов, находящихся в списке ожидания, удаляется из этого списка, т. е. его выполнение снова логически разрешено, и в должный момент ему будет выделено процессорное время (опять-таки см. выше «Иерархия системы», уровень 0).

Следствие 1. *Если значение семафора неположительное, то его абсолютная величина равна числу процессов, находящихся в его списке ожидания.*

Следствие 2. *P-операция представляет потенциальную задержку, дополнительная к ней V-операция представляет снятие барьера.*

Примечание 1. P- и *V*-операции являются «неделимыми действиями»; т. е., встретившись «одновременно» в параллельных процессах, они не накладываются друг на друга, в том смысле, что их можно рассматривать как выполняемые одна после другой.

Примечание 2. Если результирующее значение семафора после *V*-операции отрицательно, то его список ожидания первоначально содержал более одного процесса. Не определено – т. е. логически несущественно, – какой из ожидавших процессов удаляется из списка ожидания.

Примечание 3. Следствием описанного выше механизма является то, что процесс, чье динамическое выполнение разрешено, может потерять этот статус только при реальном выполнении, т. е. выполнив *P*-операцию над семафором, значение которого неположительно.

Во время концептуального планирования системы выяснилось, что мы используем семафоры двумя совершенно разными способами. Разница между ними настолько заметная, что, оглядываясь назад, мы удивляемся, честно ли было предоставлять два способа для использования одних и тех же примитивов. С одной стороны, мы имеем семафоры, используемые для взаимного исключения, с другой стороны – частные семафоры.

28.7.2. Взаимное исключение. В следующей программе показаны два параллельных циклических процесса (между операторными скобками «**parbegin**» и «**parend**»), которые начинают работать после создания и инициализации среды.

```
begin semaphore mutex; mutex := 1;
   parbegin
      begin
         L1 :  P(mutex); критическая секция 1; V(mutex);
               оставшаяся часть цикла 1; go to L1
      end;
      begin
         L2 : P(mutex); критическая секция 2; V(mutex);
               оставшаяся часть цикла 2; go to L2
      end
   parend
end
```

В результате *P*- и *V*-операций над «mutex» действия, отмеченные как «критические секции», взаимно исключают друг друга; данная схема допускает простое расширение на более чем два параллельных процесса, максимальное значение мьютекса равно 1, минимальное значение равно –(*n* – 1), если имеется *n* параллельных процессов.

Критические секции используются всегда и исключительно для обеспечения однозначности при проверке и модификации переменных состояния (выделенных в окружающей среде), которые описывают текущее состояние системы (в той мере, в какой это необходимо для регулирования гармоничного взаимодействия разных процессов).

28.7.3. Частные семафоры. С каждым последовательным процессом связан ряд частных семафоров, и никакой другой процесс не может выполнить над ними *P*-операцию. Среда инициализирует эти семафоры значением 0, их максимальное значение равно 1, а минимальное значение равно –1.

Достигнув стадии, на которой разрешение его динамического выполнения зависит от текущих значений переменных состояния, процесс выполняет такой псевдокод:

```
P(mutex);
«проверка и модификация переменных состояния,
    включая условие V(частный семафор)»;
V(mutex);
P(частный семафор).
```

Если проверка определяет, что данный процесс должен продолжаться, то он выполняет операцию «*V*(частный семафор)» – значение семафора при этом изменяется с 0 на 1, – в противном случае эта *V*-операция пропускается, и обязанность выполнить ее в подходящий момент перекладывается на другие процессы. Отсутствие или наличие этой обязанности отражается на конечных значениях переменных состояния при выходе из критической секции.

Достигнув стадии, на которой в результате его выполнения один (или несколько) блокированных процессов должны теперь получить разрешение на продолжение, процесс выполняет такой псевдокод:

```
P(mutex);
«модификация и проверка переменных состояния, включая нуль
или более V-операций над частными семафорами других процессов»;
V(mutex).
```

Введением подходящих переменных состояния и соответствующим программированием критических секций может быть реализована любая стратегия назначения внешних устройств, буферных областей и т. д.

Объем кодирования и рассуждений можно значительно сократить, заметив, что в двух взаимодополнительных критических секциях, показанных выше, такую проверку можно выполнить путем введения понятия «неустойчивой ситуации», например: имеется свободный читатель и процесс, нуждающийся в читателе. Если неустойчивая ситуация возникает, она ликвидируется (включая одну или более *V*-операций над частными семафорами) именно в той же критической секции, в которой возникла.

28.7.4. Обеспечение гармоничного взаимодействия. Все последовательные процессы в системе можно рассматривать как циклические процессы, в которых отмечена некоторая нейтральная точка, так называемая «начальная позиция», в которой все процессы находятся, когда система пребывает в состоянии покоя.

Когда циклический процесс покидает свою начальную позицию, «он принимает задачу»; только когда задача будет выполнена, не раньше, процесс возвращается в начальную позицию. Каждый циклический процесс имеет определенную задачу, потребляющую процессор (например, выполнение пользовательской программы, дебуферизация порции данных для вывода на принтер и т. д.).

Гармоничность взаимодействия доказана на трех стадиях.

1. Доказано, что хотя процесс, выполняющий задачу, может по ходу дела порождать конечное число задач для других процессов, единственная начальная задача не может породить бесконечное число задач. Доказательство простое: поскольку процесс может порождать задачи только для процессов на более низких уровнях иерархии, то цикличность исключена. (Если процесс, которому нужен сегмент с барабана, породил задачу для контроллера сегментов, должны быть приняты специальные меры, гарантирующие, что запрошенный сегмент останется в оперативной памяти по крайней мере до тех пор, пока запрашивающий процесс не обратится к этому сегменту. Без таких предосторожностей можно было бы заставить конечное число задач порождать бесконечное число задач для контроллера сегментов, и система начала бы пробуксовывать, занимаясь ни к чему не ведущим страничным обменом.)
2. Доказано, что не может быть так, что все процессы вернулись в свои начальные позиции, в то время как где-то в системе еще ожидает порожденная, но не принятая задача. (Это доказывается путем демонстрации неустойчивости только что описанной ситуации.)
3. Доказано, что после принятия начальной задачи все процессы в конце концов (снова) окажутся в своих начальных позициях. Каждый процесс, блокированный в ходе выполнения задачи, полагается на то, что другие процессы устранят барьер. По существу, доказательство заключается в демонстрации отсутствия «циклического ожидания»: процесс *P* ожидает процесса *Q*, ожидающего процесса *R*, ожидающего процесса *P*. (Обычный термин для циклического ожидания – «смертельные объятия».) <…>

29 О вреде оператора go to (1968)

Эдсгер Дейкстра

Движение за «структурное программирование» не возникло на пустом месте. Оно развивалось по мере взросления индустрии программного обеспечения: компиляторы совершенствовались, поэтому у программистов стало меньше поводов писать хитроумный код, чтобы сэкономить несколько тактов процессора или байтов памяти. С другой стороны, системы укрупнялись, поэтому команды стремились писать более понятный код с легко объяснимой модульной структурой. У первых процессоров не было команд для работы со стеком. Но к 1968 году рекурсия уже перестала быть экзотикой и поддерживалась архитектурно, а циклические конструкции типа **while** прочно вошли в языки программирования (эта статья была написана примерно в то время, когда Вирт проектировал язык Pascal).

Поэтому, как признавал сам Дейкстра (см. главу 26), эта статья – просто «письмо редактору», оно является не столько вкладом в науку, сколько подведением итогов ширящемуся движению, дополненным призывом к действию. Так это или нет, но статья оказала значительное влияние на практику программирования – в следующем десятилетии слова «структурное программирование» встречались в заголовках сотен статей и книг. Как и любой призыв к ортодоксальности, письмо Дейкстры встретило и оппозицию или, по крайней мере, пожелания большой гибкости. Дональд Кнут (Knuth 1974a) доказывал, что некоторые вещи проще делаются с помощью оператора go to, и высказывал мнение, что структурное программирование вполне может сосуществовать с go to. Но в общем и целом поле боя осталось за аргументами Дейкстры – были внесены изменения в методики преподавания программирования и определены минимальные требования к лингвистическим конструкциям в языках программирования.

Еще одно замечание по тексту: фраза «considered harmful» (о вреде), ныне склоняемая на разные лады авторами работ по информатике, не принадлежит Дейкстре. Редактор, Никлаус Вирт, придумал этот заголовок, решив опубликовать эту заметку (в оригинале она называлась «A Case Against the GOTO Statement»[1]) как письмо, а не научную статью. Но и Вирту эта формулировка тоже не принадлежит; к моменту опубликования статьи она уже вошла в журналистскую традицию (Laplante 1996, стр. 420).

Редактору. За многие годы я утвердился во мнении о том, что квалификация программистов – функция, обратно зависящая от частоты появления операторов **go to** в их программах. Позже я открыл, почему оператор **go to** производит такой пагубный эффект, и я убежден в том, что оператор **go to** должен быть отменен в языках программирования «высокого уровня» (т. е. отовсюду, кроме, возможно, простого машинного кода). В то время я не счел это открытие слишком важным. Теперь же я отправляю свои соображения для публикации, потому что меня подтолкнула к этому развернувшаяся сейчас дискуссия на эту тему.

Мое первое замечание состоит в том, что хотя деятельность программиста заканчивается с конструированием корректной программы, процесс, который выполняется под управлением этой программы, является истинной сущностью этой деятельности; этот процесс должен завершиться с заданным эффектом; поведение этого процесса должно удовлетворять заданным спецификациям. Хотя, когда программа уже написана, «изготовление» соответствующего процесса поручается машине.

Мое второе замечание – то, что наши интеллектуальные силы в большей степени ориентированы на статические отношения и что наши способности представлять процессы, развивающиеся во времени, развиты относительно плохо. Исходя из этого, мы должны делать (как мудрые программисты, осознающие свои ограничения) все возможное, чтобы сократить концептуальную пропасть между статической программой и динамическим процессом, чтобы сделать соответствие между программой (разворачивающейся в пространстве текста) и процессом (разворачивающимся во времени) настолько очевидным, насколько это возможно.

Давайте теперь рассмотрим, как можно охарактеризовать состояние процесса. (Мы можем представить себе этот вопрос очень конкретно: предположим, что процесс, рассматриваемый как последовательность действий во времени, остановлен после произвольного действия; какие данные необходимо иметь, чтобы точно определить порядок, в котором следует повторить процесс, чтобы дойти до той же точки?) Если текст программы представляет собой просто конкатенацию, скажем, операторов присваивания (в рамках данной дискуссии будем рассматривать их как описания одиночных действий), то достаточно в тексте программы указать точку между двумя последовательными описаниями действий (successive action descriptions). (В отсутствие оператора **go to** я могу позволить себе синтаксическую не-

[1] Дело против оператора GOTO. – *Прим. перев.*

однозначность в последних трех словах предыдущего предложения: если мы разбираем их как «последовательные (описания действий)», то имеем в виду последовательность в тексте; если же мы разбираем их как «описания (последовательных действий)», то имеем в виду последовательность во времени.) Давайте назовем указатель на соответствующее место в тексте «текстуальным индексом».

Когда мы вводим условное предложение **(if** *B* **then** *A***)**, предложение с альтернативами **(if** *B* **then** *A*1 **else** *A*2**)**, предложения выбора, введенные К. А. Р. Хоаром (case [i] of (*A*1, *A*2, ..., *An*)), или условные выражения, предложенные Дж. Маккарти (*B*1 → *E*1, *B*2 → *E*2, ..., *Bn* → *En*), состояние процесса по-прежнему характеризуется единственным текстуальным индексом.

Но как только в язык включаются процедуры, приходится признать, что единственного текстуального индекса уже недостаточно. В случае если текстуальный индекс указывает внутрь тела процедуры, динамическое состояние может быть однозначно охарактеризовано, только если известно также, с каким вызовом процедуры он связан. С введением процедур мы можем охарактеризовать состояние процесса при помощи последовательности текстуальных индексов, длина которой должна быть равна динамической глубине вызова процедуры.

Теперь давайте рассмотрим предложения повторения (например, **while** *B* **repeat** *A* или **repeat** *A* **until** *B*). С позиций логики эти предложения избыточны, потому что мы можем выразить повторения при помощи рекурсивных процедур. Из соображений реалистичности я не хочу их исключать: с одной стороны, предложения повторений могут быть весьма удобно реализованы на современном ограниченном оборудовании, с другой стороны, модель мышления, известная как «индукция», делает нас хорошо подготовленными к интеллектуальному восприятию процессов, порождаемых предложениями повторения. С введением предложений повторения текстуальных индексов уже недостаточно для описания динамического состояния процесса. Однако с каждым входом в предложения повторения мы можем связать так называемый «динамический» индекс, исправно подсчитывающий порядковый номер текущего повторения. Если предложения повторения (так же, как вызовы процедур) могут быть вложенными, то мы обнаруживаем, что развитие процесса может быть однозначно охарактеризовано последовательностью (смешанной) текстуальных и (или) динамических индексов.

Суть в том, что значения этих индексов не контролируются программистом; они генерируются (то ли при компиляции его программы, то ли при динамическом выполнении процесса) независимо от того, желает он того или нет. Они обеспечивают независимые координаты, в которых описывается состояние процесса. Зачем нам такие независимые координаты? Причина в том – и это представляется неотъемлемым свойством последовательных процессов, – что мы можем интерпретировать значение переменной только с учетом состояния процесса. Если мы хотим подсчитать число *n*, скажем, людей в комнате, первоначально пустой, то можем достичь этого путем увеличения *n* на единицу всякий раз, как видим, что кто-то вошел в комнату. Между моментом, когда мы увидели входящего в комнату, и моментом, когда увеличили *n*, значение *n* равно числу людей в комнате минус единица!

Из-за разнузданного применения оператора **go to** становится весьма трудно найти осмысленный набор координат, в которых можно было бы описать состояние процесса. Обычно во внимание принимают также и значения некоторых избранных переменных, но тут это не пройдет, потому что от состояния процесса зависит, как интерпретировать эти значения! Конечно, и с оператором **go to** все-таки возможно однозначно описать состояние процесса, включив в рассмотрение счетчик количества действий, выполненных от момента запуска программы (т. е. некоторую разновидность нормированных часов). Трудность в том, что такая координата, хотя и однозначная, абсолютно бесполезна. В такой системе координат чрезвычайно трудно определить все те точки выполнения программы, в которых n равно числу людей в комнате минус единица (к примеру)!

Оператор **go to** в том виде, в каком он существует, попросту слишком примитивен; он создает довольно сильное искушение внести неразбериху в программу. Представленные пункты могут рассматриваться и оцениваться как попытка ограничить его применение. Я не утверждаю, что упомянутые пункты являются исчерпывающими в том смысле, что удовлетворяют всем требованиям, но какие бы соображения ни высказывались (например, предложение прекращения), они обязаны удовлетворять важному требованию: должна существовать независимая от программиста система координат для описания процесса полезным и управляемым способом.

В завершение следовало бы высказать благодарности, но трудно решить, кому именно. Могу ли я судить о том, кто оказал влияние на мои рассуждения? Совершенно очевидно, что не обошлось без влияния Питера Ландиса и Кристофера Стрейчи. Наконец, я хочу отметить (я помню это совершенно точно), как Хайнц Земанек на встрече, посвященной предварительному обсуждению языка Algol, в начале 1959 г. в Копенгагене предельно ясно выразил свои сомнения в том, должен ли оператор **go to** трактоваться как синтаксически равный оператору присваивания. В какой-то степени я виню себя за то, что не оценил тогда по достоинству следствия из его замечания.

Замечания о нежелательности оператора **go to** далеко не новы. Я помню, что читал явные рекомендации ограничить использование **go to** аварийным выходом, но не помню точно где, вероятно, у Ч. Э. Р. Хоара. В работе (Wirth and Hoare 1966, раздел 3.2.1) Вирт и Хоар делают замечание в том же направлении, обосновывая языковую конструкцию выбора: «Как и условные предложения, она отражает динамическую структуру программы более ясно, чем оператор **go to** и переключатели, и при этом устраняет необходимость введения в программу большого числа меток».

В работе (Böhm and Jacopini 1966), кажется, доказана (логическая) избыточность оператора **go to**. Однако практиковать более или менее механическую трансляцию произвольной блок-схемы в блок-схему без переходов не рекомендуется. В таком случае трудно ожидать, что результирующая блок-схема будет понятнее исходной.

30 Метод исключения Гаусса не оптимален (1969)

Фолькер Штрассен

Умножение матриц – такая простая операция, что трудно представить, что в ней осталось что-то неизученное. Чтобы перемножить две матрицы A и B размера $n \times n$, получив в результате матрицу C размера $n \times n$, нужно вычислить n^2 скалярных произведений строк A на столбцы B. Вычисление каждого скалярного произведения требует n умножений и $n - 1$ сложений чисел, т. е. всего получается n^3 умножений и $n^2(n - 1)$ сложений. Что тут еще можно сказать?

Оказывается, много чего. Немецкий математик Фолькер Штрассен (родился в 1936 году), возможно, пытался доказать нижнюю границу, т. е. что n^3 умножений не только достаточно, но и необходимо, а в результате открыл описываемый ниже алгоритм. В статье есть две примечательные идеи. Первая – что рекурсивный алгоритм типа «разделяй и властвуй» может оказаться лучше традиционного, если существует способ вычислить произведение матриц 2×2, выполнив меньше 8 умножений. Но даже после доказательства этого факта все равно испытываешь удивление от того, что накладные расходы на реализацию рекурсии асимптотически окупаются. Второе удивительное открытие заключается в том, что для вычисления произведения двух матриц 2×2 достаточно 7 умножений. Любой школьник мог бы убедиться в этом на переменке, имея только листок бумаги; но за те несколько столетий, что люди перемножают матрицы, никто не обратил на это внимания, потому что не возникало необходимости. (Аналогичный алгоритм умножения целых чисел Карацубы и Офмана [Karatsuba and Ofman 1962] к тому времени был уже известен. Он рекурсивно вычисляет произведение двух $2n$-битовых чисел, выполняя три умножения n-битовых чисел, так что время перемножения

n-битовых чисел имеет порядок $O(n^{\log_2 3}) \approx n^{1.58}$, что лучше традиционного алгоритма, имеющего сложность $\Theta(n^2)$.)

Алгоритм Штрассена нелегко реализовать корректно и эффективно, но его полезность в случае хорошей реализации не просто теоретическая. Открытие того факта, что матрицы $n \times n$ можно перемножить, выполнив $n^{\log_2 7} \approx n^{2.8}$ умножений, привело к до сих пор не решенной задаче о минимальном показателе степени. На момент написания этой книги было доказано, что он не превышает 2.373, но можно ли сделать его меньше 2, неизвестно; впрочем, эти более экзотические алгоритмы практически бесполезны.

Эта статья, наряду с работой Karatsuba and Ofman (1962), ввела метод «разделяй и властвуй» в арсенал средств решения разнообразных алгоритмических проблем. А ее следствия для эффективного решения систем линейных уравнений – оно и вынесено в заголовок статьи – сами по себе примечательны.

30.1

Ниже описывается алгоритм, который вычисляет коэффициенты произведения двух квадратных матриц A и B порядка n по коэффициентам A и B менее, чем за $4.7 \cdot n^{\log 7}$ арифметических операций (все логарифмы в этой статье двоичные, т. е. $\log 7 \approx 2.8$; обычный же метод требует приблизительно $2n^3$ арифметических операций). Этот алгоритм порождает алгоритмы обращения матрицы порядка n, решения системы n линейных уравнений с n неизвестными, вычисления определителя порядка n и т. д.; все они требуют меньше $Cn^{\log 7}$ арифметических операций, где C – постоянная.

Этот факт следует сравнить с результатом, полученным в работе Klyuev and Kokovkin-Shcherbak (1965), о том, что метод исключения Гаусса для решения системы линейных уравнений оптимален, если только ограничиться операциями умножения строк и столбцов. Отметим также, что в работе Winograd (1968) обычные алгоритмы умножения и обращения матриц и решения систем линейных уравнений модифицированы так, что примерно половина умножений была заменена сложениями и вычитаниями. Автору доставляет большое удовольствие выразить благодарность Д. Бриллингеру за плодотворное обсуждение предмета настоящей работы и С. Куку и Б. Парлетту, побудившим меня написать эту статью.

30.2

Мы определим алгоритмы $\alpha_{m,k}$ умножения матриц порядка $m2^k$ индукцией по k: $\alpha_{m,0}$ – обычный алгоритм умножения матриц (требующий m^3 умножений и $m^2(m-1)$ сложений). Если $\alpha_{m,k}$ уже известен, то определим $\alpha_{m.k+1}$ следующим образом.

Если A, B – перемножаемые матрицы порядка $m2^{k+1}$, то пусть

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \qquad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

где A_{ik}, B_{ik}, C_{ik} – матрицы порядка $m2^k$. Теперь вычислим

$$\begin{aligned}
\mathrm{I} &= (A_{11} + A_{22})(B_{11} + B_{22});\\
\mathrm{II} &= (A_{21} + A_{22})B_{11};\\
\mathrm{III} &= A_{11}(B_{12} - B_{22});\\
\mathrm{IV} &= A_{22}(-B_{11} + B_{21});\\
\mathrm{V} &= (A_{11} + A_{12})B_{22};\\
\mathrm{VI} &= (-A_{11} + A_{21})(B_{11} + B_{12});\\
\mathrm{VII} &= (A_{12} - A_{22})(B_{21} + B_{22});\\
C_{11} &= \mathrm{I} + \mathrm{IV} - \mathrm{V} + \mathrm{VII};\\
C_{21} &= \mathrm{II} + \mathrm{IV};\\
C_{12} &= \mathrm{III} + \mathrm{V};\\
C_{22} &= \mathrm{I} + \mathrm{III} - \mathrm{II} + \mathrm{VI},
\end{aligned}$$

используя $\alpha_{m,k}$ для умножения и обычный алгоритм для сложения и вычитания матриц порядка $m2^k$.

Применяя индукцию по k, легко видеть, что

Факт 1. $\alpha_{m,k}$ вычисляет произведение двух матриц порядка $m2^k$ за $m^3 7^k$ умножений и $(5 + m)m^2 7^k - 6(m2^k)^2$ сложений и вычитаний чисел.

Таким образом, умножить две матрицы порядка 2^k можно за 7^k умножений и менее 6×7^k сложений и вычитаний.

Факт 2. Произведение двух матриц порядка n можно вычислить, выполнив менее $4.7n^{\log 7}$ арифметических операций.

Доказательство. Пусть $k = \lfloor \log n - 4 \rfloor$, $m = \lfloor n2^{-k} \rfloor + 1$, тогда $n \leqslant m2^k$. Замена матриц порядка n матрицами порядка $m2^k$ сводит нашу задачу к задаче оценки числа операций в алгоритме $\alpha_{m,k}$. В силу факта 1, это число равно

$$\begin{aligned}
&(5 + 2m)m^2 7^k - 6(m2^k)^2\\
&\quad < (5 + 2(n2^{-k} + 1))(n2^{-k} + 1)^2 7^k\\
&\quad < 2n^3(7/8)^k + 12.03n^2(7/4)^k \quad \text{(здесь использован тот факт, что } 16 \cdot 2^k \leqslant n)\\
&\quad = (2(8/7)^{\log n - k} + 12.03(4/7)^{\log n - k})n^{\log 7}\\
&\quad \leqslant \max_{4 \leqslant t \leqslant 5}(2(8/7)^t + 12.03(4/7)^t)n^{\log 7}\\
&\quad \leqslant 4/7 \cdot n^{\log 7}
\end{aligned}$$

в силу выпуклости.

Рассмотрим теперь обращение матрицы. Для применения описанных ниже алгоритмов необходимо предположить не только, что матрица обратима, но и что все встречающиеся операции деления имеют смысл (разумеется, аналогичное предположение необходимо и в методе исключения Гаусса).

Определим алгоритмы $\beta_{m,k}$ обращения матриц порядка $m2^k$ индукцией по k: $\beta_{m,}0$ – обычный метод исключения Гаусса. Если $\beta_{m,k}$ уже известен, то определим $\beta_{m,k+1}$ следующим образом.

Если A – подлежащая обращению матрица порядка $m2^{k+}1$, то пусть

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad A^{-1} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

где A_{ik}, C_{ik} – матрицы порядка $m2^k$. Тогда вычисляем

$$\begin{aligned} \mathrm{I} &= A_{11}^{-1}; \\ \mathrm{II} &= \mathrm{A}_{21} \cdot \mathrm{I}; \\ \mathrm{III} &= \mathrm{I} \cdot \mathrm{A}_{12}; \\ \mathrm{IV} &= \mathrm{A}_{21} \cdot \mathrm{III}; \\ \mathrm{V} &= \mathrm{IV} - \mathrm{A}_{22}; \\ \mathrm{VI} &= \mathrm{V}{-}1; \\ \mathrm{C}_{12} &= \mathrm{III} \cdot \mathrm{VI}; \\ \mathrm{C}_{21} &= \mathrm{VI} \cdot \mathrm{II}; \\ \mathrm{VII} &= \mathrm{III} \cdot \mathrm{C}_{21}; \\ \mathrm{C}_{11} &= \mathrm{I} - \mathrm{VII}; \\ \mathrm{C}_{22} &= -\mathrm{VI}, \end{aligned}$$

используя алгоритм $\alpha_{m,k}$ для умножения, $\beta_{m,k}$ для обращения и обычный алгоритм для сложения и вычитания двух матриц порядка $m2^k$.

Используя индукцию по k, легко видеть

Факт 3. Алгоритм $\beta_{m,k}$ вычисляет обращение матрицы порядка $m2^k$ за $m2^k$ делений, не более $\frac{6}{5}m^3 7^k - m2^k$ умножений и не более $\frac{6}{5}(5 + m)m^2 7^k - 7(m2^k)^2$ сложений и умножений чисел.

Следующий факт точно так же следует из факта 2.

Факт 4. Обращение матрицы порядка n можно вычислить, выполнив меньше $5.64 \cdot n^{\log 7}$ арифметических операций.

Похожие результаты имеют место для решения системы линейных уравнений и вычисления определителя (используется тот факт, что $\det A = (\det A_{11}) \det(A_{22} - A_{21}A_{11}^{-1}A_{12})$).

31 Аксиоматическая основа компьютерного программирования (1969)

Ч. Э. Р. Хоар

Ч. Э. Р. «Тони» Хоар (родился в 1934 году) был удостоен премии Тьюринга в 1980 году «за фундаментальный вклад в определение и проектирование языков программирования» (Hoare 1981). Приведенный ниже фрагмент отражает его самый влиятельный вклад: стремление формализовать настойчивое требование Дейкстры считать компьютерное программирование разновидностью математических рассуждений и выполнить повестку, обозначенную Джоном Маккарти (McCarthy 1963): «Вместо того чтобы отлаживать программу, следует доказывать, что она отвечает спецификациям, и это доказательство должно проверяться компьютерной программой».

Хоар получил образование в Оксфорде, где изучал современную аналитическую философию и попутно познакомился с математической логикой. Он поступил на работу в компьютерную компанию Elliott Brothers, где возглавил группу, отвечавшую за разработку компилятора для языка программирования Algol 60. Благодаря этому опыту он познакомился с Эдсгером Дейкстрой и осознал важность языков, на которых идеи программирования можно было бы выражать кратко и элегантно. В 1968 году он занял должность профессора в университете Квинс в Белфасте. Там он прочел статью Роберта Флойда «Присваивание смысла программам» (Floyd, 1967), в которой описывался способ присоединения инвариантных предикатов к ребрам блок-схемы таким образом, что поведение программы можно было подвергнуть строгому

анализу. Хоар решил избавиться от нотации блок-схем и присоединять инварианты непосредственно к предложениям программы, в результате полный логический анализ перестал казаться немыслимым.

Самые плодотворные годы своей карьеры Хоар посвятил трудным проблемам программирования конкурентных систем и проектированию языковых систем, которые упростили бы написание и верификацию программ. Но его имя также навсегда будет связано с алгоритмом быстрой сортировки Quicksort, который легко реализуется и использует рекурсию неожиданным способом. Эта формулировка Quicksort (Hoare, 1962) стала еще одним побочным продуктом работы Хоара над Algol 60. Предоставив простые средства для выражения рекурсии, этот язык позволил выявить структуру алгоритма.

С момента опубликования основополагающей работы Хоара тема формальной верификации программ испытывала взлеты и падения. Теперь это стандартный инструмент порождения низкоуровневого кода, который может показаться человеку нечитаемым и, соответственно, трудным для понимания и рассуждения. В своих наиболее амбициозных формах этот предмет даже возбуждал сильную антипатию (см. главу 44). Впоследствии, по-прежнему выступая за формальные методы в программировании, Хоар признавал, что был излишне оптимистичен в оценке перспектив доказательств правильности применительно к большим системам (Hoare 1996). «Десять лет назад ученые, занимавшиеся формальными методами (и я в их числе – самый ошибающийся), предсказывали, что мир программирования с благодарностью примет любую помощь, обещанную формализацией в решении проблем обеспечения надежности, возникающих, когда программы становятся все больше, а вопросы безопасности выходят на передний план, – писал он. – Ныне программы стали очень большими и очень ответственными, выйдя далеко за пределы комфортной применимости формальных методов. Проблем и ошибок было более чем достаточно, но почти всегда их можно было отнести на счет неудовлетворительного анализа требований или неадекватного управления проектом. Оказалось, что мир не так уж сильно страдает от тех проблем, которые наши исследования изначально пытались решить».

Хотя во время написания этой работы Хоар преподавал в университете Квинс, большую часть своей карьеры он проработал профессором в Оксфорде. В 2000 году он был произведен в рыцари.

В этой работе сделана попытка исследовать логические основания компьютерного программирования с помощью методов, которые впервые были применены при изучении геометрии, а затем распространены на другие разделы математики. Речь идет о нахождении наборов аксиом и правил вывода, которые можно использовать для доказательства свойств компьютерных программ. Приводятся примеры таких аксиом и правил и формальное доказательство одной простой теоремы. Наконец, выдвигаются аргументы в пользу того, что дальнейшее изучение этих вопросов может принести важную теоретическую и практическую пользу.

31.1. Введение

Компьютерное программирование – точная наука, позволяющая в принципе вывести все свойства программы и все последствия выполнения ее в заданном окружении из текста самой программы с помощью дедуктивных рассуждений. Дедуктивные рассуждения подразумевают применение корректных правил вывода к набору корректных аксиом. Таким образом, желательно и интересно найти аксиомы и правила вывода, которые можно было бы положить в основу рассуждений о компьютерной программе. Конкретный выбор аксиом в какой-то мере зависит от выбора языка программирования. В целях иллюстрации эта работа ограничивается очень простым языком, который, по сути дела, является подмножеством всех современных процедурно ориентированных языков.

31.2. Компьютерная арифметика

Первое требование к корректному рассуждению о программе – знать свойства элементарных операций, которые она выполняет, например сложения и умножения целых чисел. К сожалению, в некоторых отношениях компьютерная арифметика отличается от арифметики, знакомой математикам, и выбирать подходящий набор аксиом следует с осторожностью. Например, аксиомы на рис. 31.1 образуют сравнительно небольшую выборку из аксиом, относящихся к целым числам. Из этого неполного набора аксиом можно вывести простые теоремы, например:

$$x = x + y \times 0;$$

$$y \leqslant r \supset r + y \times q = (r - y) + y \times (1 + q).$$

A1	$x + y = y + x$	сложение коммутативно	A7	$x + 0 = x$
A2	$x \times y = y \times x$	умножение коммутативно	A8	$x \times 0 = x$
A3	$(x + y) + z = x + (y + z)$	сложение ассоциативно	A9	$x \times 1 = x$
A4	$(x \times y) \times z = x \times (y \times z)$	умножение ассоциативно		
A5	$x \times (y + z) = x \times y + x \times z$	умножение дистрибутивно относительно сложения		
A6	$y \leqslant x \supset (x - y) + y = x$	сложение взаимно уничтожается с вычитанием		

Рис. 31.1

Приведем доказательство второй из них:

$$\begin{aligned}(r - y) + y \times (1 + q) &= (r - y) + (y \times 1 + y \times \mathrm{q}) && \text{(A5)}\\ &= (r - y) + (y + y \times q) && \text{(A9)}\\ &= ((r - y) + y) + y \times q && \text{(A3)}\\ &= r + y \times q \text{ при условии, что } y \leqslant r. && \text{(A6)}\end{aligned}$$

Аксиомы A1–A9, конечно, справедливы для бесконечного множества целых чисел, традиционно рассматриваемого в математике. Однако они также справедливы и для конечного множества «целых чисел», которыми оперируют компьютеры в предположении, что они ограничены только неотрицательными числами. Их истинность не зависит от размера множества; более того, она в значительной мере не зависит и от выбора метода обработки «переполнения», например:

1. Строгая интерпретация: результат операции, приведшей к переполнению, не определен; если переполнение происходит, то программа не завершается. Заметим, что в этом случае равенства A1–A9 строгие в том смысле, что обе части определены или не определены одновременно.
2. Верхняя граница: результатом операции, приведшей к переполнению, считается максимальное представимое значение.
3. Модульная арифметика: результат операции, приведшей к переполнению, вычисляется по модулю размера множества представимых целых чисел.

Все три метода продемонстрированы на рис. 31.2 в виде таблицы сложения и умножения для тривиально малой модели, в которой единственными представимыми целыми числами являются 0, 1, 2 и 3.

+	0	1	2	3
0	0	1	2	3
1	1	2	3	*
2	2	3	*	*
3	3	*	*	*

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	*	*
3	0	3	*	*

Строгая интерпретация

+	0	1	2	3
0	0	1	2	3
1	1	2	3	3
2	2	3	3	3
3	3	3	3	3

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	3
3	0	3	3	3

Верхняя граница

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Модульная арифметика

Рис. 31.2

Интересно отметить, что различные системы, удовлетворяющие аксиомам A1–A9, можно строго отличить друг от друга, выбрав одну конкретную аксиому из множества взаимно исключающих альтернатив. Например, бесконечная арифметика удовлетворяет аксиоме

$$\neg\exists x \forall y\ (y \leqslant x), \qquad (A10_I)$$

тогда как все конечные арифметики удовлетворяют аксиоме:

$$\forall x\ (x \leqslant \max), \qquad (A10_F)$$

где «max» – наибольшее представимое целое число.

Аналогично все три способа отношения к переполнению можно различить по выбору одной из следующих аксиом, связанных со значением max + 1:

$\neg\exists x\ (x = \max + 1)$ (строгая интерпретация); $(A11_S)$

$\max + 1 = \max$ (верхняя граница); $(A11_B)$

$\max + 1 = 0$ (модульная арифметика). $(A11_M)$

Выбрав одну из этих аксиом, можно использовать ее при выводе свойств программ; однако эти свойства не обязательно имеют место, если реализация окружения, в котором выполняется программа, не удовлетворяет выбранной аксиоме.

31.3. Выполнение программы

Как уже отмечалось, цель этого исследования – заложить логическое основание для доказательства свойств программы. Одно из самых важных свойств любой программы – выполняет оно предполагаемую функцию или нет. Предполагаемую функцию программы или ее части можно определить, сформулировав общие утверждения относительно значений, которые релевантные переменные должны принимать после выполнения программы. Эти утверждения обычно не назначают конкретных значений каждой переменной, а описывают некоторые общие свойства значений и связи между ними. Для выражения таких утверждений мы пользуемся обычной нотацией математической логики, для большей понятности используются знакомые правила предшествования операторов там, где это возможно.

Во многих случаях корректность результатов программы (или ее части) зависит от значений переменных перед запуском программы. Эти начальные предусловия успешного выполнения можно задать с помощью общего утверждения того же типа, что для описания результатов, полученных после завершения. Для того чтобы сформулировать связь между предусловием (P), программой (Q) и описанием результата ее выполнения (R), мы введем новое обозначение:

$$P\{Q\}R.$$

Это можно интерпретировать следующим образом: «Если утверждение P истинно до запуска программы Q, то утверждение R будет истинно после ее завершения». Если не задано никаких предусловий, то будем писать **true**$\{Q\}R$.

Приведенная ниже трактовка в основных чертах следует работе (Floyd 1967), но применяется к текстам, а не к блок-схемам.

31.3.1. Аксиома присваивания. Присваивание, безусловно, является одной из самых характерных черт программирования цифрового компьютера, наиболее выпукло отличающей его от других разделов математики. Поэтому вызывает удивление, что аксиома, которой мы руководствуемся в рассуждениях о присваивании, не сложнее любой, встречающейся в элементарной логике. Рассмотрим предложение присваивания:

$$x := f,$$

где x – идентификатор простой переменной; f – выражение языка программирования без побочных эффектов, но, возможно, содержащее x.

Теперь любое утверждение $P(x)$, которое справедливо для (значения) x после присваивания, должно было быть также справедливо для (значения) выражения f, вычисленного до присваивания, т. е. со старым значением x. Таким образом, если $P(x)$ должно быть справедливо после присваивания, то $P(f)$ должно быть справедливо до присваивания. Этот факт можно выразить более формально:

D0 аксиома присваивания

$$\vdash P_0\{x := f\}P,$$

где x – идентификатор переменной; f – выражение; P_0 получается из P подстановкой f вместо всех вхождений x.

Можно отметить, что D0 на самом деле является вовсе не аксиомой, а аксиомной схемой, которая описывает бесконечное множество аксиом с общим шаблоном. Этот шаблон описывается в чисто синтаксических терминах, и легко проверить, верно ли, что произвольный конечный текст удовлетворяет этому шаблону и тем самым может считаться аксиомой, допустимой в любой строке доказательства.

31.3.2. Правила следования. Помимо аксиом, для дедукции необходимо по крайней мере одно правило вывода, которое позволяет выводить новые теоремы из одной или более аксиом или ранее доказанных теорем. Правило вывода имеет вид «Если $\vdash X$ и $\vdash Y$, то $\vdash Z$», т. е. если утверждения вида X и Y уже доказаны как теоремы, то Z также считается доказанной как теорема. Простейшее правило вывода говорит, что если выполнение программы Q гарантирует истинность утверждения R, то оно также гарантирует истинность любого утверждения, логически вытекающего из R. Кроме того, если известно, что P является предусловием, при котором программа Q порождает результат R, то таким же предусловием является любое утверждение, из которого логически следует P. Эти правила можно выразить более формально:

D1 правила следования

$$\text{Если } \vdash P\{Q\}R \text{ и } \vdash R \supset S \text{ то } \vdash P\{Q\}S$$

$$\text{Если } \vdash P\{Q\}R \text{ и } \vdash S \supset P \text{ то } \vdash S\{Q\}R$$

31.3.3. Правила композиции. Программа обычно состоит из последовательности предложений, выполняемых друг за другом. Предложения могут быть разделены точкой с запятой или эквивалентным символом, обозначающим композицию процедур: $(Q_1; Q_2; \ldots; Q_n)$. Чтобы избежать громоздких многоточий, мы можем начать всего с двух предложений $(Q_1; Q_2)$, поскольку более длинные последовательности можно восстановить с помощью вложенности, $(Q_1; (Q_2; (\ldots (Q_{n-1}; Q_n) \ldots)))$. Убирание скобок в этой вложенной конструкции можно рассматривать как соглашение, основанное на ассоциативности «оператора ;», – точно так же скобки убираются из арифметического выражения $(t_1 + (t_2 + (\ldots (t_{n-1} + t_n) \ldots)))$. Правило вывода, связанное с композицией, говорит, что если доказанный результат первой части программы совпадает с предусловием, при котором вторая часть программы порождает предполагаемый результат, то и вся программа породит предполагаемый результат, при условии что предусловие первой части удовлетворяется.

Более формально:

D2 правило композиции

$$\text{Если } \vdash P\{Q_1\}R_1 \text{ и } \vdash R_1\{Q_2\}R \text{ то } \vdash P\{(Q_1; Q_2)\}R$$

31.3.4. Правило итерации. Важная особенность компьютера с хранимой программой – способность повторно выполнять некоторую часть программы (S), пока условие (B) не станет ложным. Простой способ выразить такую итерацию – позаимствовать из языка Algol 60 нотацию **while**:

$$\textbf{while } B \textbf{ do } S$$

При выполнении этого предложения компьютер сначала проверяет условие B. Если оно ложно, то S пропускается и выполнение цикла завершается. В противном случае S выполняется и B проверяется снова. Это действие повторяется, пока B не окажется ложным. Рассуждение, ведущее к формулировке правила вывода для итерации, выглядит следующим образом. Предположим, что P – утверждение, которое всегда истинно по завершении S, при условии что оно истинно перед его выполнением. Тогда очевидно, что P будет истинно после любого числа итераций предложения S (даже при отсутствии итераций). Кроме того, известно, что управляющее условие B ложно, когда итерации, наконец, завершаются. Возможна чуть более сильная формулировка ввиду того факта, что можно предположить истинность B в начале выполнения S:

D3 правило итерации

$$\text{Если } \vdash P \wedge B\{S\}P \text{ то } \vdash P\{\textbf{while } B \textbf{ do } S\}\neg B \wedge P$$

31.3.5. Пример. Приведенных выше аксиом достаточно для построения доказательства свойств простых программ, например процедуры нахождения частного q и остатка r от деления x на y. Предполагается, что все переменные определены на множестве неотрицательных целых чисел, удовлетворяющем аксиомам на рис. 31.1. Для простоты мы используем тривиальный, но неэф-

фективный метод последовательных вычитаний. Предлагаемая программа имеет вид:

$$((r := x;\ q := 0);\ \textbf{while}\ y \leqslant r\ \textbf{do}\ (r := r - y;\ q := 1 + q))$$

У этой программы есть важное свойство: после ее завершения мы можем восстановить числитель x, прибавив остаток r к произведению делителя y на частное q (т. е. $x = r + y \times q$). Кроме того, остаток меньше делителя. Эти свойства можно выразить формально:

$$\textbf{true}\{Q\}\neg y \leqslant r \wedge x = r + y \times q$$

где Q – показанная выше программа. Это утверждение выражает необходимое (но не достаточное) условие «корректности» программы.

Формальное доказательство этой теоремы приведено на рисунке ниже [Примечание редактора: рисунок опущен.]. Как и все формальные доказательства, оно чрезмерно громоздкое, и было бы нетрудно ввести систему обозначений, в которых оно стало бы гораздо короче. Еще более действенный метод уменьшения громоздкости формальных доказательств состоит в том, чтобы вывести общие правила построения доказательств из простых правил, принятых за постулаты. Корректность этих общих правил можно было бы установить, продемонстрировав, что любую теорему, которую можно доказать с их помощью, можно было бы доказать (хотя и более громоздко) без них. После того как мощный набор дополнительных правил разработан, «формальное доказательство» сводится к неформальному указанию на то, как его можно было бы построить.

31.4. Общие замечания

В аксиомах и правилах вывода, упоминаемых в этой работе, неявно предполагалось отсутствие побочных эффектов при вычислении выражений и условий. При доказательстве свойств программ на языке, допускающем побочные эффекты, было бы необходимо доказывать их отсутствие в каждом случае еще до применения соответствующей техники доказательства. Если основная цель языка программирования высокого уровня – помогать в построении и верификации корректных программ, то вызывает сомнения, действительно ли использование функциональной нотации для вызова процедур с побочными эффектами является преимуществом. Еще один недостаток приведенных выше аксиом и правил состоит в том, что они не дают никакой основы для доказательства того, что программа успешно завершается. Программа может не завершаться из-за бесконечного цикла либо из-за нарушения налагаемого реализацией лимита, например на диапазон числовых операндов, на объем потребляемой памяти или на время по часам операционной системы. Поэтому нотацию «$P\{Q\}R$» следует интерпретировать как «при условии что программа успешно завершается, R описывает свойства ее результата». Довольно просто модифицировать аксиомы, так чтобы их

нельзя было использовать для предсказания «результатов» незавершающихся программ; но тогда практическое использование аксиом зависело бы от знания многих зависящих от реализации особенностей, например объема памяти и быстродействия компьютера, диапазона представимых чисел и выбора метода обработки переполнения. Оставляя в стороне доказательство отсутствия бесконечных циклов, наверное, было бы лучше доказывать «условную» корректность программы и полагаться на то, что реализация выдаст предупреждение, если будет вынуждена снять программу из-за нарушения какого-то лимита.

Наконец, необходимо перечислить некоторые области, которые не были рассмотрены, например арифметика вещественных чисел, операции над битами и символами, арифметика комплексных чисел, арифметика дробных чисел, массивы, записи, определение оверлеев, файлы, ввод-вывод, объявления, подпрограммы, параметры, рекурсия и параллельное выполнение. Даже характеристика целочисленной арифметики далеко не полна. Мы не предвидим серьезных затруднений при попытке включения этих вопросов, при условии что язык программирования остается простым. Области, представляющие реальные трудности, – это метки и переходы, указатели и именованные параметры. Доказательства корректности программ, в которых эти средства используются, вероятно, будут более сложными, и неудивительно, что это должно быть отражено в сложности соответствующих аксиом.

31.5. Доказательства корректности программы

Самое важное свойство программы – делает ли она то, что ожидает ее пользователь. Если эти ожидания можно строго описать с помощью утверждений о значениях переменных в конце (или в промежуточных точках) выполнения программы, то описанные в этой статье методы можно использовать для доказательства корректности программы, при условии что реализация языка программирования согласуется с аксиомами и правилами, использованными в доказательстве. Сам этот факт также можно было бы установить дедуктивным рассуждением с применением системы аксиом, описывающей логические свойства аппаратных схем. Если корректность программы, ее компилятора и оборудования компьютера установлены с математической строгостью, то можно будет полагаться на результаты программы и предсказывать ее свойства с уверенностью, ограниченной лишь надежностью электроники.

Практика предоставления доказательств для нетривиальных программ не получит широкого распространения, пока не появятся значительно более мощные методы доказательства, и даже тогда эта задача не станет легкой. Но, принимая во внимание возрастающую цену программной ошибки, практические преимущества доказательства правильности программ в конечном итоге перевесят трудности. В настоящее время единственный метод, которым программист может убедить себя в корректности своей программы, – выполнить ее в частных случаях и внести исправления, если результаты не

совпали с ожидаемыми. Обнаружив, что программа, похоже, работает на достаточно широком множестве примеров, он начинает верить, что она работает всегда. Время, потраченное на тестирование программы, часто составляет более половины времени, потраченного на весь проект, и при реалистичной оценке стоимости машинного времени две трети (а то и больше) затрат на проект связано с устранением ошибок на этой стадии.

Цена устранения ошибок, обнаруженных уже после сдачи программы в эксплуатацию, часто оказывается больше, особенно в случае тех элементов программного обеспечения, поставляемого производителем компьютера, для которых значительную часть затрат несет пользователь. И наконец, стоимость ошибки в некоторых типах программ почти не поддается исчислению – потерянный космический корабль, потерпевший крушение самолет, мировая война. Таким образом, внедрение доказательства правильности программ в практику – не только теоретическое устремление в интересах академической респектабельности, но и серьезная рекомендация по снижению затрат, связанных с программными ошибками.

Практика доказательства правильности программ, вероятно, поможет сгладить и некоторые другие проблемы, терзающие мир компьютеров. Например, существует проблема программной документации, которая необходима, во-первых, чтобы проинформировать потенциального пользователя подпрограммы, как ей пользоваться и что она делает, а во-вторых, чтобы помочь при последующем развитии, когда понадобится модифицировать программу так, чтобы она отвечала изменившимся обстоятельствам, или улучшить ее в свете новых знаний. Самый строгий метод формулирования назначения подпрограммы, а равно и условий ее правильного применения – сделать утверждения о значениях переменных до и после ее выполнения. Доказательство правильности этих утверждений можно было бы затем использовать в качестве леммы при доказательстве правильности любой программы, вызывающей данную подпрограмму. Таким образом, в случае большой программы структура целого может быть ясно отражена в структуре доказательства его правильности. Если же возникнет необходимость модифицировать программу, то всегда будет допустимо заменить любую подпрограмму другой, удовлетворяющей тому же критерию правильности. Наконец, вполне вероятно, что при изучении деталей алгоритма доказательство поможет объяснить не только, что происходит, но и почему.

Еще одна проблема, которую позволит решить в той мере, в какой она вообще разрешима, практика доказательства правильности программ, – перенос программ с одной компьютерной архитектуры на другую. Даже когда программа написана на так называемом машинно независимом языке, во многих больших проектах используются некоторые машинно зависимые свойства конкретной реализации, так что при переносе на другую машину возможны неприятные сюрпризы, устранение которых может дорого обойтись. Однако присутствие машинно зависимой особенности всегда можно обнаружить заранее, т. к. попытка доказать правильность такой программы на основе машинно независимых аксиом неизбежно потерпит неудачу. Тогда у программиста будет выбор: либо переформулировать алгоритм машинно независимым образом, возможно, с помощью запросов к окружению, либо,

если это требует слишком больших усилий или неэффективно, осознанно построить машинно зависимую программу и при доказательстве ее правильности опираться на какую-то машинно зависимую аксиому, например на один из вариантов аксиомы A11 (стр. 379). В последнем случае аксиому нужно явно включить в качестве одного из предусловий успешного выполнения программы. Тогда программу можно будет, не теряя уверенности в правильности результата, перенести на любую другую машину, удовлетворяющую той же машинно зависимой аксиоме, но если возникает необходимость перенести ее на неудовлетворяющую реализацию, то все места, где необходимы изменения, будут недвусмысленно аннотированы тем фактом, что в этом месте доказательство зависит от истинности нарушенной машинно зависимой аксиомы.

Таким образом, видится, что практика доказательства правильности программ позволит решить три самые насущные проблемы индустрии программного обеспечения: надежность, документирование и совместимость. Однако в настоящее время доказательство правильности программ представляет трудности даже для программистов высшей квалификации и применимо только к самым простым программам. Как и в других областях, за надежность приходится расплачиваться простотой.

31.6. Формальное определение языка

Высокоуровневые языки, такие как Algol, Fortran или Cobol, обычно планируется реализовывать для широкого круга компьютеров разного размера, конфигурации и конструкции. Серьезной проблемой стало определение этих языков со строгостью, достаточной для обеспечения совместимости всех реализаций. Поскольку цель такой совместимости – возможность замены программ, написанных на одном и том же языке, то один из способов обеспечить ее – настаивать на том, чтобы все реализации языка «удовлетворяли» аксиомам и правилам вывода, лежащим в основе доказательств свойств программ, написанных на этом языке; тогда все основанные на этих доказательствах предсказания будут выполнены во всех случаях, кроме аппаратных сбоев. На самом деле это эквивалентно принятию аксиом и правил вывода в качестве окончательной и авторитетной спецификации семантики языка.

Отвлекаясь от непосредственного и, быть может, даже доказуемого критерия корректности реализации, аксиоматическая техника определения семантики языка программирования выглядит как формальный синтаксис в отчете о Algol 60 в том смысле, что она достаточно проста для того, чтобы ее мог понять как автор реализации, так и более-менее искушенный пользователь языка. Только преодолев этот ширящийся коммуникационный разрыв в одном документе (быть может, даже доказуемо непротиворечивом), можно получить максимальное преимущество от формального определения языка.

Еще одно большое преимущество использования аксиоматического подхода заключается в том, что аксиомы предлагают простую и гибкую технику, позволяющую оставлять некоторые аспекты языка неопределенными, на-

пример диапазон целых чисел, точность представления чисел с плавающей точкой и выбор метода обработки переполнения. Это абсолютно необходимо для целей стандартизации, потому что иначе язык будет невозможно реализовать эффективно на разном оборудовании. Таким образом, стандарт языка программирования должен содержать систему универсально применимых аксиом, а также дополнительные аксиомы, описывающие альтернативы, доступные автору реализации. Пример использования аксиом для этой цели был приведен в § 31.2.

Еще одна цель формального определения языка – помочь при проектировании улучшенных языков программирования. Регулярность, ясность и простота реализации синтаксиса Algol 60, по крайней мере отчасти, могут быть объяснены элегантной формальной техникой его определения. Использование аксиом может дать аналогичные преимущества в области «семантики», поскольку представляется вероятным, что язык, описываемый несколькими «самоочевидными» аксиомами, на основе которых относительно легко строить доказательства, будет предпочтительнее языка со многими невразумительными аксиомами, которые трудно применить для доказательства. Кроме того, аксиомы позволяют проектировщику языка просто и прямо выражать свои общие намерения, не перегружая их массой деталей, обычно сопутствующих алгоритмическим описаниям. Наконец, аксиомы можно формулировать так, что они будут практически независимыми друг от друга, поэтому проектировщик может свободно работать над одной аксиомой или группой аксиом, не опасаясь неожиданных последствий взаимодействия с другими частями языка. <…>

32 Реляционная модель данных для больших совместно используемых банков данных (1970)

Эдгар Ф. Кодд

В академических кругах принято думать, что наука о вычислениях сводится к алгоритмам. Но в бизнесе в центре вычислений всегда были данные. Конечно, эти два взгляда не противоречат друг другу, но мир, ориентированный на данные, выглядит совершенно иначе, чем ориентированный на алгоритмы. Один специалист по базам данных говорил мне, что данные – это океан, глубокий, вечный и таинственный, а алгоритмы – лишь суденышки, бороздящие его поверхность.

Вплоть до 1960-х годов компьютеры использовались в основном для вычисления чисел – либо значений математических функций (например, таблицы, построенные компьютером Mark I Эйкена), либо параметров физических явлений (например, астрономические вычисления или предсказание поведения атомной бомбы). Конечно, для обсчета любого физического явления нужны числовые данные, но у ранних компьютеров было недостаточно памяти для обработки больших объемов данных. Научный интерес к компьютерам диктовался необходимостью выполнять численные расчеты, а также необычайным открытием онтологии алгоритмов Чёрчем и Тьюрингом. Потрясающая работа Тьюринга в Блетчли-Парке по вскрытию шифров

была упражнением на логику и тщательно контролируемый комбинаторный поиск, для которого имелось совсем немного данных (перехваченных зашифрованных сообщений).

Конечно, Ванневар Буш предвидел важность обработки больших объемов данных («Устройства выборки ... скоро смогут перебирать данные со скоростью, гораздо большей нынешних нескольких сотен в минуту», стр. 156). Говард Эйкен и Грейс Хоппер предвидели важность приложений для бизнеса, а корпорация International Business Machines, выросшая из компаний, поставлявших электрические табуляторы для Бюро переписи населения США, заняла доминирующее положение на рынке компьютеров для бизнеса. Начиная с середины 1960-х IBM разрабатывала базу данных для инвентаризации имущества космической программы Apollo. Модель данных в этой системе, первоначально получившая название IMS, выросла из теории графов: у механических деталей есть поддетали, а некоторые поддетали могут использоваться в нескольких разных сборках, поэтому связи между сущностями напоминали ребра, соединяющие вершины ориентированного графа.

Объемы данных, управляемых такими системами, неуклонно росли, и вскоре стало ясно, что способ описания данных и запросов к ним необязательно должен соответствовать структурам для хранения данных в памяти. Когда-то компиляторы позволили отделить формулировку алгоритмов от деталей исполняющего их машинного кода, а теперь возникла необходимость обсуждать данные с формальной строгостью, оставляя самой компьютерной системе решение проблемы оптимальной организации данных на физических устройствах, так чтобы обеспечить эффективность хранения и поиска.

Для Эдгара Фрэнка Кодда (1923–2003) разговаривать о данных удобнее всего было на языке, выведенном из исчисления предикатов. Данные следовало организовать в виде отношений. Отношение – это множество *n*-кортежей, например множество упорядоченных троек или упорядоченных четверок, в которых каждая позиция, или «столбец», содержит данные определенного типа. Отношение можно изобразить в виде таблицы, в которой каждому кортежу соответствует строка, но порядок строк в таблице семантически несуществен, поскольку логически совокупность *n*-кортежей является множеством.

В этой основополагающей статье Кодд изложил основы реляционного представления данных и, самое главное, разработал реляционную алгебру для комбинирования отношений. Большую часть своей карьеры Кодд провел в IBM, и поводом для этой работы стала его неудовлетворенность существовавшими тогда системами баз данных, в которых логическая структура базы данных (а следовательно, и логика программ, обращающихся к базе) была тесно переплетена с внутренним «физическим» представлением. IBM восприняла новаторские идеи Кодда без энтузиазма, быть может, потому что успешная реляционная система баз данных вступила бы в конкуренцию с уже имевшимися продуктами IBM, поэтому Кодду ничего не оставалось, кроме как основать собственную компанию. Экспериментальные реализации реляционной модели базы данных начали появляться в 1970-х годах, это была System R производства IBM и система Ingres Майкла Стоунбрейкера в Беркли. Между 1979 и 1981 годом компании Oracle Corporation, Tandem Computers, Relational Technology Inc. Стоунбрейкера и IBM выпустили коммерческие

реализации, сделав модель стандартом де-факто. Сама модель и связанный с ней язык управления данными SQL теперь вездесущи, а заслуги Кодда были признаны присуждением ему премии Тьюринга.

32.1. Реляционная модель и нормальная форма

32.1.1. Введение. Эта статья посвящается применению элементарной теории отношений к системам, которые обеспечивают совместный доступ к большим банкам форматированных данных. За исключением статьи Чайлдса (Childs 1968), основной областью применения отношений к системам данных являются дедуктивные системы ответов на вопросы. В статье Ливейна и Марона (Levein and Maron 1967) приводятся многочисленные ссылки на работы в этой области.

С другой стороны, в данной статье рассматриваются проблемы *независимости* данных, т. е. независимости прикладных программ и интерактивных действий от увеличения числа типов данных и изменений представления данных, а также проблемы некоторых видов *несогласованности* данных, от которых можно ожидать неприятностей даже в недедуктивных системах.

Реляционное представление (или модель) данных, описываемое в § 32.1, обладает некоторыми преимуществами по отношению к графовой, или сетевой, модели (Bachman 1965; McGee 1969), которая в настоящее время наиболее распространена среди систем, не основанных на логике. Реляционная модель предоставляет средства описания данных на основе только их естественной структуры, т. е. не нуждаясь во введении какой-либо дополнительной структуры для целей машинного представления. Соответственно, эта модель обеспечивает основу языка данных высокого уровня, который поддерживает максимальную независимость между программами, с одной стороны, и машинным представлением и организацией данных – с другой.

Преимуществом реляционного представления является также то, что оно дает надежную основу для решения проблем выводимости, избыточности и согласованности отношений; эти проблемы обсуждаются в § 32.1. С другой стороны, сетевая модель привела к возникновению ряда недоразумений, не последним из которых является ошибочный вывод связей при выводе отношений. <...>

Наконец, реляционное представление дает возможность более четко оценить область действия и логические ограничения существующих систем форматированных данных, а также сравнить достоинства (с логической точки зрения) разных представлений данных в одной системе. Соответствующие примеры приводятся в разных частях этой статьи. Реализация систем, поддерживающих реляционную модель, не обсуждается.

32.1.2. Зависимости данных в существующих системах. Предоставление таблиц описания данных в разрабатываемых сегодня системах является существенным продвижением на пути к независимости данных (IBM 1965b,a; Bleier 1967; IDS 1968). Наличие таких таблиц облегчает изменение некоторых

характеристик представления данных, хранимых в банках данных. Однако набор характеристик представления данных, которые могут быть изменены без нанесения логического ущерба некоторым прикладным программам, продолжает оставаться ограниченным. Далее, модель данных, с которыми работают пользователи, по-прежнему загромождается характеристиками представления; в особенности это касается представлений коллекций данных (а не одиночных элементов данных). Тремя основными видами зависимости данных, которые все еще требуется устранить, являются зависимость порядка, зависимость индексации и зависимость путей доступа. В некоторых системах эти зависимости четко не отделены одна от другой.

32.1.2.1. Зависимость порядка. Элементы данных в банке данных могут храниться разными способами, одни из которых не предполагают наличия какого-либо упорядочения, другие допускают участие каждого элемента только в одном упорядочении, а третьи – в нескольких упорядочениях. Обратим внимание на те существующие системы, в которых требуется или хотя бы допускается хранение элементов данных, по крайней мере в одном полном порядке, тесно связанном с зависимым от аппаратуры порядком адресов. Например, записи в файле, описывающем детали, могут храниться в порядке убывания серийных номеров. В таких системах прикладным программам обычно разрешается предполагать, что порядок представления записей идентичен порядку их хранения (или является его частью). Эти прикладные программы, пользующиеся свойствами упорядоченности файла, скорее всего, не смогут правильно работать, если по какой-то причине потребуется этот порядок изменить. Аналогичные замечания имеют место для случая, когда порядок хранения реализуется посредством указателей.

Нет необходимости выделять в качестве примера какую-либо одну систему, поскольку во всех хорошо известных информационных системах, имеющихся на сегодняшнем рынке, не проводится четкое разделение порядка представления и порядка хранения. Для обеспечения такой независимости требуется решить существенные проблемы реализации.

32.1.2.2. Зависимость индексации. В контексте форматированных данных индекс обычно рассматривается как компонент представления данных, предназначенный исключительно для повышения эффективности. Наличие индексов ускоряет выполнение запросов и операций обновления, но в то же время замедляет операции вставки и удаления. С точки зрения информативности, индекс является избыточным компонентом представления данных. Если в системе используются индексы и она должна хорошо справляться с изменениями характера доступа к банку данных, то, вероятно, потребуется возможность время от времени создавать и уничтожать индексы. Возникает вопрос: могут ли при этом остаться неизменными прикладные программы и интерактивная деятельность?

В современных системах форматированных данных применяются разнообразные подходы к индексации. В TDMS (Bleier 1967) обеспечивается обязательная индексация по всем атрибутам. В текущей версии IMS (IBM 1965b) пользователю предоставляется выбор для каждого файла: между полным отсутствием индексации (иерархическая последовательная организация)

и индексацией только по первичному ключу (иерархическая индексно-последовательная организация). Ни в одном из этих случаев логика пользовательского представления не зависит от обязательно поддерживаемых индексов. Однако в IDS проектировщикам файлов предоставляется возможность выбора индексных атрибутов и добавления индексов в структуру файла с помощью дополнительных цепочек. Прикладные программы, в которых для повышения эффективности используются эти индексные цепочки, должны ссылаться на них по именам. Такие программы не смогут работать правильно, если эти цепочки будут впоследствии удалены.

32.1.2.3. Зависимость путей доступа. Многие существующие системы форматированных данных предоставляют пользователю файлы с древовидной организацией или немного более общие сетевые модели данных. На прикладные программы, предназначенные для работы с этими системами, логически влияют изменения структуры деревьев и сетей. Приведем простой пример.

Предположим, что банк данных содержит информацию о деталях и проектах. Для каждой детали хранится номер детали, название детали, описание детали, количество используемых деталей этого типа и количество заказанных деталей. Для каждого проекта хранится номер проекта, название проекта и описание проекта. Если в проекте используется некоторый тип детали, регистрируется также количество деталей этого типа, предназначенных для данного проекта. Предположим, что система требует, чтобы пользователь или проектировщик файлов объявлял или определял данные в терминах древовидных структур. Тогда для представления упомянутой выше информации годится любая из представленных выше иерархических структур (см. структуры 1–5, рис. 32.1–32.5).

Файл	*Сегмент*	*Поля*
F	ДЕТАЛЬ	номер детали
		наименование детали
		описание детали
		имеющееся количество
		заказанное количество
	ПРОЕКТ	номер проекта
		наименование проекта
		описание проекта
		подтвержденное количество

Рис. 32.1. Структура 1. Проекты подчинены Деталям

Файл	*Сегмент*	*Поля*
F	ПРОЕКТ	номер проекта
		наименование проекта
		описание проекта
	ДЕТАЛЬ	номер детали
		наименование детали
		описание детали
		имеющееся количество
		заказанное количество
		подтвержденное количество

Рис. 32.2. Структура 2. Детали подчинены Проектам

Файл	*Сегмент*	*Поля*
F	ДЕТАЛЬ	номер детали
		наименование детали
		описание детали
		имеющееся количество
		заказанное количество
G	ПРОЕКТ	номер проекта
		наименование проекта
		описание проекта
	ДЕТАЛЬ	номер детали
		подтвержденное количество

Рис. 32.3. Структура 3. Детали и Проекты наравне,
Связь назначения деталей проектам подчинена Проектам

Файл	*Сегмент*	*Поля*
F	ДЕТАЛЬ	номер детали
		наименование детали
		описание детали
		имеющееся количество
		заказанное количество
	ПРОЕКТ	номер проекта
		подтвержденное количество
G	ПРОЕКТ	номер проекта
		наименование проекта
		описание проекта

Рис. 32.4. Структура 4. Детали и Проекты наравне,
Связь назначения деталей проектам подчинена Деталям

Файл	*Сегмент*	*Поля*
F	ДЕТАЛЬ	номер детали
		наименование детали
		описание детали
		имеющееся количество
		заказанное количество
G	ПРОЕКТ	номер проекта
		наименование проекта
		описание проекта
H	ПОДТВЕРЖДЕНИЕ	номер детали
		номер проекта
		подтвержденное количество

Рис. 32.5. Структура 5. Детали, Проекты
и Связь назначения деталей проектам наравне

Теперь рассмотрим задачу распечатки номера детали, названия детали и количества деталей этого типа для каждой детали, используемой в проекте с названием «альфа». Следующие наблюдения могут быть сделаны независимо от того, какая конкретная информационная система с древовидной организацией информации применяется для решения этой задачи. Если для этого разрабатывается программа *P*, ориентированная на использование одной из пяти приведенных выше структур (т. е. *P* не проверяет, какова реальная структура представления данных), то *P* не сможет работать по меньшей мере с тремя из оставшихся структурами. Точнее, если *P* успешно работает со структурой 5, то она не сможет работать со всеми остальными; если *P*

успешно работает со структурой 3 или 4, то она не сможет работать со структурами 1, 2 и 5; если *P* успешно работает со структурой 1 или 2, она не сможет работать работать со структурами 3, 4 и 5. В каждом случае причина проста. Если не проверяется, какая именно структура имеет место, то *P* потерпит неудачу из-за попытки обратиться к несуществующему файлу (в доступных сегодня системах это трактуется как ошибка) или из-за отсутствия попытки обратиться к файлу, содержащему нужную информацию. Читателю, который в этом сомневается, рекомендуется написать пробные программы для решения этой простой задачи.

Поскольку в общем случае непрактично создавать прикладные программы, которые проверяют все древовидные структуры, допускаемые системой, эти программы перестают работать, когда оказывается необходимо изменить структуру.

Системы, предоставляющие пользователям сетевую модель данных, сталкиваются с аналогичными трудностями. И в случае деревьев, и в случае сетей пользователь (или его программа) должен использовать набор путей доступа к данным. Не важно, насколько близко соответствие между этими путями и определяемыми указателями путей в хранимом представлении (в IDS это соответствие является предельно простым, в TDMS – совсем наоборот). В результате независимо от конкретного вида хранимого представления интерактивные операции и программы становятся зависимыми от существования пользовательских путей доступа. Одно из решений – принять политику, согласно которой однажды определенный пользователем путь доступа нельзя вывести из употребления до тех пор, пока существуют программы, использующие этот путь доступа. Такая политика непрактична, поскольку число путей доступа в модели, общей для сообщества пользователей банка данных, в конце концов станет чрезмерно велико.

32.1.3. Реляционное представление данных. Термин *отношение* используется здесь в его общепринятом математическом смысле. Для заданных множеств $S_1, S_2, \ldots, S_n$ (не обязательно различных) *R* является отношением на этих *n* множествах, если представляет собой множество *n*-кортежей, для каждого из которых первый элемент взят из множества S_1, второй – из множества S_2 и т. д. Мы будем называть S_j *j-м доменом R*. Говорят, что такое отношение *R* имеет *степень n*. Отношения степени 1 часто называют *унарными*, степени 2 – *бинарными*, степени 3 – *тернарными* и степени *n* – *n-арными*.

Для наглядности мы будем часто использовать представление отношений в виде массивов, но нужно помнить, что это конкретное представление не является существенной частью излагаемого реляционного представления. Массив, представляющий *n*-арное отношение *R*, обладает следующими свойствами:

1. Каждая строка представляет *n*-кортеж элементов *R*.
2. Порядок строк не существен.
3. Все строки различны.
4. Порядок столбцов существен – он соответствует порядку $S_1, S_2, \ldots, S_n$ доменов, на которых определено *R* (однако обратите внимание на замечания ниже по поводу отношений с упорядоченными и неупорядоченными доменами).

5. Смысл каждого столбца частично выражается посредством его пометки именем соответствующего домена.

В примере на рис. 32.6 показано отношение степени 4 *поставка*. В этом отношении отражаются незавершенные поставки деталей от указанных поставщиков для указанных проектов в указанных количествах.

поставка	(поставщик	деталь	проект	количество)
	1	2	5	17
	1	3	5	23
	2	3	7	9
	2	7	5	4
	4	1	1	12

Рис. 32.6. Отношение степени 4

Можно спросить: если столбцы помечены именами соответствующих доменов, зачем нужна упорядоченность столбцов? Как показывает рис. 32.7, для двух столбцов могут иметься одинаковые заголовки (означающие одинаковые домены), но смысл этих столбцов может быть различным. Показанное отношение называется *компонент*. В этом тернарном отношении два первых домена называются *деталь*, а третий – *количество*. Семантика отношения *компонент* (*x*, *y*, *z*) состоит в том, что деталь *x* является непосредственным компонентом (или составной частью) детали *y*, и для сборки одного экземпляра детали *y* требуется *z* экземпляров детали *x*. Это отношение играет критическую роль в задаче разузлования изделий.

Примечателен тот факт, что некоторые современные системы (главным образом основанные на файлах с древовидной структурой) не в состоянии обеспечить представление данных для отношений, которые имеют два или более одинаковых домена. Примером такой системы является текущая версия IMS/360 (IBM 1965b).

Ко всем данным, находящимся в банке данных, можно относиться как к коллекции изменяющихся во времени отношений. Эти отношения обладают разными степенями. За время существования каждого *n*-арного отношения в него могут вставляться дополнительные *n*-кортежи, удаляться существующие кортежи и изменяться компоненты существующих *n*-кортежей.

Однако во многих коммерческих, правительственных и научных банках данных степени некоторых отношений бывают достаточно велики (степень 30 не является редкостью). Обычно не следует обременять пользователей потребностью помнить порядок доменов в любом отношении (например, порядок *поставщик–деталь–количество* в отношении *поставка*). Поэтому мы предлагаем, чтобы пользователи имели дело не с *отношениями* с упорядоченными доменами, а со *связями*, которые являются их аналогами для неупорядоченных доменов. (Говоря математическим языком, связь – это класс эквивалентности отношений, эквивалентных относительно перестановки доменов (см. § 32.2.1.1).) Для этого домены должны быть однозначно идентифицируемы, по крайней мере в пределах любого данного отношения, без использования их позиции. Таким образом, там, где существует два или

более одинаковых домена, мы требуем, чтобы в каждом случае имена доменов были уточнены отдельным *именем роли*, служащим для указания роли, которую этот домен играет в данном отношении. Например, в отношении *компонент*, приведенном на рис. 32.7, первый домен *деталь* может быть обозначен именем роли *суб*, а второй – именем *супер*, чтобы пользователи могли работать со связью *компонент* и ее доменами – *суб.деталь*, *супер.деталь*, *количество*, не полагаясь на какой-либо порядок этих доменов.

компонент (деталь	деталь	количество)
1	5	9
2	5	7
3	5	2
2	6	12
3	6	3
4	7	1
6	7	1

Рис. 32.7. Отношение
с двумя одинаковыми доменами

Короче говоря, мы предлагаем, чтобы пользователи взаимодействовали с реляционной моделью данных, состоящей из изменяющихся во времени связей (а не с отношениями). Пользователь не должен знать про связь ничего, кроме ее имени и имен ее доменов (с указанными в случае необходимости именами ролей). (Естественно, пользователь производит ввод данных в компьютерную систему и их выборку гораздо более эффективно, если он понимает смысл данных.) И эта информация могла бы предоставляться системой в виде меню (с соблюдением ограничений безопасности и конфиденциальности) по запросу пользователя.

Как правило, существует много альтернативных способов применения реляционной модели в банках данных. Для обсуждения предпочтительного способа (или нормальной формы) мы должны вначале ввести несколько дополнительных понятий (активный домен, первичный ключ, внешний ключ, непростой домен) и установить некоторые связи с терминологией, используемой в настоящее время при программировании информационных систем. Далее в этой статье мы не будем заботиться о различии между отношениями и связями, за исключением тех случаев, когда выгодно явно подчеркнуть это различие.

Рассмотрим пример банка данных, включающего отношения, которые содержат информацию о деталях, проектах и поставщиках. Одно отношение, называемое *деталь*, определено на следующих (и, возможно, на некоторых других) доменах:

1. номер детали;
2. наименование детали;
3. цвет детали;
4. вес детали;
5. количество имеющихся деталей;
6. количество заказанных деталей.

Каждый из этих доменов является, по существу, набором значений, некоторые или все из которых могут содержаться в банке данных в любой момент времени. Если еще можно предположить, что в некоторый момент представлены все цвета деталей, то сделать такое предположение обо всех возможных весах, наименованиях и номерах деталей вряд ли возможно. Мы будем называть множество хранимых в некоторый момент времени значений *активным доменом* в этот момент.

Как правило, один домен (или комбинация доменов) данного отношения содержит значения, позволяющие однозначно идентифицировать каждый элемент (*n*-кортеж) этого отношения. Такой домен (или комбинация доменов) называется *первичным ключом*. В приведенном примере первичным ключом мог бы быть номер детали, в то время как цвет детали – нет. Первичный ключ *неизбыточен*, если он либо является простым доменом (не комбинацией), либо представляет собой такую комбинацию, что ни один из входящих в нее доменов не является лишним для однозначной идентификации каждого элемента. Отношение может обладать более чем одним неизбыточным первичным ключом. В случае когда отношение содержит два или более неизбыточных первичных ключа, произвольно выбирается один из них и называется *Первичным Ключом* этого отношения.

Общее требование к элементам отношения – возможность ссылаться на другие элементы того же других отношений. Ключи обеспечивают средства пользовательского уровня (не единственные) для выражения таких перекрестных ссылок. Мы будем называть домен (или комбинацию доменов) отношения *R внешним ключом*, если он не является Первичным Ключом *R*, но его элементы являются значениями Первичного Ключа некоторого отношения *S* (возможность совпадения *R* и *S* не исключается). В отношении *поставка*, приведенном на рис. 32.6, комбинация *поставщик*, *деталь*, *проект* – Первичный Ключ, в то время как каждый из этих доменов, взятый по отдельности, является внешним ключом.

Ранее наблюдалась сильная тенденция трактовать данные в банке данных как состоящие из двух частей: одна часть состоит из описаний сущностей (например, описаний поставщиков), а другая – из отношений между различными сущностями или типами сущностей (например, отношение *поставка*). Такое различие трудно поддержать, если отношение может содержать внешние ключи. В пользовательской реляционной модели отсутствуют преимущества от такого разделения (тем не менее такие преимущества могут существовать при применении реляционных концепций к машинному представлению пользовательского набора связей).

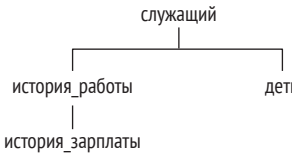
Таким образом, мы обсудили примеры отношений, определенных над простыми доменами – доменами, элементы которых являются атомарными (не поддающимися декомпозиции) значениями. В рамках реляционного подхода могут обсуждаться и неатомарные значения. Мы имеем в виду то, что некоторые домены могут в качестве элементов содержать отношения. Эти отношения могут, в свою очередь, быть определены на непростых доменах и т. д. Например, одним из доменов, на которых определено отношение *служащий*, мог бы быть домен *история зарплаты*. Элемент домена *история зарплаты* – бинарное отношение, определенное на домене *дата*

и домене *зарплата*. Домен *история зарплаты* является множеством всех таких бинарных отношений. В любой момент времени в банке данных существует столько же экземпляров отношения *история зарплаты*, сколько и работников. Напротив, для отношения *служащий* имеется только один экземпляр.

Термины *атрибут* и *повторяющаяся группа* в современной терминологии баз данных – приблизительные аналоги простого и непростого домена соответственно. Основная путаница в существующей терминологии проистекает из отсутствия различия между типом и экземпляром (например, «запись») и между компонентами пользовательской модели данных, с одной стороны, и ее машинным представлением – с другой (опять же «запись» в качестве примера).

32.1.4. Нормальная форма. Отношение, все домены которого являются простыми, может быть представлено двумерным массивом указанного выше вида с однородными столбцами. Для отношения с одним или более непростыми доменами требуются несколько более сложные структуры данных. По этой причине (остальные будут приведены ниже) возможность устранения непростых доменов кажется стоящей дополнительного исследования. В действительности существует очень простая процедура такого устранения, которую мы будем называть нормализацией.

Рассмотрим, например, набор отношений, приведенный на рис. 32.8. *История работы* и *дети* – непростые домены отношения *служащий*. *История зарплаты* – непростой домен отношения *история работы*. В дереве на рис. 32.8 показаны только эти взаимосвязи указанных непростых доменов.



служащий (*номер_служащего*, имя, дата_рождения, история_работы, дети)
история_работы (*дата_приема_на_работу*, название, история_зарплаты)
история_зарплаты (*дата_назначения_зарплаты*, зарплата)
дети (*имя_ребенка*, год_рождения)

Рис. 32.8. Ненормализованное множество

Нормализация выполняется следующим образом. Начиная с отношения, находящегося в корне дерева, взять его Первичный Ключ и каждое непосредственно подчиненное отношение расширить путем вставки домена или комбинации доменов этого Первичного Ключа. Первичный Ключ каждого расширенного таким образом отношения состоит из Первичного Ключа до расширения, дополненного Первичным Ключом, скопированным из родительского отношения. После этого из родительского отношения вычеркиваются все непростые домены, удаляется корневой узел дерева, и та же процедура повторяется для каждого из оставшихся поддеревьев.

Результатом нормализации набора отношений, приведенного на рис. 32.8, является набор отношений, показанный на рис. 32.9. Первичный Ключ каждого отношения выделен курсивом, чтобы показать, как такие ключи расширяются в процессе нормализации.

служащий′ (*номер_служащего*, имя, дата_рождения)
история_работы′ (*номер_служащего*, *дата_приема_на_работу*, название)
история_зарплаты′ (*номер_служащего*, *дата_приема_на_работу*,
дата_назначения_зарплаты, зарплата)
дети′ (*номер_служащего*, *имя_ребенка*, год_рождения)

Рис. 32.9. Нормализованное множество

Чтобы можно было применить описанную нормализацию, ненормализованный набор отношений должен удовлетворять следующим условиям:

1. Граф взаимосвязей непростых доменов является набором деревьев.
2. Ни один первичный ключ не включает в себя непростые домены.

Автор не знает приложений, в которых потребовалось бы ослабление этих условий. Возможно введение операций дальнейшей нормализации. В данной статье это не обсуждается.

Простота представления отношений массивами, осуществимая в случае приведения всех отношений к нормальной форме, предоставляет преимущества не только при хранении, но также при передаче больших объемов данных между системами с принципиально различными представлениями данных. Применение при передаче соответствующим образом упакованного представления в виде массива обеспечило бы следующие преимущества:

1. Передаваемая форма не содержала бы указателей (значениями которых могут быть адреса или смещения).
2. В ней отсутствовали бы все зависимости от схемы хеш-адресации.
3. Она не содержала бы ни индексов, ни упорядоченных списков.

Если реляционная модель пользователя приведена к нормальной форме, имена элементов данных в банке данных могут иметь более простую форму, чем в противном случае. В общем случае имя будет иметь следующую форму:

$$R(g).r.d,$$

где R – имя отношения, g – необязательное имя поколения, r – необязательное имя роли, d – имя домена. Поскольку g необходимо только в случае существования или ожидаемого появления нескольких поколений данного отношения, а r необходимо, только если отношение R имеет два или более доменов с именем d, простой формы $R.d$ часто будет достаточно.

32.1.5. Некоторые лингвистические аспекты. На основе описанной выше реляционной модели данных возможна разработка универсального подъязыка данных, основанного на прикладном исчислении предикатов. Если набор отношений находится в нормальной форме, достаточно исчисления предикатов первого порядка. Такой язык предоставлял бы собой критерий лингвистической мощности для всех остальных предлагаемых языков данных и сам являлся бы кандидатом для встраивания (с соответствующими изменениями синтаксиса) в многочисленные включающие языки (программирования, командно или проблемно ориентированные). Хотя описание подробностей такого языка не является целью данной статьи, укажем его отличительные черты.

Пусть *R* – подъязык данных, а *H* – включающий язык. *R* допускает объявление отношений и их доменов. В каждом объявлении отношения указывается первичный ключ этого отношения. Объявленные отношения добавляются в системный каталог для использования каждым членом сообщества пользователей, обладающим соответствующими правами. В языке *H* возможны объявления, отражающие представление этих отношений в памяти (такие объявления могут быть более изменчивыми). *R* допускает спецификацию выборки любого подмножества данных из банка данных. Действия по запросу такой выборки регулируются ограничениями безопасности.

Универсальность подъязыка данных основана на его декларативности (а не на возможностях вычислений). В больших банках данных каждое подмножество данных имеет очень большое количество возможных (и осмысленных) описаний, даже если предполагается (как мы это и делаем), что существует только конечное множество функций, к которым система может обращаться для задания выбираемых данных. Следовательно, класс выражений ограничения, которые могут быть использованы для спецификации множества, должен иметь выразительную мощность класса правильно построенных формул прикладного исчисления предикатов. Хорошо известно, что для сохранения этой выразительной мощности не обязательна возможность выражения (в соответствующем синтаксисе) каждой формулы выбранного исчисления предикатов. Например, достаточно ограничиться формулами, находящимися в предваренной нормальной форме (Church 1956).

В выражениях ограничения или других частях оператора выборки могут потребоваться арифметические функции. Они могут быть определены в *H* и использованы в *R*.

Определенное таким образом множество может быть просто выбрано в ответ на запрос или сохранено для возможных изменений. Вставки принимают форму добавления новых элементов в объявленные отношения без учета порядка, который может существовать во внутреннем представлении. Удаления, видимые всем пользователям (а не какому-то одному пользователю или группе пользователей), принимают форму исключения элементов из объявленных отношений. Одни удаления и обновления могут порождаться другими, если в *R* объявлены зависимости удаления и обновления между указанными отношениями.

Одно из существенных последствий принятого представления данных для языка, используемого для выборки данных, – принципы именования элементов данных и множеств. Некоторые аспекты этого вопроса были обсуждены в предыдущем разделе. В обычном сетевом представлении пользователь часто обременен придумыванием и использованием бóльшего числа имен отношений, чем это необходимо, т. к. имена ассоциированы с путями (или типами путей), а не с отношениями.

Зная, что хранится некоторое отношение, пользователь будет ожидать, что сможет работать с ним, пользуясь произвольным сочетанием его аргументов как «известными», а остальными аргументами – как «неизвестными», поскольку основная информация (высотой с Эверест) содержится именно в них. Эту особенность системы (отсутствующую во многих современных информационных системах) мы будем называть (логически) *симметричным*

использованием отношения. Естественно, симметричность в производительности не ожидается.

Для поддержки симметричного использования одного бинарного отношения необходимы два ориентированных пути. Для отношения степени n количество путей, которые нужно именовать и контролировать, составляет n факториал ($n!$).

Опять-таки, если принять реляционное представление, в котором каждое n-арное отношение ($n > 2$) должно быть описано пользователем как вложенное выражение, включающее только бинарные отношения (см., например, LEAP System Фельдмана [Feldman and Rovner 1968]), то вместо всего лишь $n + 1$ имен при использовании прямой n-арной нотации, описанной в § 32.1.2, потребуется образовать $2n - 1$ имен. Например, 4-арное отношение *поставка*, приведенное на рис. 32.6 и содержащее 5 имен в n-арной нотации, во вложенной двоичной нотации было бы представлено в виде

P(*поставщик*, *Q*(*деталь*, *R*(*проект*, *количество*)))

и, таким образом, в нем использовалось бы 7 имен.

Другим недостатком такого представления является его асимметрия. Несмотря на то что эта асимметрия не препятствует симметричному использованию, она, без сомнения, сильно затрудняет формулирование некоторых запросов пользователями (попробуйте, например, выразить с применением *Q* и *R* запрос о том, какие детали и в каком количестве используются в данном проекте).

32.1.6. Выразимые, именованные и хранимые отношения. С банком данных связаны два набора отношений: *множество именованных* и *множество выразимых* отношений. Множество именованных отношений составляют те отношения, которые сообщество пользователей может идентифицировать посредством простых имен (или идентификаторов). Отношение *R* входит в множество именованных отношений после его объявления пользователем, обладающим соответствующими правами; оно покидает это множество, когда авторизованный пользователь отменяет объявление *R*.

Множество выразимых отношений включает все отношения, которые могут быть описаны выражениями языка данных. Такие выражения включают простые имена отношений из множества именованных отношений, имена поколений, ролей и доменов, логических связок, кванторов исчисления предикатов и некоторые символы отношений, например > и =. Множество именованных отношений является подмножеством множества выразимых отношений – как правило, очень малым подмножеством.

Поскольку некоторые отношения из множества именованных отношений могут быть не зависящими от времени комбинациями других отношений из этого множества, полезно обсудить связывание с множеством именованных отношений набора утверждений, определяющих эти не зависящие от времени ограничения. Мы отложим дальнейшее обсуждение данного вопроса до введения нескольких операций над отношениями (см. § 32.2).

Одна из основных проблем, с которой сталкивается разработчик системы данных, призванной поддержать реляционную модель для пользователей, –

определение класса поддерживаемых хранимых представлений. В идеале разнообразие допустимых представлений данных должно быть достаточным, чтобы удовлетворить полный спектр требований к производительности для всех установок системы, но не более. Слишком большое разнообразие приведет к ненужным накладным расходам на хранение и многократно повторяющуюся интерпретацию описаний используемых в данный момент структур.

Для любого класса хранимых представлений система управления данными должна предоставить средство преобразования запросов пользователя, выраженных на языке данных реляционной модели, в соответствующие (и эффективные) действия над текущим хранимым представлением. Для языка данных высокого уровня разработка такого средства представляет трудную проблему. Тем не менее эту проблему необходимо решить: по мере увеличения количества пользователей, получающих параллельный доступ к большому банку данных, ответственность за обеспечение эффективного времени отклика и пропускной способности системы переходит от индивидуального пользователя к системе управления данными.

32.2. Избыточность и согласованность

32.2.1. Операции над отношениями. Поскольку отношения являются множествами, к ним применимы все обычные теоретико-множественные операции. Однако результат может не быть отношением; например, объединение бинарного и тернарного отношений не является отношением.

Операции, обсуждаемые ниже, предназначены специально для отношений. Они вводятся по причине их ключевой роли при получении отношений из других отношений. Основной областью применения этих операций являются информационные системы без логического вывода (т. е. системы, которые не обеспечивают механизм логического вывода), хотя они не обязательно перестают быть применимыми при добавлении таких механизмов.

Для большинства пользователей эти операции не представляют интереса. Однако проектировщикам информационных систем и людям, занимающимся управлением банком данных, следует знать их в совершенстве.

32.2.1.1. Перестановка. Бинарное отношение представляется в виде массива с двумя столбцами. В результате перестановки этих столбцов получается обратное отношение. В общем случае при перестановке столбцов n-арного отношения про результирующее отношение говорят, что оно является *перестановкой* заданного отношения. Существует, например, 4! = 24 перестановок отношения *поставка*, приведенного на рис. 32.6, с учетом тождественной перестановки, которая оставляет порядок столбцов неизменным.

Поскольку пользовательская реляционная модель состоит из набора связей (отношений с неупорядоченными доменами), перестановка не является существенной для такой модели, рассматриваемой отдельно. Она тем не менее существенна при рассмотрении хранимых представлений этой модели. В системе, обеспечивающей симметричное использование отношений, множество запросов, на которые может быть получен ответ с использованием

хранимого отношения, совпадает с множеством запросов, на которые может быть получен ответ с использованием любой перестановки этого отношения. Хотя одновременное хранение отношения и некоторой его перестановки не является логически необходимым, соображения производительности могут сделать это целесообразным.

32.2.1.2. Проекция. Предположим, что мы выбрали некоторые столбцы отношения (отбросив остальные) и затем удалили из полученного массива все дубликаты строк. Полученный в результате массив представляет отношение, называемое *проекцией* данного отношения.

Оператор выбора π используется для получения любой требуемой перестановки, проекции или комбинации этих операций. Так, если L – список индексов, $L = i_1, i_2, \ldots, i_k$, и R – n-арное отношение ($n \geqslant k$), то $\pi_L(R)$ – k-арное отношение, j-м столбцом которого является i_j-й столбец R ($i = 1, 2, \ldots, k$) и из которого удалены дубликаты строк. Рассмотрим отношение *поставка* на рис. 32.6. Перестановка проекции этого отношения приведена на рис. 32.10. Заметим, что в этом частном случае проекция имеет меньше n-кортежей, чем исходное отношение.

π_{31}(*поставка*)

(*проект*	*поставщик*)
5	1
5	2
1	4
7	2

Рис. 32.10. Перестановка проекции отношения с рис. 32.6

32.2.1.3. Соединение. Предположим, что даны два бинарных отношения, имеющих некоторый общий домен. При каких условиях можно скомбинировать эти отношения для построения тернарного отношения, сохраняющего всю информацию из данных отношений?

В примере на рис. 32.11 показаны два отношения R и S, которые могут быть соединены без потери информации, а на рис. 32.12 представлен результат соединения R и S. Бинарное отношение R *соединимо* с бинарным отношением S, если существует тернарное отношение U такое, что $\pi_{12}(U) = R$ и $\pi_{23}(U) = S$. Любое такое тернарное отношение называется *соединением* R и S. Если R, S являются бинарными отношениями, такими, что $\pi_2(R) = \pi_1(S)$, то R соединимо с S. Одно из соединений, которое всегда существует в таком случае, – это *естественное соединение*, определяемое как

$$R^*S = \{(a,b,c) : R(a,b) \wedge S(b,c)\},$$

где $R(a, b)$ принимает значение *истина*, если (a, b) является элементом R, и $S(b, c)$ – аналогично. Очевидно, что

$$\pi_{12}(R^*S) = R$$

и

$$\pi_{23}(R^*S) = S.$$

R	(*поставщик*	*деталь*)	*S*	(*деталь*	*проект*)
	1	1		1	1
	2	1		1	2
	2	2		2	1

Рис. 32.11. Два соединимых отношения

*R*S*	(*поставщик*	*деталь*	*проект*)
	1	1	1
	1	1	2
	2	1	1
	2	1	2
	2	2	1

Рис. 32.12. Естественное соединение *R* и *S* (рис. 32.11)

Заметим, что соединение, показанное на рис. 32.12, является естественным соединением отношений *R* и *S*, представленных на рис. 32.11. Другое соединение показано на рис. 32.13.

U	(*поставщик*	*деталь*	*проект*)
	1	1	2
	2	1	1
	2	2	1

Рис. 32.13. Другое соединение *R* и S (рис. 32.11)

При обследовании этих отношений обнаруживается элемент (элемент 1) домена *деталь* (домена, по которому производится соединение), обладающий тем свойством, что он имеет более одного вхождения и в *R*, и в *S*. Этот элемент увеличивает количество возможных соединений. Такой элемент домена соединения называется *точкой неоднозначности* относительно соединения *R* и *S*.

Если $\pi_{21}(R)$ или *S* являются функциями, то при соединении *R* с *S* не может возникнуть точка неоднозначности. В этом случае естественное соединение является единственным соединением *R* с *S*. Заметим, что повторяющееся уточнение «*R* с *S*» необходимо, поскольку *S* может быть соединимым с *R* (как и *R* с *S*), и это соединение должно быть предметом полностью отдельного рассмотрения. На рис. 32.11 ни одно из отношений *R*, $\pi_{21}(R)$, *S*, $\pi_{21}(S)$ не является функцией. <...>

32.2.2. Избыточность <...>

32.2.3. Согласованность <...>

32.2.4. Заключение. В § 32.1 предлагается реляционная модель данных, служащая основой для защиты пользователей систем форматированных данных от потенциально разрушительных изменений представления данных, вызванных увеличением банка данных или изменением нагрузки. Вводится нормальная форма для набора изменяющихся во времени отношений.

В § 32.2 определяются операции над отношениями и два вида избыточности, которые затем применяются к решению проблемы поддержки данных в согласованном состоянии. Такая поддержка может стать очень серьезной проблемой по мере увеличения количества различных типов данных, объединенных в общие банки данных. <...>

33 Управление разработкой больших компьютерных систем (1970)

Уинстон У. Ройс

Уинстон Ройс (1929–1995) по образованию был авиационным инженером, а по опыту работы – инженером-программистом. С 1961 по 1994 год возглавлял проекты в авиакосмической промышленности, сначала в компании TRW, а затем в Lockheed. Как и Фред Брукс (глава 40), он применил свой опыт разработки больших систем к составлению практического руководства по совершенствованию этого процесса.

В этой статье суммированы ключевые идеи, которые он развил в последующих работах. Идеи Ройса не потеряли актуальности и сегодня. Рисунки 33.1 и 33.2 ввели в обиход термин «модель водопада» в области разработки ПО; так называют протокол, требующий, чтобы каждый этап разработки был полностью завершен до перехода к следующему этапу без возможности возврата; при этом любые проблемы, обнаруженные на последующих этапах, должны быть отнесены на счет неудачного исполнения предыдущих. Однако сам Ройс никогда не использовал термин «водопад», а в статье ясно описываются элементы того, что сегодня мы назвали бы «итеративной», или «гибкой» (agile), методикой разработки, в которой ранние реализации считаются предварительными и используются для уточнения проекта.

Ройс придает меньше значения кодированию и выдвигает на первый план проектирование, анализ и документацию; эти идеи созвучны призывам Дейкстры логически и тщательно обдумывать программу, прежде чем

приступать к написанию кода. Ройсу не свойственна математическая строгость Дейкстры; будучи человеком, отвечающим за поставки огромных систем, используемых в особо ответственных авиационных и авиакосмических комплексах, он не мог настаивать на простоте и элегантности как высших ценностях. Его приверженность к тщательному тестированию не произвела бы впечатления на Дейкстру, который совершенно правильно замечал, что тестирование может лишь выявить ошибки, но не доказать правильность. И тем не менее эта статья – кладезь полезной премудрости, она прошла испытание временем, и по сей день к советам автора стоит прислушаться.

33.1. Введение

Я собираюсь описать свой взгляд на управление разработкой больших программных систем. В течение последних девяти лет я был вовлечен в ряд проектов, связанных с планированием миссий космических кораблей, управлением ими и послеполетным анализом. В этих проектах я видел разный уровень успеха в части запуска в эксплуатацию вовремя и в рамках бюджета. В результате этого опыта у меня появились некоторые соображения, которыми я здесь хочу поделиться.

33.2. Функции разработки компьютерных программ

В любой компьютерной разработке, вне зависимости от размера и сложности, можно выделить два шага. Первый – анализ, за которым следует кодирование, как представлено на рис. 33.1. Столь простая концепция реализации – в общем-то, все, что нужно, если задача достаточно мала и если готовый продукт будет использоваться теми, кто его создавал. Кроме того, это те два вида деятельности, за которые заказчики готовы платить, т. к. оба шага включают в себя творческую работу, которая вносит вклад в полезность конечного продукта. Однако если план реализации большого проекта включает в себя лишь эти шаги, то проект обречен на провал. Необходимы многие дополнительные шаги, ни один из которых не влияет столь непосредственно на конечный продукт, как анализ и кодирование, но при этом все они существенно увеличивают расходы. Типичный заказчик предпочтет не оплачивать эти шаги, а типичный исполнитель – не выполнять их. Главная функция руководства состоит в том, чтобы убедить в необходимости дополнительных работ обе группы, а затем обеспечить их выполнение разработчиком.

Более полный подход к разработке представлен на рис. 33.2. Шаги анализа и кодирования по-прежнему на месте, но им предшествуют два уровня анализа требований, между ними вклинивается шаг проектирования, а завершает все шаг тестирования. Эти дополнения рассматриваются отдельно от анализа и кодирования, поскольку отличаются от них по способу выпол-

нения. Их надо планировать иначе и включать в них других специалистов, чтобы использовать ресурсы проекта более рационально.

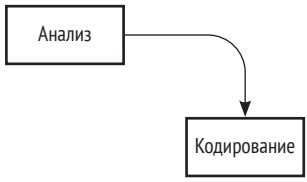


Рис. 33.1. Шаги реализации для небольшой программы, предназначенной для внутреннего использования

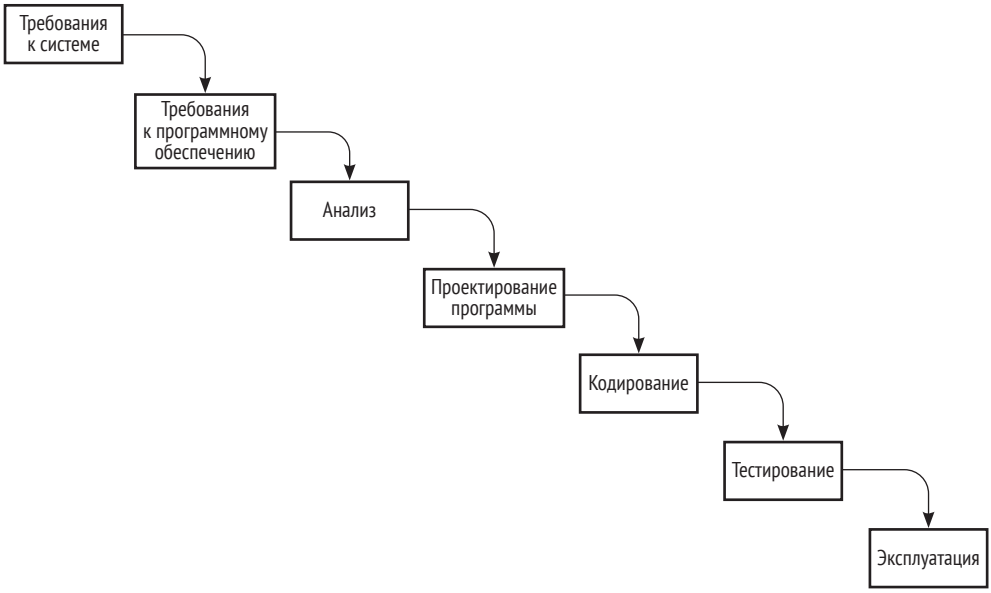


Рис. 33.2. Шаги реализации большой компьютерной программы, поставляемой заказчику

На рис. 33.3 изображены итеративные связи между последовательными шагами на этой схеме. Последовательность шагов основана на следующей идее: завершение одного шага может повлиять на решения, принятые в предыдущем шаге, и, естественно, на следующий шаг, но редко на более удаленные шаги. Достоинство всего этого в том, что по мере конкретизации проекта изменения уже принятых решений остаются в контролируемых пределах. В любой момент после завершения анализа требований существует ясный рубеж, к которому можно вернуться в случае непредвиденных трудностей на этапе проектирования. По сути дела, это второй рубеж обороны, позволяющий в максимальной степени сохранить и использовать результаты работы, проделанной на более ранних стадиях проекта.

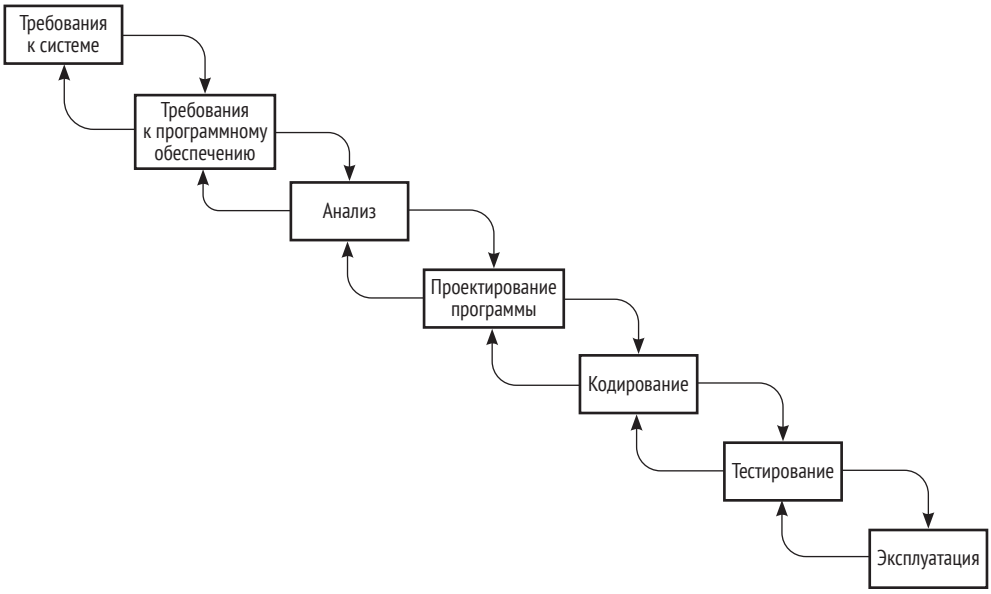


Рис. 33.3. Хочется надеяться, что итеративное взаимодействие между фазами ограничено только непосредственно граничащими шагами

Я верю в описанную концепцию, но реализация, представленная выше, рискованна и чревата провалом. Проблема показана на рис. 33.4. Фаза тестирования, которая расположена в самом конце цикла разработки, – первый момент в проекте, когда становятся видны отклонения по таким параметрам, как скорость работы, объем хранения, ввод-вывод и т. д., от первоначальных результатов анализа. Эти явления невозможно точно проанализировать. Их нельзя получить, например, как результаты решения дифференциальных уравнений в математической физике. Однако если эти параметры выйдут за пределы внешних ограничений, придется переделывать весь проект. Простая заплата на двоичный код или переделка какого-то изолированного участка программы не решит подобные проблемы. Изменения в проекте будут, скорее всего, значительны, т. к. требования, на которых проект основывался и которые являлись стержнем всего, оказались нарушены. Придется или изменять требования, или существенно переделывать проект. В результате проект вернется в исходную точку, и можно ожидать превышения бюджета и (или) сроков на 100 %.

Можно заметить, что мы обошли шаги анализа и кодирования. Естественно, невозможно построить продукт без этих шагов, но, вообще говоря, управлять ими сравнительно легко, и они мало влияют на требования, проектирование и тестирование. У меня есть опыт, когда целые отделы были заняты вычислением орбит, определением положения космического корабля, математической оптимизацией полезного груза и т. д., но когда эти отделы заканчивали свою важную и трудную работу, результирующий программный код сводился к нескольким строкам арифметических операций. Если в процессе сложной работы аналитики допустили ошибку, то для ее исправления

неизменно было достаточно внести мелкое изменение в код, что не имело разрушительных последствий в других местах.

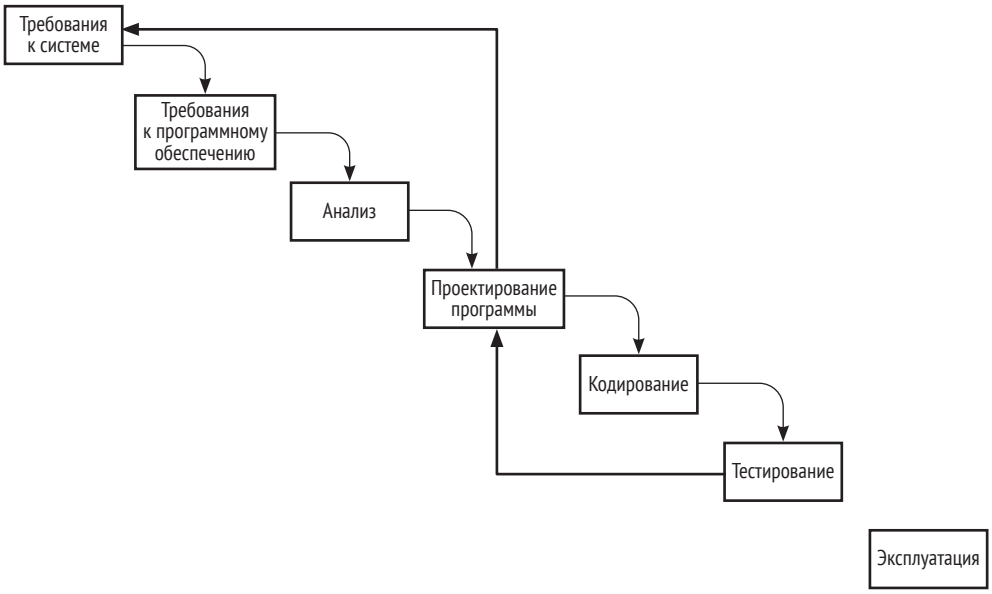


Рис. 33.4. К сожалению, в показанном выше процессе итерации проектирования никогда не ограничиваются соседними шагами

Однако я полагаю, что у представленного подхода имеются фундаментальные достоинства. Далее в этой статье будет представлено пять дополнений, которые необходимо ввести в базовую концепцию, чтобы устранить большинство рисков при разработке.

33.3. Шаг 1: все начинается с проектирования программы

Первый шаг по улучшению процесса представлен на рис. 33.5. Между этапами сбора требований и анализа добавлен шаг предварительного проектирования. Эту процедуру можно критиковать на том основании, что проектировщик программы вынужден заниматься своим делом, находясь в относительном вакууме начальных требований к программе без какого-либо анализа. В результате предварительный проект может оказаться ошибочным, если сравнивать его с конечным проектом, который получился бы после завершения анализа. Замечание верное, но не учитывает важного обстоятельства. Пользуясь этим методом, проектировщик гарантирует, что программа не рухнет из-за проблем со скоростью работы, объемом памяти или неопределенности данных. При переходе к последующей фазе анализа проектировщик должен

передать аналитику информацию об ограничениях на время работы, объем памяти и другие эксплуатационные характеристики таким образом, чтобы тот мог оценить последствия. Если аналитику действительно нужно больше ресурсов для решения уравнения, их придется позаимствовать у других аналитиков. Таким образом, все аналитики и все проектировщики участвуют в осмысленном процессе проектирования, итогом которого является правильное распределение вычислительных ресурсов и памяти. Если суммарных ресурсов не хватает или же первоначальный зачаточный проект был неверен, то это будет замечено на ранней стадии и исправлено на новой итерации требований и предварительного проектирования еще до перехода к окончательному проектированию, кодированию и тестированию. Как реализуется эта процедура? Необходимы следующие шаги:

1. Начинайте процесс проектирования силами проектировщиков программы, а не аналитиков или программистов.
2. Проектируйте, определяйте и распределяйте режимы обработки данных, даже рискуя совершить ошибку. Выделяйте операции обработки, функции, проектируйте базу данных, определяйте способы обработки данных в базе, распределяйте время вычислений, определяйте интерфейсы и режимы работы с операционной системой, описывайте обработку ввода-вывода и предварительно определяйте эксплуатационные процедуры.
3. Составьте обзорный документ, понятный, информативный и актуальный. Каждый сотрудник должен иметь общее представление о системе. По крайней мере один человек должен иметь глубокое представление о системе, которое частично приходит в процессе написания обзорного документа.

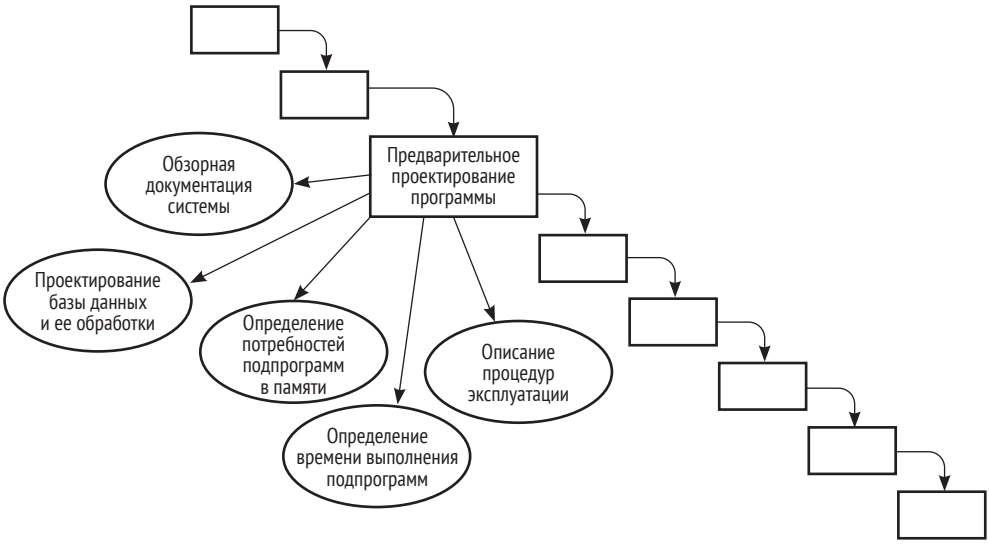


Рис. 33.5. Шаг 1: убедитесь,
что предварительное проектирование закончено до начала анализа

33.4. Шаг 2: документируйте проект

Тут уместно задаться вопросом: «сколько документации?» Мой ответ: «очень много»; во всяком случае, больше, чем большинство программистов, проектировщиков и аналитиков написали бы, если бы были предоставлены сами себе. Первое правило управления проектами разработки программ: бескомпромиссное соблюдение требований по документированию.

Иногда меня приглашают проанализировать состояние других проектов. Мой первый шаг – проверить состояние документации. Если документация в плачевном состоянии, моя рекомендация проста. Заменить руководство проектом. Остановить всю работу, не связанную с документированием. Привести документацию в приемлемое состояние. Управление разработкой программного обеспечения просто невозможно без высокой степени документирования. В качестве примера хочу предложить одну оценку для сравнения. При закупке компьютерного оборудования на 5 миллионов долларов я ожидаю, что спецификация на 30 страницах содержит достаточно деталей для управления процессом закупки. Если же речь идет о закупке программного обеспечения на 5 миллионов долларов, то я бы считал, что документация объемом 1500 страниц – как раз то, что нужно для достижения сравнимой степени контроля.

Зачем так много документации?

1. Каждый проектировщик должен взаимодействовать с проектировщиками соседних подсистем, со своим руководством и, возможно, с заказчиком. Устное общение недостаточно осязаемо, чтобы считаться основательной базой для определения интерфейсов или принятия управленческих решений. Необходимость составить приемлемое описание на бумаге вынуждает проектировщика занять однозначную позицию и предоставить материальные свидетельства завершения работы. Это не дает проектировщику прятаться за словами «работа на 90 % готова» в течение нескольких месяцев.
2. На ранних фазах разработки ПО документация *является* и спецификацией, и проектом. До начала кодирования эти три слова (документация, спецификация, проект) – синонимы. Если плоха документация, плох и проект. Если документации нет, то нет и проекта, а есть лишь люди, которые обдумывают и обсуждают проект, что уже неплохо, но недостаточно.
3. Документация начинает приносить реальную пользу, исчисляемую в деньгах, на последующих этапах процесса разработки – во время тестирования, эксплуатации и перепроектирования. Ценность документации можно описать в терминах трех конкретных осязаемых ситуаций, с которыми сталкивается каждый руководитель проекта.
   a. На этапе тестирования при наличии хорошей документации руководитель может сфокусировать внимание персонала на поиске ошибок в программе. Без хорошей документации каждая ошибка, большая или маленькая, анализируется одним человеком – скорее

всего, именно тем, кто эту ошибку и допустил, потому что он единственный, кто понимает эту часть программы.

b. На этапе эксплуатации при наличии хорошей документации руководитель может научить эксплуатационников, как пользоваться программой лучше и дешевле. Без хорошей документации программу должны будут эксплуатировать те, кто ее создавал. Обычно этих людей эксплуатация интересует мало, и они не смогут работать так же эффективно, как эксплуатационники. В этой связи стоит отметить, что если на этапе эксплуатации произойдет сбой, то в первую очередь всегда винят программу. Чтобы или оправдать программу, или исправить ошибку, необходима четкая и ясная документация.

c. После первоначального периода эксплуатации, когда пора вносить улучшения в программу, хорошая документация позволит эффективно выполнить перепроектирование, модификацию и модернизацию на месте. Если документации нет, то обычно все существующее ПО приходится отправлять на свалку, даже в случае относительно небольших изменений.

На рис. 33.6 показан план документации, соотнесенный с описанными выше шагами. Обратите внимание, что создано шесть документов и что в момент поставки окончательного продукта документы 1, 3, 4, 5 и 6 обновлены и актуальны.

33.5. Шаг 3: проделайте это дважды

После документации следующий критерий успеха связан с оригинальностью продукта. Если программа разрабатывается с нуля, организуйте процесс так, чтобы заказчику была передана не первая, а вторая версия, по крайней мере в том, что касается критических частей проекта и эксплуатации. На рис. 33.7 показано, как это можно сделать с помощью моделирования. Заметим, что это просто весь процесс в миниатюре, выполненный за время, небольшое сравнительно с тем, что необходимо для полной разработки. Природа такого моделирования может сильно зависеть от того, сколько времени занимает весь процесс, и от природы критических проблем, подлежащих моделированию. Если весь проект рассчитан на 30 месяцев, то на разработку пилотной модели можно выделить 10 месяцев. При планировании могут использоваться более-менее формальные методы контроля, процедуры документирования и т. д. Если же общая длительность составляет лишь 12 месяцев, то пилотный проект можно уложить, скажем, в три месяца, чтобы получить рычаг воздействия на магистральную разработку. В этом случае от участников требуется квалификация особого рода. Они должны интуитивно чувствовать анализ, кодирование и проектирование. Они должны «нутром чуять» проблемные места в проекте, моделировать их, моделировать их альтернативы, игнорировать простые аспекты проекта, которые не имеет

смысла дополнительно изучать на этой ранней стадии, и в результате получить программу, не содержащую ошибок. В конечном счете смысл всего этого, как и при моделировании, состоит в том, чтобы точно, а не «на пальцах», изучить вопросы времени выполнения, памяти и т. д. Без такого моделирования руководитель проекта вынужден полагаться на прикидочные оценки. А моделирование хотя бы позволит ему подвергнуть экспериментальной проверке некоторые ключевые гипотезы и свести к минимуму количество мест, где приходится полагаться на экспертное мнение, каковое в области проектирования компьютерных программ (как и при оценке максимального взлетного веса, стоимости завершения работ или ставки на победителя двух последовательных забегов на скачках) неизменно оказывается чрезмерно оптимистичным.

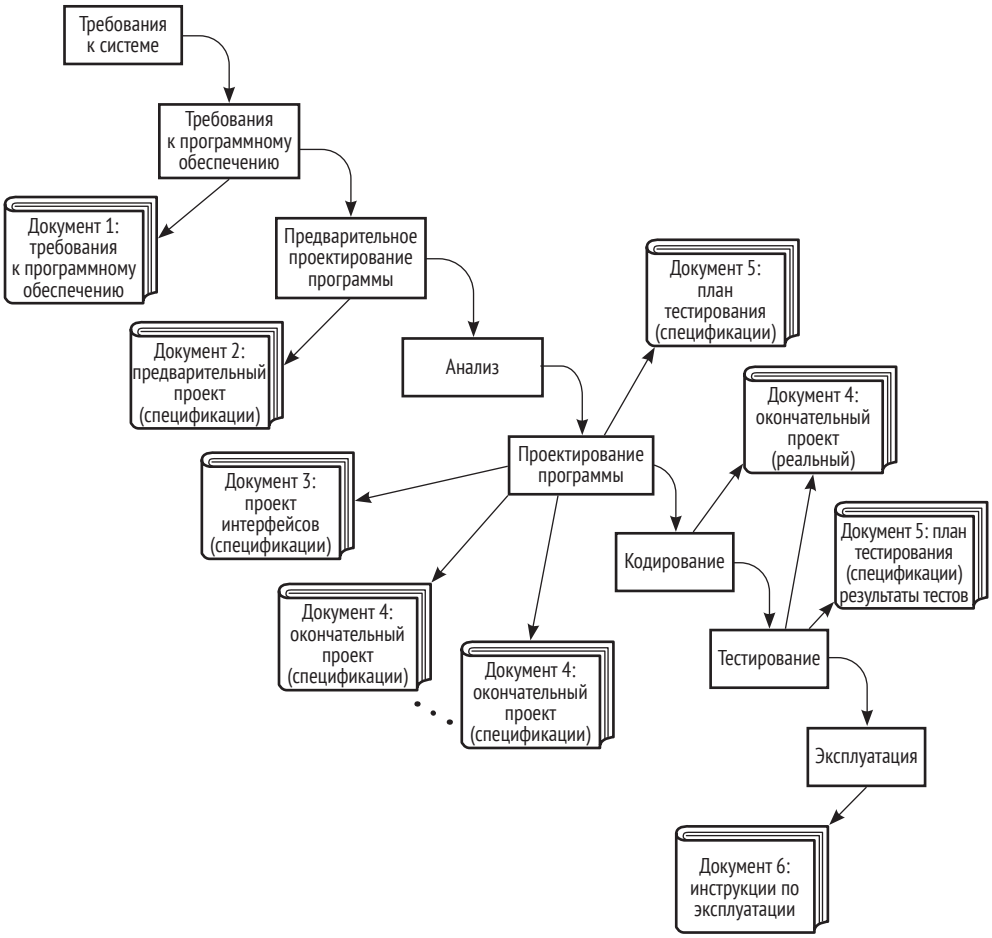


Рис. 33.6. Шаг 2: убедитесь, что документация полна и актуальна. Необходимы по крайней мере шесть разных документов

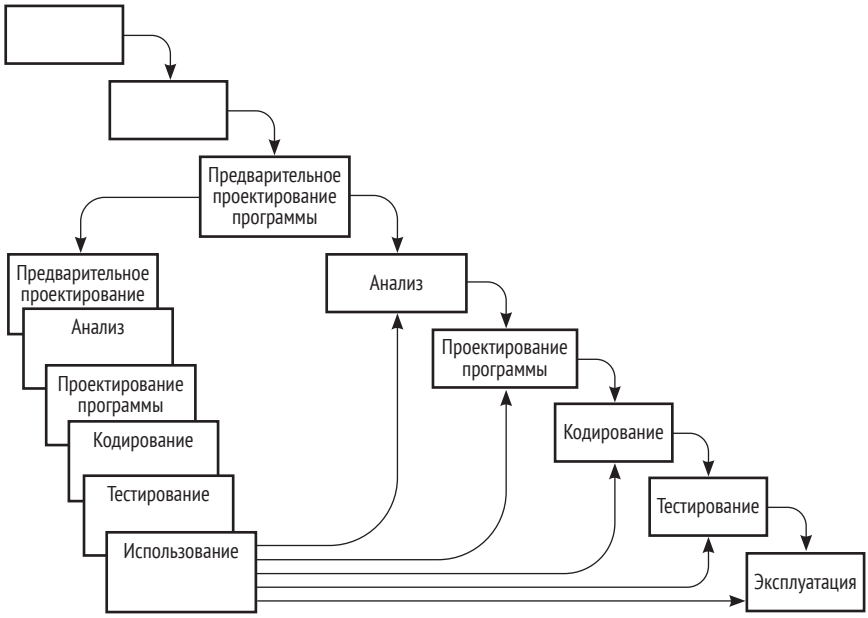


Рис. 33.7. Шаг 3. Попытайтесь проделать работу дважды – результаты первой попытки дают предварительную модель окончательного продукта

33.6. Шаг 4: планируйте, контролируйте и отслеживайте ход тестирования

Несомненно, самым крупным потребителем проектных ресурсов, будь то трудозатраты, машинное время или управленческие таланты, является этап тестирования. На этом этапе риски в терминах бюджета и графика выполнения проекта максимальны. Это последний этап проекта, когда еще к чему-то можно вернуться (а может быть, уже и нельзя).

Предыдущие три рекомендации – проектировать до начала анализа и кодирования, документировать детально и создавать пилотную модель – нацелены на обнаружение и устранение проблем еще до начала этапа тестирования. Однако даже после всего этого предстоит этап тестирования, на котором нужно сделать много важного. На рис. 33.8 показаны некоторые дополнительные аспекты тестирования. При планировании тестирования я рекомендую принять во внимание следующие соображения.

1. Многие части процесса тестирования лучше всего поручить специалистам, которые могли быть и не принимать участия в первоначальном проектировании. Если говорят, что только проектировщик способен качественно выполнить тестирование, потому что только он знает, как программа устроена, то это признак некачественной документации. При наличии хорошей документации вполне возможно задействовать

специалистов по контролю качества ПО, которые, по моему мнению, лучше справятся с тестированием, чем проектировщик.

2. Большинство ошибок тривиальны и могут быть легко обнаружены при визуальном просмотре. Каждое слово анализа и каждый кусочек кода должны быть изучены глазами стороннего специалиста, который не участвовал в проведении этого анализа или написании этого кода, но сможет обнаружить такие вещи, как пропущенный знак минус, пропущенное умножение на 2, переходы по неправильному адресу и т. д., – в этом и состоит суть вычитки аналитической документации и кода. Не используйте для этих целей компьютер – это слишком дорого.
3. Тестируйте каждый логический путь в программе по крайней мере один раз с применением какой-нибудь числовой проверки. Будь я заказчиком, я не принял бы продукт до выполнения и подтверждения такой процедуры. Этот шаг способен выявить большинство ошибок кодирования. Хотя эта процедура тестирования на первый взгляд кажется простой, для большой и сложной программы довольно трудно пройти по всем логическим путям, контролируя входные значения. Кто-то скажет, что это почти невозможно. И тем не менее я настаиваю на том, что каждый логический путь в коде должен быть пройден хотя бы один раз.
4. После того как простые ошибки (а их большинство, и они скрывают ошибки более серьезные) устранены, настает время для контрольной проверки программы. В нужный момент процесса разработки и в руках подходящего специалиста сам компьютер является лучшим инструментом проверки. Ключевое управленческое решение здесь – выбрать момент и назначить ответственного за контрольную проверку.

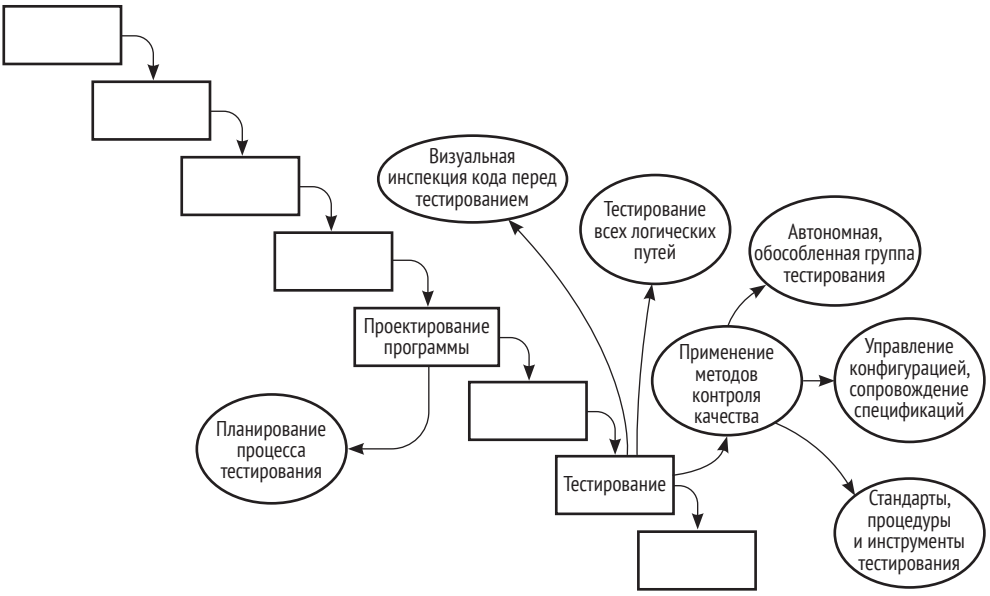


Рис. 33.8. Шаг 4: планируйте, контролируйте и отслеживайте ход тестирования компьютерной программы

33.7. Шаг 5: привлекайте заказчика

Уж не знаю почему, но вопрос о том, что именно должна делать программа, остается предметом широкой интерпретации даже после предварительного согласования. Важно привлекать заказчика, чтобы он формально подтвердил свое согласие на ранних этапах, еще до окончательной поставки. Давать подрядчику полную свободу действий между определением требований и вводом в эксплуатацию – значит нарываться на неприятности. На рис. 33.9 показаны три точки после определения требований, в которых понимание, оценка, мнение и обязательства заказчика могут оказать поддержку процессу разработки.

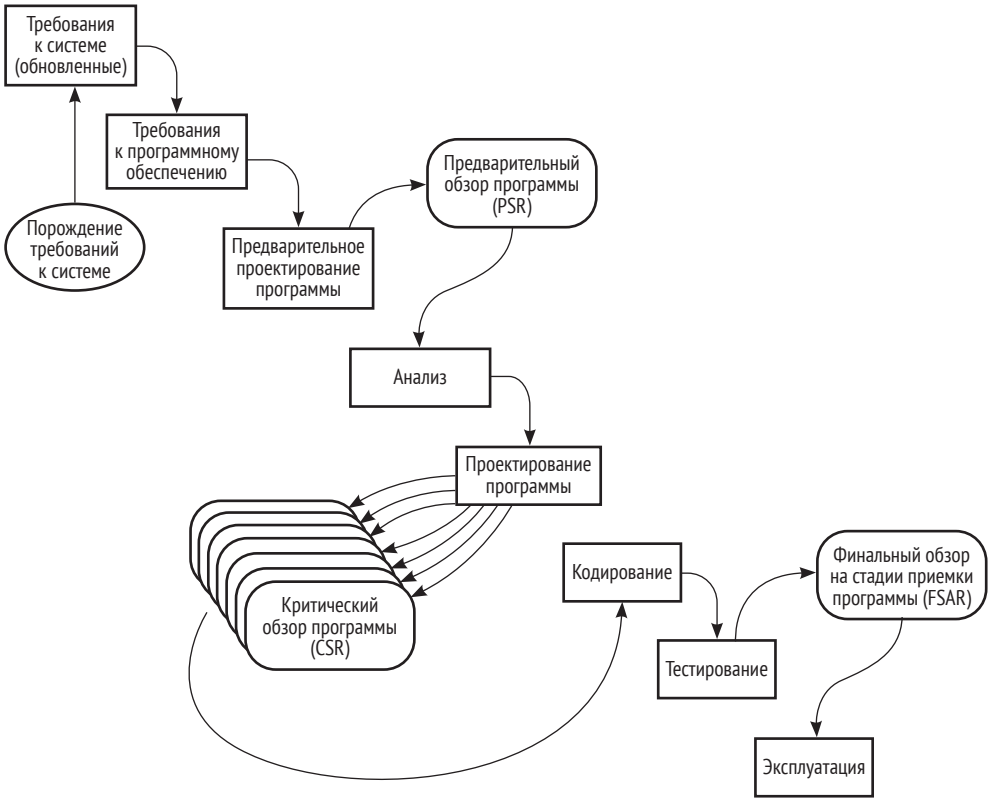


Рис. 33.9. Шаг 5: привлекайте заказчика –
участие заказчика должно быть формальным, глубоким и постоянным

33.8. Итоги

На рис. 33.10 представлены все пять шагов, которые я считаю необходимыми для преобразования рискованного процесса разработки в процесс, итогом

Рис. 33.10. Итог

которого станет желаемый продукт. Я подчеркиваю, что за каждый элемент приходится платить дополнительно. Если бы относительно простой процесс, не включающий описанные здесь пять осложнений, работал успешно, тогда, конечно, эти деньги не стоило бы тратить. Но мой опыт показывает, что более простой метод никогда не приносит успеха в больших проектах, а затраты на переделки значительно превышают те, что понадобились бы на описанный пятишаговый процесс.

34 Сложность процедур вывода теорем (1971)

Стивен А. Кук

Через десять лет после того, как Крускал, не опускаясь до пояснений, назвал «практичным» свой алгоритм нахождения остовного дерева (глава 17), Джек Эмондс, рассматривая максимальные паросочетания в графах, сделал паузу, чтобы провести различие между алгоритмами с полиномиальным и экспоненциальным временем выполнения (как мы их теперь называем). В «отвлечении» в начале статьи (Edmonds 1965) читаем: «Необходимо пояснение по поводу использования слов “эффективный алгоритм”. <…> Для практических целей разница между алгебраическим и экспоненциальным порядком зачастую оказывается более важной, чем разница между финитным и нефинитным». Примерно в то же время математик Алан Кобхэм (Cobham 1965) описал независимый от машины и модели класс, названный им $\mathcal{L}$, функций, вычислимых за время, «ограниченное полиномом от длин участвующих в их определении чисел».

В своей основополагающей работе Стивен Кук (родился в 1939 году) определяет класс $\mathcal{L}_*$, который Эдмондс упоминает в связи с задачами, имеющими решения «алгебраического порядка», и который мы сегодня называем $\mathcal{P}$, или PTIME. То есть $\mathcal{L}_*$ или $\mathcal{P}$ – теоретико-множественный аналог класса функций $\mathcal{L}$ Кобхэма. Затем Кук дает важнейшее определение полиномиальной сводимости задач и доказывает, что все задачи, допускаемые за полиномиальное время недетерминированной машиной Тьюринга, полиномиально сводимы к набору выполнимых формул исчисления высказываний в конъюнктивной нормальной форме. (Обозначение $\mathcal{NP}$ этого класса было принято вскоре после публикации данной статьи – см. главу 36 и работу Knuth (1974b). Кроме того, теорема формулируется в терминах тавтологичности, т. е. класса со-$\mathcal{NP}$, а не – как привычно нам сегодня – выполнимости, т. е. класса $\mathcal{NP}$.)

Кук получил докторскую степень по математике в Гарварде в 1966 году за работу по сложности умножения и других математических функций, выполненную под руководством логика Ван Хао. Он был принят в штат математического факультета в Беркли на должность доцента. По какому-то нелепому недоразумению, иначе не назовешь, в 1970 году Куку было отказано в пожизненном контракте в Беркли, и он перешел в Торонтский университет. Эта короткая работа была представлена на крупной конференции по теоретической информатике, в ней он продемонстрировал, что ограниченные во времени недетерминированные вычисления можно кратко описать булевыми формулами, а значит, из существования полиномиального алгоритма булевой выполнимости сразу следовало бы существование полиномиальных алгоритмов для любых задач класса $\mathcal{NP}$. Само доказательство простое – блестящим озарением было описание вычислений формулами, далекое эхо работы Тьюринга (Turing 1936, здесь стр. 92). Результат Кука дал старт все еще незавершенному поиску ответа на вопрос о справедливости равенства $\mathcal{P} = \mathcal{NP}$ и сотням других исследований по теории сложности.

У вопроса о равенстве $\mathcal{P} = \mathcal{NP}$ есть еще один важный исторический аспект. Работая независимо и в относительной изоляции, советский математик Леонид Левин (родился в 1948 году) определил тот же самый класс «задач универсального перебора» (Levin 1973), которые стали больше известны под названием $\mathcal{NP}$-полных задач. Открытия Кука и Левина были сделаны почти одновременно, хотя публикация работы Левина задержалась. Существование $\mathcal{NP}$-полных задач теперь называется теоремой Кука–Левина. Левин эмигрировал в США в 1978 году и теперь является профессором Бостонского университета.

34.0. Краткое описание

Показано, что любую задачу распознавания, решаемую на недетерминированной машине Тьюринга за полиномиально ограниченное время, можно «свести» к задаче определения того, является ли заданная формула исчисления высказывания тавтологией. Здесь слово «свести» означает, грубо говоря, что первую задачу можно детерминированно решить за полиномиальное время, при условии что имеется оракул для решения второй. На основе этого понятия сводимости определяются полиномиальные степени трудности и показано, что задача определения тавтологичности имеет ту же полиномиальную степень, что и задача установления изоморфности одного из двух заданных графов некоторому подграфу второго. Обсуждаются и другие примеры. Вводится и обсуждается метод измерения сложности процедур доказательства для исчисления предикатов.

В этой статье под *множеством слов* понимается множество слов над некоторым фиксированным, большим, конечным алфавитом Σ. Этот алфавит настолько велик, что включает символы всех описанных здесь множеств. Все машины Тьюринга рассматриваются как детерминированные распознающие устройства, если явно не оговорено противное.

34.1. Тавтологии и полиномиальная сводимость

Зафиксируем формализм для исчисления высказываний, в котором формулы записываются в виде слов над алфавитом Σ. Поскольку мы будем требовать бесконечно большого числа пропозициональных символов (атомов), каждый такой символ будет состоять из элемента Σ, за которым следует число в двоичной записи, позволяющее отличить этот символ от всех остальных. Таким образом, формула длины n может насчитывать всего приблизительно $n/\log n$ различных символов функций и предикатов. Логическими связками являются ∧ (и), ∨ (или) и ¬ (не).

Множество тавтологий (обозначаемой {тавтологии}) – это некоторое рекурсивное множество слов над этим алфавитом, и нас интересует хорошая нижняя граница времени его распознавания. Здесь мы не предлагаем такой нижней границы, но теорема 1 свидетельствует о том, что множество {тавтологии} трудно для распознавания, поскольку многие кажущиеся трудными задачи могут быть сведены к установлению тавтологичности. Под словом «сведены» мы понимаем, грубо говоря, что если бы тавтологичность можно было установить мгновенно (с помощью «оракула»), то эти задачи можно было бы решить за полиномиальное время. Чтобы сделать это понятие точным, мы введем в рассмотрение спрашивающие машины, похожие на машины Тьюринга с оракулами, описанные в работе Kreider and Ritchie (1964).

Спрашивающей машиной называется многоленточная машина Тьюринга с одной выделенной лентой, называемой *лентой вопросов*, и тремя выделенными состояниями, называемыми *спрашивающим состоянием*, *да-состоянием* и *нет-состоянием* соответственно. Если M – спрашивающая машина, а T – множество слов, то T-вычислением M называется такое вычисление M, для которого M в начале вычисления находится в своем начальном состоянии и входное слово w записано на ее входной ленте, и всякий раз, как M попадает в спрашивающее состояние и на ленте вопросов записано слово u, следующим состоянием M будет да-состояние, если $u \in T$, и нет-состояние, если $u \notin$ T. Мы считаем, что «оракул» знает T и переводит M в да-состояние или нет-состояние.

Определение. Множество слов S называется *P-сводимым* (P означает «полиномиально») к множеству слов T, если существует некоторая спрашивающая машина M и полином $Q(n)$ такие, что для каждой входной строки w T-вычисление на машине M с входом w останавливается не позднее, чем после $Q(|w|)$ шагов ($|w|$ – длина w), и оказывается в допускающем состоянии тогда и только тогда, когда $w \in S$.

Нетрудно видеть, что P-сводимость – транзитивное отношение. Таким образом, отношение E на множествах строк, определенное так, что $(S, T) \in E$ тогда и только тогда, когда S и T P-сводимы друг к другу, является отношением эквивалентности. Класс эквивалентности, содержащий множество S, будем обозначать deg(S) (полиномиальная степень трудности S).

Определение. Будем обозначать $\mathcal{L}_*$ класс deg({0}), где 0 обозначает нулевую функцию.

Таким образом, $\mathcal{L}_*$ – класс множеств, распознаваемых за полиномиальное время. $\mathcal{L}_*$ обсуждался в работе (Cook 1971a, стр. 5) и является словарным аналогом класса $\mathcal{L}$ функций, который рассматривался Кобхэмом (Cobham 1965).

Определим теперь следующие специальные множества слов.

1. *Задача о подграфе* состоит в том, чтобы для двух заданных конечных неориентированных графов установить, является ли первый изоморфным подграфу второго. Граф G можно представить словом $\overline{G}$ над алфавитом {0, 1, $*$}, выписывая последовательно строки его матрицы смежности, отделенные друг от друга символами $*$. Обозначим {пара подграфов} множество строк $\overline{G_1}{*}{*}\overline{G_2}$ таких, что G_1 изоморфен подграфу G_2.
2. *Задачу об изоморфизме графов* можно представить множеством {пара изоморфных графов} всех слов $\overline{G_1}{*}{*}\overline{G_2}$ таких, что G_1 изоморфен G_2.
3. Множество {простые числа} – это множество всех простых чисел, записанных в двоичном виде.
4. Множество {ДНФ-тавтологии} – это множество слов, представляющих тавтологии в дизъюнктивной нормальной форме.
5. Множество D_3 состоит из тех тавтологий в дизъюнктивной нормальной форме, в которых каждый дизъюнктивный член состоит не более чем из трех конъюнктивных (каждый из которых является атомом или отрицанием атома).

Теорема 1. Если множество строк S распознается какой-либо недетерминированной машиной Тьюринга за полиномиальное время, то S P-сводимо к множеству {ДНФ-тавтологии}.

Следствие. Каждое из множеств 1–5 P-сводимо к множеству {ДНФ-тавтологии}.

Это следует из того, что каждое из этих множеств или его дополнение распознается некоторой недетерминированной машиной Тьюринга за полиномиальное время.

Доказательство теоремы 1. Предположим, что недетерминированная машина Тьюринга M распознает множество слов S за время $Q(n)$, где $Q(n)$ – полином. По заданному входному слову w машины M мы построим формулу исчисления высказываний $A(w)$ в конъюнктивной нормальной форме такую, что $A(w)$ выполнима тогда и только тогда, когда M принимает w. Таким образом, $\neg A(w)$ легко привести к дизъюнктивной нормальной форме (с помощью правил де Моргана), и $\neg A(w)$ является тавтологией тогда и только тогда, когда $w \notin S$. Так как все построение можно выполнить за время, полиномиально зависящее от $|w|$ (длины w), теорема доказана.

Мы также можем предположить, что машина Тьюринга M имеет только одну ленту, бесконечно продолжающуюся вправо, но имеющую самую левую ячейку. Занумеруем ячейки числами 1, 2, … слева направо. Зафиксируем вход w длины n в машину M и предположим, что $w \in S$. Тогда существует вычисление машины M с входом w, завершающееся в допускающем состоянии не более чем за $T = Q(n)$ шагов. Формула $A(w)$ будет построена из многих разных пропозициональных символов, чья предполагаемая семантика, перечисленная ниже, ссылается на такое вычисление.

Предположим, что алфавит ленты машины M представляет собой множество $\{\sigma_1, \ldots, \sigma_l\}$, а множество ее состояний – $\{q_1, \ldots, q_r\}$. Заметим, что поскольку вычисление состоит не более чем из $T = Q(n)$ шагов, никакая ячейка с номером, большим T, не может обозреваться головкой.

Пропозициональные символы:

$P^i_{s,t}$ для $1 \leqslant i \leqslant l$, $1 \leqslant s, t \leqslant T$. $P^i_{s,t}$ истинно тогда и только тогда, когда s-я ячейка ленты на шаге t содержит символ σ_i.

Q^i_t для $1 \leqslant i \leqslant r$, $1 \leqslant t \leqslant T$. Q^i_t истинно тогда и только тогда, когда на шаге t машина находится в состоянии q_i.

$S_{s,t}$ для $1 \leqslant s, t \leqslant T$ истинно тогда и только тогда, когда на шаге t головка ленты обозревает ячейку s.

Формула $A(w)$ является конъюнкцией $B \wedge C \wedge D \wedge E \wedge F \wedge G \wedge H \wedge I$, построенной, как описано ниже. Заметим, что $A(w)$ – конъюнктивная нормальная форма.

B утверждает, что на каждом шаге t обозревается одна и только одна ячейка. B является конъюнкцией $B_1 \wedge B_2 \wedge \ldots \wedge B_T$, где B_t утверждает, что на шаге t обозревается одна и только одна ячейка:

$$B_t = (S_{1,t} \vee S_{2,t} \vee \cdots \vee S_{T,t}) \wedge \left(\bigwedge_{1 \leq i < j \leq T} (\neg S_{i,t} \vee \neg S_{j,t}) \right).$$

Для $1 \leqslant s \leqslant T$ и $1 \leqslant t \leqslant T$ $C_{s,t}$ утверждает, что на шаге t в ячейке s находится один и только один символ. C есть конъюнкция всех таких $C_{s,t}$.

D утверждает, что для каждого t имеется одно и только одно состояние.

E утверждает, что выполнены начальные условия:

$$E = Q^0_1 \wedge S_{1,1} \wedge P^{i_1}_{1,1} \wedge P^{i_2}_{2,2} \wedge \cdots \wedge P^{i_n}_{n,1} \wedge P^1_{n+1,1} \wedge \cdots \wedge P^1_{T,1},$$

где $w = \sigma_{i_1} \ldots \sigma_{i_n}$, q_0 – начальное состояние, а σ_1 – пустой символ.

F, G и H утверждают, что для каждого t значения P, Q и S обновляются надлежащим образом. Например, G есть конъюнкция $G^t_{i,j}$ по всем t, i, j, где $G^t_{i,j}$ утверждает, что если на шаге t машина находится в состоянии q_i и обозревает символ σ_j, то на шаге $t + 1$ машина находится в состоянии q_k, где q_k – состояние, определяемое функцией переходов машины M. [Примечание редактора: здесь в варианте оригинальной статьи, выложенном на сайте Кука, имеется рукописное примечание «(или состояния: недетерминированность)». Действительно, представленное выше построение правильно, только если машина детерминированная; для недетерминированных машин оно должно выглядеть несколько иначе.]

$$G^t_{i,j} = \bigwedge_{s=1}^{T} (\neg Q^i_t \vee \neg S_{s,t} \vee \neg P^j_{s,t} \vee Q^k_{t+1}).$$

Наконец, формула I утверждает, что на некотором шаге машина достигает допускающего состояния. Машину M следует модифицировать, заставив ее продолжать вычисления каким-либо тривиальным способом после достижения допускающего состояния, так чтобы формула $A(w)$ была выполнимой.

Теперь совсем просто проверить, что $A(w)$ обладает всеми свойствами, перечисленными в первом абзаце доказательства.

Теорема 2. Следующие множества попарно P-сводимы друг к другу (а потому их полиномиальные степени трудности одинаковы):{тавтологии}, {ДНФ-тавтологии}, D_3, {пара подграфов}.

Замечание. Мы не смогли включить в этот список множества {простые числа} и {пара изоморфных графов}. Для доказательства того, что {тавтологии} P-сводимо к {простые числа}, по-видимому, необходимы глубокие результаты из теории чисел, а демонстрация P-сводимости {тавтологии} к {пара изоморфных графов}, вероятно, опровергнет гипотезу Корнеля (Corneil and Gotlieb 1970), из которой он выводит, что задача об изоморфизме графов может быть решена за полиномиальное время.

Кстати говоря, из процедуры Дэвиса–Патнэма (Davis and Putnam 1960) нетрудно видеть, что множество D_2, состоящее из всех ДНФ-тавтологий в дизъюнктивной нормальной форме, содержащих не более двух конъюнктивных членов в каждом дизъюнктивном, принадлежит $\mathcal{L}_*$. Поэтому D_2 нельзя включить в список из теоремы 2 (если только все множества в этом списке не принадлежат $\mathcal{L}_*$).

Доказательство теоремы 2. В силу следствия из теоремы 1, каждое множество P-сводимо к {ДНФ-тавтологии}. Поскольку {ДНФ-тавтологии}, очевидно, P-сводимо к {тавтологии}, остается показать, что {ДНФ-тавтологии} P-сводимо к D_3, а D_3 P-сводимо к {пара подграфов}.

Чтобы показать, что {ДНФ-тавтологии} P-сводимо к D_3, предположим, что A – формула исчисления высказываний в дизъюнктивной нормальной форме. Пусть $A = B_1 \vee B_2 \vee \ldots \vee B_k$, где $B_1 = R_1 \wedge \ldots \wedge R_s$ и каждое R_i – атом или отрицание атома и $s > 3$. Тогда A является тавтологией тогда и только тогда, когда A' – тавтология, где

$$A' = P \wedge R_3 \wedge \ldots \wedge R_s \vee \neg P \wedge R_1 \wedge R_2 \vee B_2 \vee \ldots \vee B_k$$

и P – новый атом. Так как мы уменьшили количество конъюнктивных членов в B_1, этот процесс можно повторять, пока в итоге не будет получена формула, содержащая не более трех конъюнктивных членов в одном дизъюнктивном. Ясно, что весь процесс ограничен по времени некоторым полиномом от длины A.

Остается показать, что D_3 P-сводится к {пара подграфов}. Пусть A – формула в дизъюнктивной нормальной форме, в которой каждая конъюнкция имеет длину 3, т. е. $A = C_1 \vee \ldots \vee C_k$, где $C_i = R_{i1} \wedge R_{i2} \wedge R_{i3}$, и каждое R_{ij} – атом или отрицание атома. Пусть теперь G_1 – полный граф с вершинами $\{v_1, v_2, \ldots, v_k\}$, а G_2 – граф с вершинами $\{u_{ij}\}$, $1 \leqslant i \leqslant k$, $1 \leqslant j \leqslant 3$ такой, что u_{ij} соединена ребром с u_{rs} тогда и только тогда, когда $i \neq r$ и оба литерала (R_{ij}, R_{rs}) не образуют противоположную пару (т. е. не имеют ни вида $(P, \neg P)$, ни вида $(\neg P, P)$). Таким образом, для формулы A можно подобрать значения, при которых она становится ложной тогда и только тогда, когда существует гомоморфизм графов $\phi : G_1 \to G_2$ такой, что для любого i $\phi(v_i) = u_{ij}$ для некоторого j. (Гомоморфизм указывает для каждого i, какой из атомов R_{i1}, R_{i2}, R_{i3} следует считать ложным, а избирательное отсутствие ребер в G_2 гарантирует, что истинностные значения для всех атомов совместимы.)

Чтобы гарантировать, что взаимно однозначный гомоморфизм $\phi : G_1 \to G_2$ обладает тем свойством, что для любого i $\phi(v_i) = u_{ij}$ для некоторого j, изменим G_1 и G_2 следующим образом. Выберем графы $H_1, H_2, \ldots, H_k$, отличающиеся друг от друга настолько сильно, что если G_1' образован из G_1 путем присоединения H_i к v_i, $1 \leqslant i \leqslant k$ и G_2' образован из G_2 путем присоединения H_i к каждой из вершин u_{i1}, u_{i2} и u_{i3}, $1 \leqslant i \leqslant k$, то любой взаимно однозначный гомоморфизм $\phi : G_1' \to G_2'$ обладает сформулированным выше свойством. Нетрудно показать, что такое построение можно выполнить за полиномиальное время. Тогда G_1' можно вложить в G_2' тогда и только тогда, когда $A \notin D_3$. Это завершает доказательство теоремы 2.

34.2. Обсуждение

Теорема 1 и следствие из нее убедительно свидетельствуют о том, что не так-то просто определить, является ли заданная формула исчисления высказываний тавтологией, даже если она выражена в дизъюнктивной нормальной форме. Совместно теоремы 1 и 2 позволяют предположить, что бесполезно искать полиномиальную процедуру решения задачи о подграфе, потому что успех означал бы, что полиномиальные процедуры существуют и для многих других, по-видимому трудноразрешимых, задач. Конечно, то же самое относится к любой комбинаторной задаче, к которой P-сводима задача о тавтологии.

Более того, эти теоремы наводят на мысль, что {тавтологии} – хороший кандидат на роль интересного множества, которое не принадлежит $\mathcal{L}_*$, и я полагаю, что стоит приложить значительные усилия к доказательству этой гипотезы. Такое доказательство стало бы большим успехом в теории сложности.

Ввиду кажущейся несомненной сложности множества {ДНФ-тавтологии} интересно исследовать процедуру Дэвиса–Патнэма (Davis and Putnam 1960). Эта процедура предназначалась для определения того, является ли выполнимой заданная формула в конъюнктивной нормальной форме, но, конечно, «двойственная» процедура устанавливает, является ли тавтологией заданная формула в дизъюнктивной нормальной форме. Мне пока не удалось найти серию примеров, показывающую, что эта процедура (трактуемая со всей снисходительностью, чтобы избежать некоторых подводных камней) должна требовать более чем полиномиального времени. Равно как не удалось найти интересную верхнюю границу требуемого времени.

Если представлять словами натуральные числа (или k-кортежи натуральных чисел), используя m-адическую или иную подходящую нотацию, то замечания, высказанные в предыдущих разделах, можно будет применить к множествам чисел (или к k-местным отношениям на числах). Нетрудно видеть, что множество отношений, принимаемых за полиномиальное время некоторой недетерминированной машиной Тьюринга, – это в точности множество $\mathcal{L}^+$ отношений вида

$$(\exists y \leqslant g_k(\bar{x}))\ R(\bar{x}, y), \tag{34.1}$$

где $g_k(x) = 2^{(l(\max \bar{x}))^k}$, $l(z)$ – диадическая длина z, а $R(\bar{x}, y)$ – отношение из $\mathcal{L}_*$. ($\mathcal{L}^+$ является классом расширенных положительных рудиментарных отношений Беннета [Bennett, 1962].) Если бы мы убрали границу квантора в формуле (34.1), то класс $\mathcal{L}^+$ превратился бы в класс всех рекурсивно перечислимых множеств. Таким образом, если $\mathcal{L}^+$ рассматривать как аналог класса рекурсивно перечислимых множеств, то определение тавтологичности будет аналогом проблемы останова; следовательно, согласно теореме 1, тавтологии имеют полную степень точно так же, как проблема останова имеет полную рекурсивно перечислимую степень. К сожалению, диагональная процедура, которая показывает, что проблема останова не рекурсивна, по-видимому, не может быть приспособлена для того, чтобы показать, что {тавтологии} не принадлежит $\mathcal{L}_*$. <…>

35 Статистическая интерпретация специфичности термина и ее применение к поиску (1972)

Карен Спарк Джонс

Всемирная паутина существует только с середины 1990-х годов, а поисковые системы – и того меньше. Но задача поиска документов по ключевым словам существовала с тех пор, как появились документы. Как лучше всего построить индекс терминов, встречающихся в документах, чтобы было легко найти документ по набору интересующих терминов?

В этой статье 1972 года Карен Спарк Джонс (1935–2007) описала простой метод, который до сих пор лежит в основе систем поиска документов. Он состоит из двух частей. Чем чаще термин встречается в документе, тем, скорее всего, выше его релевантность содержимому документа. Например, статья, где много раз используется термин «зебра», вероятно, имеет какое-то отношение к зебрам. Но, конечно, в такой статье будет также много раз встречаться термин «the», поэтому одна лишь частота встречаемости в документе не может служить надежным индикатором значимости слова.

Однако важность высокой частоты термина в документе необходимо умерить, если этот термин часто встречается во многих других документах. Чем реже термин встречается в коллекции документов, тем вероятнее, что он имеет высокую значимость для тех немногих документов, в которых все же встречается. Этот множитель, играющий роль противовеса, называется «обратной частотой документа», или IDF (inverse document frequency). В большинстве современных поисковых систем в вебе IDF лежит в основе поиска веб-страниц.

Методы Спарк Джонс, тщательно выверенные с применением математической статистики, оказались замечательно полезными, хотя в них не было никакой попытки извлечь из документов смысл, проанализировать структуру предложения или объединить в один кластер похожие слова. То есть она продемонстрировала, что простая статистика текста может оказаться исключительно полезным средством анализа естественного языка.

Хотя она работала в лабораториях, а потом в Кембриджском университете с 1950-х годов до ухода на пенсию в 2002 году и стала президентом Ассоциации вычислительной лингвистики в 1994 году, звание профессора ей было присвоено только в 1999 году (Bowles 2019).

35.0. Краткое описание

Исчерпывающая полнота описаний документов и специфичность терминов, включенных в индекс, обычно считаются независимыми характеристиками. Предлагается интерпретировать специфичность статистически, как функцию термина, а не смысла термина. Исследуется влияние на поиск различных вариантов специфичности термина; в частности, эксперименты с тремя тестовыми коллекциями показывают, что часто встречающиеся термины необходимы для хорошей общей результативности. Предлагается назначать терминам веса в соответствии с частотой их встречаемости в коллекции, так чтобы присутствие менее частых, более специфичных терминов имело большую ценность, чем присутствие частых терминов. Результаты для тестовых коллекций показывают, что эта очень простая процедура приводит к значительному увеличению результативности.

35.1. Полнота и специфичность

Мы знакомы с понятиями полноты и специфичности: полнота – свойство индексных описаний, а специфичность – индексируемых терминов. Наиболее наглядно они проявляются в простой системе на основе ключевых слов, или дескрипторов. В этом случае под полнотой описания документа понимается покрытие различных затрагиваемых в нем тем сопоставленными ему терминами, а под специфичностью отдельного термина – уровень детализации, с которым представлена данная концепция.

Эти особенности системы поиска документов обсуждались, например, в работах (Cleverdon et al. 1966) и (Lancaster 1968), и в каждой отмечалось влияние различных вариантов. Например, если полноту описания документа увеличить за счет включения большего числа терминов, оставив общее количество терминов в словаре индекса постоянным, то шансы, что документ будет отвечать запросу, увеличиваются. Далее следует идея об оптимальном уровне полноты индексирования для заданной коллекции документов: среднее число дескрипторов на один документ должно быть подобрано так, чтобы по возможности максимизировать шансы на то, что документ будет отвечать запросу, избежав в то же время слишком большого числа ложных попаданий в результаты поиска. Очевидно, что понятие полноты применимо также и к запросам, и одной из функций стратегии поиска является варьирование полноты запроса. Меня, однако, будут в первую очередь интересовать описания документов.

Специфичность в том виде, в каком она описана выше, – семантическое свойство индексируемых терминов: термин специфичен тем больше или тем меньше, чем больше или меньше детализирован и точен его смысл. Это естественная точка зрения всякого, кто занимался построением полного словаря индекса. Необходимо принять какое-то решение о различительной способности отдельных терминов в дополнение к их описательной правильности. Например, индексируемый термин «напиток» можно с полным правом использовать для документов, касающихся чая, кофе и какао, как и сами термины «чай», «кофе» и «какао». Включать ли в словарь более общий термин «напиток» или отдать предпочтение терминам «чай», «кофе» и «какао», зависит от суждений о полезности для поиска различий между документами в случае принятия второго, а не первого решения. Можно также предположить, что более общий термин будет применим к большему числу документов, чем отдельные термины «чай», «кофе» и «какао», поэтому менее специфичный термин будет иметь в коллекции большее распространение, чем более специфичные.

Разумеется, здесь предполагается, что такие варианты построения словаря являются взаимно исключающими: либо «напиток», либо «чай», «кофе» и «какао». Что произойдет, если включить все четыре термина, – другой вопрос. Мы можем интерпретировать «напиток» как «прочие напитки» или явно трактовать его как родственный, но более общий термин. Однако здесь я оставляю такие альтернативы в стороне.

При построении словаря индекса специфичность индексируемых терминов рассматривается с одной точки зрения: нас интересует вероятное влияние на описание, а потому и на поиск документа, оказываемое выбором конкретных терминов или, точнее, принятием определенного множества терминов. Ибо на наши решения будут отчасти влиять отношения между терминами и то, как множество выбранных терминов характеризует набор документов. Но на протяжении всей статьи мы предполагаем какой-то уровень полноты индексирования. Мы хотим получить эффективный словарь для коллекции документов на некоторую широко известную тему, предполагая, что заданный уровень полноты индексирования достаточен для адекватного представления отдельных документов и отличения одного документа от другого.

Однако на специфичность индексируемых терминов следует смотреть с другой точки зрения. Что происходит при использовании данного словаря индекса? Например, отдавая предпочтение термину «напиток», мы предсказываем, что он будет использоваться чаще, чем «какао». Но мы плохо представляем себе, сколько будет документов, которым можно с должным основанием сопоставить «напиток». Это непросто определить, даже когда предполагается некоторый уровень полноты. Найдутся какие-то документы, которые, так сказать, умоляют сопоставить себе термин «напиток», и, возможно, у нас имеются идеи насчет доли таких документов в коллекции. Найдутся также документы, приписывать которым термин «напиток» нет никаких оснований, и эту долю, возможно, также удастся оценить. Но, к сожалению, неизбежно присутствие документов, которым «напиток» можно сопоставить или не сопоставить с равными основаниями. Поэтому в общем случае фактическое использование дескриптора может значительно отличаться от предсказанного. Доли коллекции, которым некий термин соответствует и не соответствует, можно оценить лишь очень приблизительно. На протяжении длительного периода времени характер самой коллекции в целом может претерпеть изменения, что также отразится на распределении терминов.

Именно здесь играет роль уровень полноты описания. По мере роста коллекции поддержание определенного уровня полноты может означать, что описания различных документов недостаточно отличаются друг от друга, тогда как некоторые термины используются очень часто. Вообще, вполне вероятно появление существенных различий в распределении терминов. Таким образом, может случиться, что некоторый термин становится менее эффективным в роли средства поиска независимо от его фактического смысла. Все дело в том, что он не обладает необходимой различительной способностью. Возможно, он с полным основанием сопоставлен документам в том смысле, что соответствует их содержимому, но сам по себе со временем перестал быть инструментом для отделения сравнительно малого класса документов, релевантных запросу, от остальной части коллекции. Поэтому часто встречающийся термин при поиске играет роль неспецифического термина, пусть даже в обиходном смысле обладает вполне специфическим значением.

35.2. Статистическая специфичность

Иными словами, недостаточно рассуждать об индексируемом термине исключительно в плане подготовки словаря, считая, что он отражает точность представления концепции. Следует рассматривать специфичность как функцию использования термина. Ее нужно интерпретировать как статистическую характеристику, а не как семантическое свойство индексируемых терминов. В общем случае можно ожидать, что расплывчатые термины будут встречаться чаще, но поведение отдельных терминов будет непредсказуемым. Таким образом, мы можем переопределить полноту и специфичность

для простых систем терминов: полнота описания документа – это число содержащихся в нем терминов, а специфичность термина – это число документов, к которым он относится. Тогда отношение между тем и другим становится понятным, и можно видеть, например, что изменение полноты описаний повлияет на специфичность термина: если описания становятся длиннее, то термины будут использоваться чаще. Это неизбежно для контролируемого словаря, но применимо также в случае, когда используются выделенные из документов слова, в частности в форме основы. Частоты встречаемости слов, не входивших ранее в словарь, не увеличиваются синхронно с ростом числа проиндексированных документов, а выделение большего числа ключевых слов из документа с большей вероятностью увеличит частоту уже имеющихся слов, чем приведет к порождению новых.

Коль скоро эта статистическая интерпретация специфичности и соотношение между ней и полнотой признаны, естественно попытаться применить более формальный подход к поиску оптимального уровня специфичности в словаре и оптимального уровня полноты при индексировании заданной коллекции. Оставаясь в рамках широких пределов, налагаемых требованием разумности терминов, т. е. использования таких, которые можно найти с помощью запросов и применить к документам, мы можем попытаться построить словарь со статистическими свойствами, которые, хочется надеяться, будут оптимальны для поиска. Чисто формальные вычисления могут подсказать число терминов и терминов в расчете на один документ, подходящие для различения документов с определенной степенью уверенности. Работа в этом направлении была проделана, например, в статье (Zunde and Slamecka 1967). Более неформально, предположение о том, что дескрипторы следует конструировать, так чтобы они имели приблизительно одинаковое распределение, сделанное, например, в работе (Salton 1968), мотивировано учетом влияния на поиск чисто статистических особенностей использования термина.

К сожалению, абстрактные вычисления не помогут выбрать фактические термины. Да и коллекции документов не являются статическими. Но еще важнее то, что трудно контролировать запросы. Мы можем охарактеризовать документы, стремясь обеспечить их четкое различение, а затем обнаружить, что пользователи не предъявляют запросы, в которых эти различия используются. Поэтому нам, возможно, придется принять сложившуюся де-факто неоптимальную ситуацию, когда имеются термины различной специфичности и сколько-то неприятных неспецифичных терминов. Будут встречаться термины, которые вне зависимости от первоначального намерения извлекают много документов, из которых лишь малая часть релевантна запросу. В целом такие термины доставляют больше хлопот, чем редкие, чрезмерно специфичные термины, не извлекающие ни одного документа.

Эти особенности поведения терминов можно проиллюстрировать на примерах из трех хорошо известных тестовых коллекций, взятых из проектов Aslib Cranfield, INSPEC и College of Librarianship Wales. На самом деле словари в них состоят из основ выделенных из документов ключевых слов, от которых можно ожидать большего разнообразия, чем от контролируемых терминов. Но нет причин полагать, что ситуация чем-то принципиально отличается.

Полные описания коллекций приведены в работах (Cleverdon et al. 1966), (Aitchison et al. 1970) и (Keen and Digger 1972). Интересующие нас характеристики коллекций приведены в секции A на рис. 35.1. Коллекция INSPEC, например, содержит 541 документ, индексированный по 1341 термину. Во всех коллекциях имеются очень часто встречающиеся термины: например, в коллекции Cranfield один термин встречается в 144 из 200 документов; в коллекции INSPEC один термин встречается в 112 из 541 документа, а в коллекции Keen один термин встречается в 199 из 797 документов. Эти термины необязательно представляют концепции, центральные для тематики коллекций, и не всегда они являются общими. В коллекции Keen, относящейся к научной информации, самым частым является термин «index-», а кроме него, часто встречаются термины «librar-», «inform-» и «comput-». В коллекции INSPEC самым частым является термин «theor-», а за ним следуют «measur-» и «method-». А в коллекции Cranfield самым частым является термин «flow-», за которым следуют «pressur-», «distribut-» и «bound-» (boundary). К более редким относятся такие разнородные термины, как «purchas-» и «xerograph-» для Keen, «parallel-» и «silver-» для INSPEC и «logarithm-» и «seri-» (series) для Cranfield.

		Cranfield	INSPEC	Keen
A.	Число документов	200	541	797
	Число терминов	712	1341	939
	Число терминов на один документ	32	12,2	7,9
	Число документов на один термин	9	4,9	6,1
B.	Число запросов	42	97	63
	Число представленных терминов	166	248	183
	Число терминов на один запрос	6,9	5,6	5,3
	Число документов на один термин запроса	31,6	11,5	44,8
C.	Число результативных терминов на запрос	5	3,2	3,3
	Число результативных терминов на документ	1,8	1,2	1,2
	Число результативных терминов на релевантный документ	3,6	2	1,8
D.	Число частых терминов	96	73	50
	Число частых терминов на запрос	4	2,5	2,3

Рис. 35.1

35.3. Специфичность и сопоставление

Как справляться с переменной специфичностью терминов и особенно с недостаточно специфичными терминами, когда они встречаются в запросах? С нежелательными эффектами использования частого термина, в принципе, можно бороться вполне естественно, посредством комбинирования терминов. Например, хотя термины «bound-», «layer-» и «flow-» встречаются соответственно в 73, 62 и 144 документах в коллекции Cranfield, существует всего 50 документов, индексированных всеми тремя символами. Полагаться на конъюнкцию терминов вполне нормально. В частности, это способ пре-

одолеть нежелательные последствия того факта, что запросы обычно формулируются с использованием хорошо известных, а потому более частых терминов. Неприятно, но не удивительно, что в запросах обычно употребляются термины, средняя частота которых намного выше частоты для словаря индекса в целом. Это верно для всех трех тестовых коллекций, как видно из секции B на рис. 35.1. Например, для коллекции Cranfield среднее число документов на один словарный термин равно девяти, тогда как среднее для терминов, встречающихся в запросах, равно 31.6; для коллекции Keen эти цифры равны 6.1 и 44.8.

Но, как хорошо известно, полагаться на комбинацию терминов для уменьшения ложных попаданий в результаты поиска рискованно. Да, действительно, чем больше общих терминов у документа и запроса, тем вероятнее, что документ будет релевантен запросу. К сожалению, найти совпадения для конъюнкций терминов попросту трудно. Это наглядно демонстрирует секция C на рис. 35.1, где показано, как ведет себя показатель совпадения с термином во всех трех коллекциях. Среднее число терминов в запросе варьируется от 5,3 для Keen до 6,9 для Cranfield. Но среднее число результативных терминов на запрос, т. е. среднее наивысших оценок совпадения, варьируется от 3,2 до 5,0. Важнее, однако, то, что среднее число совпавших терминов для релевантных найденных документов варьируется от всего лишь 1,8 для Keen до 3,6 для Cranfield, хотя, к счастью, среднее значение для всех найденных документов, по преимуществу нерелевантных, варьируется от жалких 1,2 до 1,8.

Одно решение этой проблемы очевидно – каким-то образом увеличить количество результативных терминов. Этого можно достичь, либо предоставив альтернативные замены данным терминам с помощью классификации, либо увеличив полноту спецификаций документа или запроса, скажем, путем добавления статистически связанных терминов. Но оба подхода требуют усилий, быть может значительных, поскольку для каждого термина необходимо идентифицировать множество родственных ему. Естественно возникает вопрос, нельзя ли эффективнее использовать существующие описания терминов, не прилагая таких усилий.

Поскольку очень часто встречающиеся термины несут ответственность за поисковый шум, один из возможных путей – попросту удалить их из запроса. То, что при этом уменьшается количество терминов, пригодных для объединенного сопоставления, уравновешивается уменьшением числа найденных нерелевантных документов. К сожалению, хотя частые термины являются причиной шума, они также необходимы для достижения разумно высокой полноты возврата. Для всех трех коллекций удаление самых частых терминов путем задания подходящего порога приводит к уменьшению общего качества поиска. Например, для коллекции INSPEC порог был задан так, чтобы удалялись термины, встречающиеся в 20 и более документах, поэтому из словаря, содержащего всего 1341 термин, было удалено 73. Для коллекции Cranfield влияние на качество поиска показано на рис. 25.2 в виде графика полнота–точность. Сопоставление осуществляется с помощью простых уровней координации терминов, а усреднение по набору запросов – с помощью простого усреднения чисел. Затем точность интерполируется по десяти стандартным значениям полноты. Такая же связь между сопоставлением по

всем терминам и только нечасто встречающимся имеет место и для других коллекций: потолок полноты возврата снижается по меньшей мере на 30 %, а для коллекции Keen – даже с 75 до 25 %, хотя точность сохраняется.

Пристальное изучение запросов показывает, почему так происходит. Мало того, что частота термина в запросах гораздо выше средней частоты в коллекции, так еще и сравнительно небольшое число частых терминов играет наибольшую роль в формулировке запроса. Например, «flow-» встречается в 12 из 42 запросов в коллекции Cranfield, а вообще для всех трех коллекций примерно половина терминов в запросе встречаются очень часто, как видно из секции D на рис. 35.1. Отбрасывание очень частых терминов сродни выплескиванию ребенка вместе с водой, поскольку они необходимы для нахождения многих релевантных документов. Комбинация нечастых терминов обладает различительной способностью, но не большей, чем комбинация частых и нечастых терминов. С другой стороны, ценность нечастых терминов наглядно проявляется, если сравнить сопоставление только с частыми терминами и сопоставление со всеми терминами, также показанное на рис. 35.2. Уровни совпадения для всех и для релевантных документов почти так же высоки, как для всех терминов, но нечастые термины поднимают уровень совпадения с релевантными документами почти до 1.

Эти особенности поиска терминов наводят на мысль, что для улучшения исходного качества поиска по всем терминам необходимо взять на вооружение хорошие свойства очень частых и нечастых терминов и минимизировать их плохие свойства. Мы должны разрешить некоторые достоинства частых терминов и еще большее количество достоинств нечастых. В любом случае мы стремимся максимизировать количество совпавших терминов.

35.4. Взвешивание по специфичности

На ум сразу приходит схема взвешивания. При обычном сопоставлении с термином, если у запроса и документа имеется общий частый термин, то он считается точно так же, как нечастый; поэтому если у запроса и документа три общих частых термина, то уровень релевантности документа будет таким же, как у документа, который имеет с запросом три общих нечастых термина. Но, наверное, стоило бы рассматривать совпадения с нечастыми терминами как более ценное, чем с частыми, не отбрасывая, однако, последнее полностью. Естественное решение – коррелировать ценность совпадения с термином с его частотой в коллекции. На этой стадии разделение терминов на частые и нечастые произвольно и, вероятно, не оптимально. Элегантный и почти наверняка лучший подход – более тесно связать ценность совпадения с относительной частотой. Как это сделать, подсказывает кривая распределения терминов в словаре, имеющая знакомую форму Ципфа. Пусть $f(n) = m$ такому, что $2^{m-1} < n \leqslant 2^m$. Тогда если в коллекции N документов, то вес термина, встречающегося n раз, равен $f(N) - f(n) + 1$. Например, в коллекции Cranfield 200 документов, поэтому термин, встречающийся 90 раз, имеет вес 2, а встречающийся 3 раза – вес 7.

Ценность термина для сопоставления, таким образом, коррелируется с его специфичностью, а уровень документа при поиске определяется суммой ценностей результативных терминов. Простые уровни координации заменяются более сложным квазиранжированием. Эффект можно проиллюстрировать разными поисковыми уровнями, при которых два документа соответствуют запросу с одним и тем же числом относительно частых и относительно нечастых терминов. В случае диапазона ценностей в коллекции Cranfield документ, сопоставляемый двум терминам с частотой 15 и 43, будет иметь при поиске уровень 5 + 3 = 8, а документ, сопоставляемый терминам с частотой 3 и 7, – уровень 7 + 6 = 13. Ясно, что по мере «растягивания» диапазона ценностей становится возможным большее различение.

Идея взвешивания термина не нова. Но обычно она связывается с предполагаемой важностью термина относительно самого документа. Например, если документ в основном посвящен краске, а лак упоминается лишь мимоходом, то можно использовать простую шкалу весов, в которой термину «краска» назначается вес 2, а термину «лак» – вес 1. Менее формально, формулируя запрос, мы можем сказать, что при поиске термин *x* следует сохранить, а термин *y* можно отбросить. Если имеется необходимая информация, то мы можем придумать более систематическое назначение весов на статистической основе. Если фактическая частота встречаемости терминов в документе (или его реферате) известна, то ее можно использовать для порождения весов. В работе (Artandi and Wolf 1969) сообщается об использовании частоты для выбора веса по трехбалльной шкале, а в работе (Salton and Lesk 1968) частота встречаемости честно используется в качестве веса. В ряде экспериментов Салтон продемонстрировал, что такое взвешивание терминов дает заметное улучшение качества поиска по сравнению с невзвешенными терминами.

Взвешивание по частоте встречаемости в коллекции, а не в документе преследует совершенно другую цель. Больший упор в нем делается на ценность термина в качестве средства для отличения одного документа от другого, а не в качестве индикатора содержимого самого документа. Связь между двумя формами взвешивания не очевидна. Иногда термин может часто встречаться в документе, но редко в коллекции, поэтому обе схемы назначат ему большой вес. Но может случиться и прямо противоположное. Важно, что акцент ставится на разные свойства терминов.

Трактовка частоты термина в коллекции в контексте сопоставления с термином, похоже, систематически не изучалась. Влияние частоты термина на статистические ассоциации изучалось, например, Леском, но это другое. Тот факт, что заданный термин, вероятно, позволит найти большое число документов, можно неформально использовать при настройке поиска, в частности в контексте онлайнового поиска, описанного, например, в работе Borko (1968). Более честным подходам, по-видимому, мешает недостаток необходимой информации. Кроме того, описанная выше процедура гораздо больше подходит для автоматического, а не ручного поиска. Поэтому представляет интерес, что частоты терминов использовались общим образом в оперативной интерактивной поисковой системе для составления внутренних отчетов, реализованной в компании A. D. Little (Curtice and Jones 1968). В этой системе

индексируемые ключевые слова автоматически выделялись из текста, поэтому схема взвешивания ассоциировалась с изменением словаря и коллекции. Но ни о каких систематических экспериментах не сообщается.

35.5. Экспериментальные результаты

Описанная система взвешивания терминов была опробована на трех коллекциях. Как уже было сказано, они сильно различаются по характеру, размерам словарей, описаниям документов и заданию запросов (см. рис. 35.1). Однако во всех случаях применение сопоставления со взвешиванием терминов давало значительное улучшение качества поиска по сравнению с простым сопоставлением с термином. Результаты, представленные в описанной выше форме, показаны на рис. 35.3, 35.4 и 35.5. Простая проверка значимости, основанная на разности площадей под кривыми, показывает, что улучшение за счет взвешивания терминов вполне значимо, а разность заметно выше необходимого минимума.

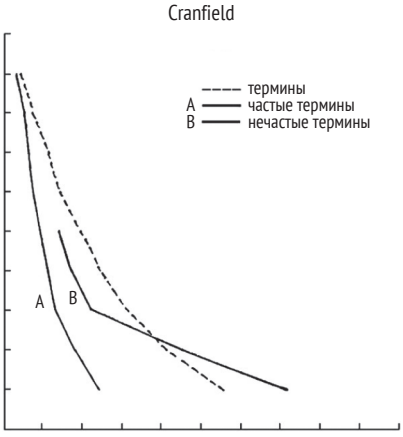


Рис. 35.2

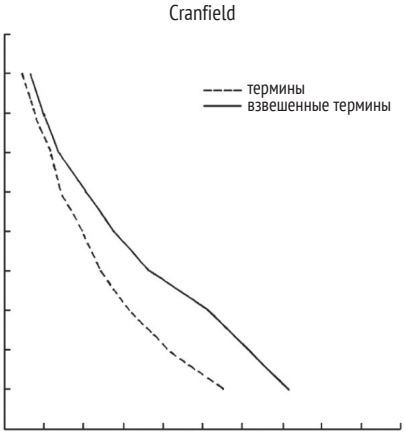


Рис. 35.3

Эти результаты представляют интерес по двум причинам. Все три коллекции использовались для целого ряда экспериментов с разными индексируемыми языками, методами поиска и т. д.; см. Cleverdon et al. (1966); Salton (1968); Salton and Lesk (1968); Spärck Jones (1971); Aitchison et al. (1970); Keen and Digger (1972). Тем не менее достигнутое здесь улучшение качества поиска ничуть не меньше улучшения качества по сравнению с сопоставлением с невзвешенной основой ключевого слова, достигнутого другими средствами, включая использование тщательно построенного тезауруса; методы итеративного поиска Салтона не допускают сравнения. Детали представления этих экспериментальных результатов разнятся, поэтому строгое сравнение невозможно, но общая картина ясна. На самом деле в той мере, в какой что-то вообще можно назвать добротным результатом в исследованиях по информационному поиску, результат этой работы является таковым. Второе, что

можно сказать по поводу представленных результатов, – то, что улучшение качества было достигнуто очень простыми средствами. Метод совместим с первоначальным простым методом индексирования – использованием выделенных ключевых слов, которые можно автоматически свести к основе; он легко реализуется при наличии автоматической процедуры сопоставления с термином, поскольку все, что требуется, – это список частот терминов, а его легко получить. Дополнительным достоинством является то, что назначенные терминам веса естественно подстраиваются к росту и изменениям коллекции. Понятно, что желательны эксперименты с гораздо большими коллекциями, чем представленные здесь; будем надеяться, что они не заставят себя долго ждать.

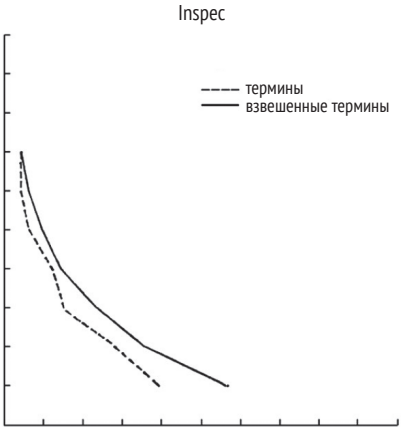


Рис. 35.4

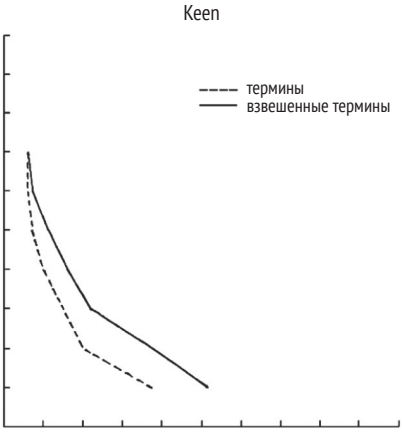


Рис. 35.5

36 Сводимость комбинаторных проблем (1972)

Ричард Карп

Ричард (Дик) Карп родился в Дорчестере, городке близ Бостона. Его отец был школьным учителем, доросшим до директора школы. Карт любил игры и головоломки и свой первый практический урок комбинаторного взрыва получил, помогая отцу составлять расписание уроков, тасуя карточки в поисках такого варианта, который удовлетворял бы различным ограничениям на использование кабинетов, доступность учителей и т. д.

Карп изучал математику в Гарварде и нашел особое очарование в дискретной математике, поскольку она отвечала его склонности к решению головоломок и поскольку он робел перед лучшими студентами по чистой математике в своей группе, некоторые из которых потом получили важные математические премии. Получив степень бакалавра в 1955 году, Карп поступил в аспирантуру по цифровым вычислительным машинам и в 1959 году получил степень доктора. Затем он начал работать в научно-исследовательском подразделении IBM, где внес вклад в изучение различных компьютерных алгоритмов, особенно в области комбинаторной оптимизации. В 1968 году он ушел из IBM и стал профессором информатики в Беркли, где и работает по сей день.

Эти детали биографии подготовили сцену для реакции Карпа на работу Кука (Cook 1971b, здесь глава 34). Карп понял, что очень многие комбинаторные задачи имеют семейное сходство с булевой выполнимостью в том смысле, что для их решения, похоже, необходим поиск в экспоненциально большом пространстве доказательства или свидетельства, которое (если будет найдено) окажется относительно малым и легко проверяемым.

В этой работе Карп определяет классы $\mathcal{P}$, $\mathcal{NP}$, полиномиальную сводимость (вариант вычислительной сводимости, которую изучал его руководитель по программе бакалавриата Хартли Роджерс) и $\mathcal{N}$-полноту. Затем он приводит список из 21 проблемы, все, очевидно, принадлежащие классу $\mathcal{NP}$, к которым прямо или косвенно сводится проблема булевой выполнимости, показав тем самым, что все они $\mathcal{NP}$-полны. У многих из этих проблем долгая история: в работе Карпа показано, что все они по существу представляют собой одну и ту же проблему и отличаются лишь несущественными вариациями. Вскоре в список Карпа были добавлены сотни других проблем, а вопрос о справедливости равенства $\mathcal{P} = \mathcal{NP}$ стал одной из самых важных нерешенных проблем современной математики. Работа (Garey and Johnson 1979) – свод, к которому специалисты по информатике обращаются, когда хотят узнать, является ли новая проблема $\mathcal{N}$-полной, – в этом случае они прекращают попытки найти эффективное точное решение и переключаются на поиск приближенных методов, более простых частных случаев и т. д.

Большой класс вычислительных проблем включает определение свойств графов, ориентированных графов, целых чисел, массивов целых чисел, конечных семейств конечных множеств, булевых формул и элементов других счетных множеств. Посредством простого кодирования таких объектов множествами слов над конечным алфавитом эти проблемы могут быть превращены в проблемы распознавания языков, и можно исследовать их вычислительную сложность. Имеет смысл считать, что такая проблема решается удовлетворительно, если найден алгоритм для ее решения, завершающий работу за время (число шагов), ограниченное полиномом от длины входного слова. Мы покажем, что многие классические нерешенные проблемы покрытия, сочетания, упаковки, нахождения маршрутов, назначения и упорядочения эквивалентны между собой в том смысле, что или каждая из них обладает полиномиально ограниченным алгоритмом, или ни одна из них такого алгоритма не имеет.

36.1. Введение

Все известные в настоящее время общие методы вычисления хроматического числа графа, определения существования гамильтонова контура в графе или решения системы линейных неравенств, в которых переменные принимают значения 0 или 1, требуют вычислений, длительность которых растет в худшем случае экспоненциально по отношению к длине слова. В настоящей статье мы докажем ряд теорем, которые дают повод (хотя прямо из них это не следует) считать, что эти проблемы, а также многие другие будут всегда оставаться трудноразрешимыми.

Нас особенно интересует существование алгоритмов, обеспечивающих завершение работы за число шагов, ограниченное полиномом от длины входа. Мы укажем класс хорошо известных комбинаторных проблем, включая

упомянутые выше, которые эквивалентны друг другу в том смысле, что полиномиально ограниченный алгоритм для любой из них будет эффективно порождать полиномиально ограниченные алгоритмы для всех. Мы также покажем, что если эти проблемы обладают полиномиально ограниченными алгоритмами, то все проблемы из очень широкого класса (грубо говоря, класса проблем, разрешимых путем обратного поиска полиномиальной глубины) обладают полиномиально ограниченными алгоритмами.

Изложим вкратце содержание настоящей статьи. Ради определенности изложение ведется в терминах распознавания языков на одноленточных машинах Тьюринга, но любая другая модель из довольно широкого множества абстрактных моделей вычислений позволяет развить подобную теорию. Пусть Σ^* – множество всех конечных слов, составленных из нулей и единиц. Произвольное подмножество Σ^* называется *языком*. Обозначим $\mathcal{P}$ класс языков, распознаваемых за полиномиальное время на одноленточных детерминированных машинах Тьюринга, а $\mathcal{NP}$ – класс языков, распознаваемых за полиномиальное время на одноленточных недетерминированных машинах Тьюринга. Пусть П – класс функций, определенных на Σ^* со значениями из Σ^*, вычислимых за полиномиальное время на одноленточных машинах Тьюринга. Пусть L и M – языки. Говорят, что $L \preccurlyeq M$ (*L сводится к M*), если существует функция $f \in$ П такая, что $f(x) \in M \Leftrightarrow x \in L$. Если $M \in \mathcal{P}$ и $L \preccurlyeq M$, то $L \in \mathcal{P}$. Назовем L и M *эквивалентными*, если $L \preccurlyeq M$ и $M \preccurlyeq L$. Назовем L (*полиномиально*) *полным*, если $L \in \mathcal{NP}$ и любой язык из $\mathcal{NP}$ сводится к L. Очевидно, что либо все полные языки принадлежат $\mathcal{P}$, либо ни один из них не принадлежит $\mathcal{P}$. Первое имеет место тогда и только тогда, когда $\mathcal{P} = \mathcal{NP}$.

Главное содержание этой статьи заключается в доказательстве того, что многие классические трудные вычислительные проблемы, возникающие в таких областях, как математическое программирование, теория графов, комбинаторика, вычислительная логика, теория переключательных схем, являются полными (и, следовательно, эквивалентными), когда они выражаются естественным путем как проблемы распознавания языков.

Эта статья навеяна работой Стивена Кука (Cook 1971b) и опирается на важную теорему из этой работы. Автор признателен Юджину Лоулеру и Роберту Тарьяну за большую помощь.

36.2. Класс $\mathcal{L}$

Существует большой класс важных вычислительных проблем, включающий в себя определение свойств графов, ориентированных графов, целых чисел, конечных семейств конечных множеств, булевых формул и элементов других счетных областей. Разумная рабочая гипотеза, которую впервые стал защищать Джек Эдмондс (Edmonds 1965) в связи с проблемами теории графов и целочисленного программирования и которая поныне широко принята, состоит в том, что такого рода проблема может считаться легко решаемой тогда и только тогда, когда существует алгоритм ее решения со временем работы, которое ограничено полиномом от размера входных данных. В этом

разделе мы вводим и начинаем исследовать класс проблем, разрешимых за полиномиальное время. <...>

Мы завершим этот раздел перечислением проблем, разрешимых за полиномиальное время. В следующем разделе мы рассмотрим несколько родственных по отношению к ним проблем, о которых неизвестно, разрешимы ли они за полиномиальное время. В приложении 1 приводится система обозначений. Каждая проблема определяется заданием (под заголовком «ВХОД») элемента, характеризующего область определения, и (под заголовком «СВОЙСТВО») свойства, которое определяет, следует ли данный вход принять.

2-ВЫПОЛНИМОСТЬ (выполнимость КНФ не более чем с двумя буквами в каждой скобке) (Cook 1971b)
ВХОД: скобки (элементарные дизъюнкции) $C_1, C_2, \ldots, C_p$, каждая из которых содержит не более двух букв
СВОЙСТВО: конъюнкция данных скобок выполнима, т. е. существует множество

$$S \subseteq \{x_1, x_2, \ldots, x_n, \overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$$

такое, что
a) S не содержит пары букв, одна из которых является отрицанием другой, и
b) $S \cap C_k \neq \emptyset$, $k = 1, 2, \ldots, p$.

МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО (Kruskal 1956, здесь глава 17)
ВХОД: G, w, W
СВОЙСТВО: существует остовное дерево веса $\leqslant W$.

КРАТЧАЙШИЙ ПУТЬ (Dijkstra 1959)
ВХОД: G, w, W, s, t
СВОЙСТВО: существует путь между s и t веса $\leqslant W$.

МИНИМАЛЬНЫЙ РАЗРЕЗ (Edmonds and Karp, 1972)
ВХОД: G, w, W, s, t
СВОЙСТВО: существует s, t разрез веса $\leqslant W$. <...>

36.3. Недетерминированные алгоритмы и теорема Кука

В этом разделе мы сформулируем важную теорему, доказанную Куком (Cook 1971b) и утверждающую, что любой язык из определенного широкого класса языков $\mathcal{NP}$ сводится к конкретному множеству S, соответствующему проблеме выполнимости булевой формулы, заданной в конъюнктивной нормальной форме.

Обозначим $\mathcal{P}^{(2)}$ класс подмножеств множества $\Sigma^* \times \Sigma^*$, распознаваемых за полиномиальное время. Для данных $L^{(2)} \in \mathcal{P}^{(2)}$ и полинома p определим язык L следующим образом:

$$L = \{x: \text{существует } y \text{ такой, что } \langle x, y\rangle = L^{(2)} \text{ и } \lg(y) \leqslant p(\lg(x))\}.$$

Будем говорить, что язык L выводим из $L^{(2)}$ путем навешивания p-ограниченного квантора существования.

Определение. $\mathcal{NP}$ есть множество языков, выводимых из элементов $^{(2)}$ путем навешивания полиномиально ограниченных кванторов существования.

Существует другая характеристика $\mathcal{NP}$ в терминах недетерминированных машин Тьюринга.

<...>

Класс $\mathcal{NP}$ весьма широк. Допуская вольность речи, можно сказать, что данная проблема принадлежит классу $\mathcal{NP}$ тогда и только тогда, когда она может быть решена путем обратного поиска полиномиально ограниченной глубины. Большое число важных вычислительных проблем, для которых неизвестно, принадлежат ли они $\mathcal{P}$, очевидно, принадлежат $\mathcal{NP}$. Например, проблема определения, можно ли раскрасить вершины графа G таким образом, что никакие две соседние вершины не будут покрашены одинаковым цветом. Недетерминированный алгоритм может просто наугад задать цвета вершин и затем проверить (за полиномиальное время), имеют ли все пары соседних вершин различный цвет.

Ввиду большой широты класса $\mathcal{NP}$ следующая теорема, принадлежащая Куку, является весьма примечательной. Проблема выполнимости определяется следующим образом:

ВЫПОЛНИМОСТЬ
ВХОД: скобки (элементарные дизъюнкции) $C_1, C_2, \ldots, C_p$.
СВОЙСТВО: конъюнкция данных скобок является выполнимой,
т. е. существует множество

$$S \subseteq \{x_1, x_2, \ldots, x_n, \overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}\}$$

такое, что
a) S не содержит никакой буквы вместе с ее отрицанием и
b) $S \cap C_k$, $k = 1, 2, \ldots, p$.

Теорема 2 (Кук). Если $L \in \mathcal{NP}$, то $L \leqslant$ ВЫПОЛНИМОСТЬ.

В теореме, установленной Куком (Cook 1971b), используется более слабое понятие сводимости, чем то, которое используется здесь, но доказательство Кука справедливо и для нашего случая.

Следствие 1. $\mathcal{P} = \mathcal{NP} \Leftrightarrow$ ВЫПОЛНИМОСТЬ $\in \mathcal{P}$.

Доказательство. Если ВЫПОЛНИМОСТЬ $\in \mathcal{P}$, то для любого $L \in \mathcal{NP}$ также $L \in \mathcal{P}$, т. к. $L \leqslant$ ВЫПОЛНИМОСТЬ. Если ВЫПОЛНИМОСТЬ $\notin \mathcal{P}$, то $\mathcal{P} \neq \mathcal{NP}$, т. к. очевидно, что ВЫПОЛНИМОСТЬ $\in \mathcal{NP}$.

Замечание. Если $\mathcal{P} = \mathcal{NP}$, то $\mathcal{NP}$ замкнут относительно дополнения и полиноминально ограниченного квантора существования. Следовательно, он также замкнут относительно полиномиально ограниченного квантора общности. Из этого следует, что полиномиально ограниченный аналог арифметической иерархии Клини (Rogers 1967) становится тривиальным, если $\mathcal{P} = \mathcal{NP}$.

Теорема 2 показывает, что если бы существовал полиномиально ограниченный алгоритм распознавания ВЫПОЛНИМОСТИ, то любая проблема, разрешимая с помощью обратного поиска полиномиальной глубины, была

бы разрешима также с помощью (детерминированного) алгоритма за полиномиальное время. Это сильное косвенное свидетельство в пользу того, что ВЫПОЛНИМОСТЬ $\notin \mathcal{P}$.

36.4. Полные проблемы

Главная цель этой статьи – установление того, что большое число важных вычислительных проблем может играть роль проблемы ВЫПОЛНИМОСТЬ в теореме Кука. Такие проблемы мы будем называть полными.

Определение 5. Язык L называется *(полиномиально) полным*, если

a) $L \in \mathcal{NP}$ и
b) ВЫПОЛНИМОСТЬ $\preccurlyeq L$.

Теорема 3. Либо все полные языки принадлежат $\mathcal{P}$, либо ни один из них не принадлежит $\mathcal{P}$. Первое имеет место тогда и только тогда, когда $\mathcal{P} = \mathcal{NP}$. <...>

Оставшаяся часть статьи главным образом посвящена доказательству следующей теоремы.

Основная теорема. Все нижеследующие проблемы полные.

1. ВЫПОЛНИМОСТЬ
   Примечание. В силу двойственности эта проблема эквивалентна определению того, является ли некоторая дизъюнктивная нормальная форма тавтологией.
2. 0-1 ЦЕЛОЧИСЛЕННОЕ ПРОГРАММИРОВАНИЕ
   ВХОД: матрица целых чисел C и вектор целых чисел d
   СВОЙСТВО: существует вектор x из нулей и единиц (0-1 вектор), для которого $Cx = d$.
3. КЛИКА
   ВХОД: граф G, положительное целое число k
   СВОЙСТВО: G содержит множество k попарно смежных вершин.
4. УПАКОВКА МНОЖЕСТВ
   ВХОД: семейство множеств $\{S_j\}$, положительное целое число l
   СВОЙСТВО: $\{S_j\}$ содержит l попарно непересекающихся множеств.
5. ВЕРШИННОЕ ПОКРЫТИЕ
   ВХОД: граф G', положительное целое число l
   СВОЙСТВО: существует множество $R \subseteq N'$ такое, что $|R| \leqslant l$ и каждое ребро графа инцидентно некоторой вершине из R.
6. ПОКРЫТИЕ МНОЖЕСТВАМИ
   ВХОД: конечное семейство конечных множеств $\{S_j\}$, положительное целое число k
   СВОЙСТВО: существует подмножество $\{T_k\} = \{S_j\}$, содержащее $\leqslant k$ множеств и такое, что $\bigcup T_k = \bigcup S_j$.
7. МНОЖЕСТВО ВЕРШИН, РАЗРЕЗАЮЩИХ КОНТУРЫ
   ВХОД: ориентированный граф H, положительное целое число k
   СВОЙСТВО: существует множество $R \subseteq V$ такое, что каждый контур графа H содержит вершину из R.

8. МНОЖЕСТВО ДУГ, РАЗРЕЗАЮЩИХ КОНТУРЫ
   ВХОД: ориентированный граф H, положительное целое k
   СВОЙСТВО: существует множество $S \subseteq E$ такое, что каждый контур графа H содержит дугу из S.
9. ГАМИЛЬТОНОВ КОНТУР
   ВХОД: ориентированный граф H
   СВОЙСТВО: H содержит контур, включающий каждую вершину ровно один раз.
10. ГАМИЛЬТОНОВ ЦИКЛ
    ВХОД: граф G
    СВОЙСТВО: G имеет цикл, включающий каждую вершину ровно один раз.
11. 3-ВЫПОЛНИМОСТЬ (ВЫПОЛНИМОСТЬ НЕ БОЛЕЕ ЧЕМ С 3 БУКВАМИ В КАЖДОЙ СКОБКЕ)
    ВХОД: скобки $D_1, D_2, \ldots, D_r$, каждая из которых содержит не более 3 букв из множества букв $(u_1, u_2, \ldots, u_m\} \cup (\overline{u_1}, \overline{u_2}, \ldots, \overline{u_m}\}$
    СВОЙСТВО: множество $\{D_1, D_2, \ldots, D_r\}$ выполнимо.
12. ХРОМАТИЧЕСКОЕ ЧИСЛО
    ВХОД: граф G, положительное целое число k
    СВОЙСТВО: существует функция $\phi: N \to Z_k$ такая, что если вершины u и v смежные, то $\phi(u) \neq \phi(v)$.
13. ПОКРЫТИЕ КЛИКАМИ
    ВХОД: граф G', положительное целое число l
    СВОЙСТВО: N' есть объединение l или меньшего числа клик.
14. ТОЧНОЕ ПОКРЫТИЕ
    ВХОД: семейство $\{S_j\}$ подмножеств множества $\{u_i, i = 1, 2, \ldots, t\}$
    СВОЙСТВО: существует подсемейство $\{T_k\} \subseteq \{S_j\}$ такое, что множества T_k не пересекаются и $\cup T_k = \cup S_j = \{u_i, i = 1, 2, \ldots, t\}$
15. МНОЖЕСТВО ПРЕДСТАВИТЕЛЕЙ
    ВХОД: семейство $\{U_i\}$ подмножеств множества $\{s_j, j = =1, 2, \ldots, r\}$
    СВОЙСТВО: существует множество W такое, что для каждого i $|W \cap U_i| = 1$.
16. ДЕРЕВО ШТЕЙНЕРА
    ВХОД: граф G, $R \subseteq N$, весовая функция w: $A \to Z$, положительное целое число k
    СВОЙСТВО: G имеет поддерево веса $\leqslant k$, множество вершин которого содержится в R.
17. 3-МЕРНОЕ ПАРОСОЧЕТАНИЕ
    ВХОД: множество $U \subseteq T \times T \times T$, где T – конечное множество
    СВОЙСТВО: существует множество $W \subseteq U$ такое, что $|W| = |T|$ и любые два его элемента отличаются по всем координатам.
18. УКЛАДКА РЮКЗАКА
    ВХОД: $(a_1, a_2, \ldots, a_n, b) \in Z^{n+1}$
    СВОЙСТВО: уравнение $\sum a_j x_j = b$ имеет 0-1 решение.
19. УПОРЯДОЧЕНИЕ РАБОТ
    ВХОД: «вектор времен выполнения» $(T_l, T_p) \in Z^p$, «вектор сроков» $(D_1, \ldots, D_p) \in Z^p$, «вектор штрафов» $(P_1, \ldots, P_p) \in Z$, положительное целое число k

СВОЙСТВО: существует перестановка π множества $\{1, 2, \ldots, p\}$ такая, что

$$\left(\sum_{j=1}^{p} (\text{если } T_{\pi(1)} + \cdots + T_{\pi(j)} > D_{\pi(j)}, \text{ то } P_{\pi(j)}, \text{ иначе } 0)\right) \leq k.$$

20. РАЗБИЕНИЕ
ВХОД: $(c_1, c_2, \ldots, c_s) \in Z^s$
СВОЙСТВО: существует множество $I \subseteq \{1, 2, \ldots, s\}$ такое, что $\sum_{h \in I} c_h = \sum_{h \notin I} c_h$.
21. МАКСИМАЛЬНЫЙ РАЗРЕЗ
ВХОД: граф G, весовая функция $w: A \to Z$, положительное целое число W
СВОЙСТВО: существует множество $S \subseteq N$ такое, что

$$\sum_{\substack{\{u,v\} \in A \\ u \in S \\ v \notin S}} w(\{u, v\}) \geq W.$$

Ясно, что все эти проблемы (или, более точно, их кодировки в Σ^*) принадлежат классу $\mathcal{NP}$. Приступим к осуществлению ряда явных сведе́ний, показывающих, что ВЫПОЛНИМОСТЬ сводится к любой из перечисленных проблем. На рис. 36.1 показана структура множества сведе́ний. Каждая линия рисунка указывает сводимость верхней проблемы к нижней.

Рис. 36.1. Полные проблемы

Для доказательства сводимости некоторого множества $\subseteq D$ к множеству $T' \subseteq D'$ мы определим функцию $F : D \to D'$, удовлетворяющую условиям леммы 2. Всякий раз читателю придется затратить некоторые усилия, чтобы убедиться, что F удовлетворяет этим условиям.

ВЫПОЛНИМОСТЬ ≤ 0-1 ЦЕЛОЧИСЛЕННОЕ ПРОГРАММИРОВАНИЕ

$$c_{ij} = \begin{cases} 1, & \text{если } x_j \in C_i \\ -1, & \text{если } \overline{x_j} \in C_i \\ 0 & \text{в противном случае} \end{cases}, \qquad i = 1,\ldots,p,\ \ j = 1,\ldots,n;$$

b_i = 1 – (число переменных с отрицанием в C_i), i = 1, 2, ..., p.

ВЫПОЛНИМОСТЬ ≤ КЛИКА

$N = \{\langle \sigma, i\rangle : \sigma$ есть буква, встречающаяся в $C_i\}$,
$A = \{\langle \sigma, i\rangle, \langle \delta, j\rangle : i \neq j$ и $\sigma \neq \bar{\delta}\}$
$k = p$, число предложений.

КЛИКА ≤ УПАКОВКА МНОЖЕСТВ

Положим $N = \{1, 2, \ldots, n\}$. Элементами множеств $S_1, \ldots, S_n$ являются те двухэлементные множества вершин $\{i, j\}$, которые не принадлежат A.

$S_i = \{\{i, j\} : \{i, j\} \notin A\}$, i = 1, 2, ..., n
$\ell = k$.

<...>

В заключение приведем список важных проблем из $\mathcal{NP}$, для которых неизвестно, полны они или нет. [Примечание редактора: сложность проблемы ИЗОМОРФИЗМ ГРАФОВ по-прежнему неизвестна, хотя в работе (László Babai 2016) достигнут значительный прогресс. Но теперь известно, что НЕПРОСТОТА ∈ $\mathcal{P}$ (Agrawal et al. 2004) и что ЛИНЕЙНЫЕ НЕРАВЕНСТВА ∈ $\mathcal{P}$ (Khachiyan 1979).]

ИЗОМОРФИЗМ ГРАФОВ
ВХОД: графы G и G'
СВОЙСТВО: G и G' изоморфны.

НЕПРОСТОТА
ВХОД: положительное целое число k
СВОЙСТВО: k составное (не простое) число.

ЛИНЕЙНЫЕ НЕРАВЕНСТВА
ВХОД: целочисленная матрица C, целочисленный вектор d
СВОЙСТВО: неравенство $Cx \geqslant d$ имеет рациональное решение.

37 Система с разделением времени Unix (1974)

Деннис Ритчи и Кеннет Томпсон

Совместимая система с разделением времени (CTSS) (глава 23) использовалась только в МТИ, но зато вдохновила огромное количество исследований и разработок по системам с разделением времени. За CTSS последовала Multics, которую МТИ даже пытался коммерциализировать. Одной из частных промышленных групп, принимавших участие в разработке Multics, была Bell Labs, накопившая опыт работы с компьютерными системами в процессе автоматизации телефонных систем. В конечном итоге Bell Labs вышла из проекта Multics, а несколько исследователей, занимавшихся им, решили построить собственную более простую учебную операционную систему, прежде всего чтобы упростить себе написание и отладку кода. Первоначально проектировавшаяся как однопользовательская система для работы на малом компьютере (отсюда и название – созвучное Multics, но противоположное по смыслу), Unix росла по мере того, как первоначальные разработчики и их коллеги находили для нее все больше применений в Bell Labs.

Bell Labs лицензировала Unix для использования за пределами компании, взимая очень небольшую плату с университетов. В результате Unix стала стандартом де-факто на факультетах компьютерных наук. Целые поколения лучших технических умов выходили из университетов со знанием ее структуры и с любовью к ее модульному, гибкому, ничем не отягощенному дизайну. Группа, трудившаяся над Unix, разработала язык программирования C, позволивший реализовать ее эффективно (оригинальная реализация была написана на языке ассемблера), и этот «высокоуровневый язык низкого уровня» стал неотъемлемой частью системного программирования. Также огромное влияние оказала иерархическая файловая система Unix (позаимствованная у Multics). В конечном итоге Unix легла в основу Berkeley Software

Distribution (BSD), проекта GNU («GNU's not Unix») Ричарда Столлмена, Linux Линуса Торвальдса и Mac OS компании Apple.

Деннис Ритчи (1941–2011) родился в семье ученого, работавшего в Bell Labs. Он изучал физику в Гарварде и информатику в аспирантуре. Докторскую диссертацию он написал и защитил в 1968 году. Но к тому времени он уже работал в Bell Labs и не нашел времени оформить окончательный вариант диссертации, поэтому степени PhD так и не получил. Ритчи работал в Bell Labs до конца своих дней. Кен Томпсон (родился в 1943 году) окончил Калифорнийский университет в Беркли, начал работать в Bell Labs в 1966 году и оставался там вплоть до выхода на пенсию в 2000 году, после чего перешел в Google. Ритчи и Томпсон отмечены премией Тьюринга за 1983 год – меньше чем через десять лет после публикации этого описания Unix.

Unix – многопользовательская интерактивная операционная система общего назначения, разработанная для компьютеров PDP-11/40 и 11/45 компании Digital Equipment Corporation (DEC). Она предлагает ряд возможностей, которые редко встретишь даже в более крупных операционных системах, в том числе: (1) иерархическая файловая система с возможностью монтирования и размонтирования томов; (2) совместимый ввод-вывод в файлы, на устройства и между процессами; (3) возможность запускать асинхронные процессы; (4) системный язык команд, выбираемый для себя каждым пользователем; (5) свыше 100 подсистем, включая дюжину языков. В этой работе обсуждаются природа и реализация файловой системы и командного интерфейса пользователя.

37.1. Введение

Существовало три версии Unix. Самая ранняя (примерно 1969–1970) работала на компьютерах Digital Equipment Corporation PDP-7 и -9. Вторая версия работала на незащищенном компьютере PDP-11/20. [Примечание редактора: т. е. компьютере, в котором не было механизмов защиты памяти для поддержки режима разделения времени.] В этой работе описывается только система для PDP-11/40 и /45 (DEC, 1972), поскольку она самая современная и между ней и предыдущими версиями Unix имеется много отличий вследствие перепроектирования средств, которые были сочтены неудачными или вовсе отсутствовали.

С тех пор как PDP-11 Unix заработала в феврале 1971 года, в эксплуатацию было введено примерно 40 установок; почти все они меньше, чем система, описанная здесь. В основном они используются для таких приложений, как подготовка и форматирование патентных заявок и других текстовых материалов, сбор и обработка данных о сбоях различных коммутаторов в компании Bell System, а также для регистрации и проверки заказов на оказание услуг телефонии. Наша собственная установка используется в основном для

исследований в области операционных систем, языков, вычислительных сетей и других вопросов информатики, а также для подготовки документов.

Быть может, самое важное достижение Unix – демонстрация того, что мощная операционная система для интерактивного использования необязательно должна быть дорогой с точки зрения затрат на оборудование и персонал; Unix может работать на оборудовании ценой всего 40 000 долларов, а на создание основного системного программного обеспечения было потрачено менее двух человеко-лет. И тем не менее Unix содержит ряд функций, которые редко предлагаются даже в гораздо более крупных системах. Однако мы надеемся, что пользователи Unix согласятся, что самыми важными характеристиками системы являются простота, элегантность и легкость использования. <...>

37.2. Аппаратное и программное окружение

<...>

37.3. Файловая система

Самая важная работа Unix – предоставить файловую систему. С точки зрения пользователя, существует три вида файлов: обыкновенные дисковые файлы, каталоги и специальные файлы.

37.3.1. Обыкновенные файлы. Такой файл содержит ту информацию, которую поместил в него пользователь, например символические или двоичные (объектные) программы. Система не ожидает от файла никакой специальной структуры. Текстовые файлы представляют собой просто последовательность символов, разбитую на строки символами перехода на новую строку. Двоичные программы – это последовательности слов в том виде, в каком они оказываются в оперативной памяти, когда программа начинает исполнение. Некоторые пользовательские программы работают с файлами, наделенными более богатой структурой: ассемблер генерирует, а загрузчик ожидает получить объектный файл в определенном формате. Однако структура файлов контролируется программами, которые их используют, а не самой системой.

37.3.2. Каталоги. Каталоги устанавливают соответствие между именами файлов и самими файлами и, таким образом, вводят структуру в файловую систему в целом. Каждый пользователь имеет каталог собственных файлов; он также может создавать подкаталоги, содержащие группы файлов, которые удобно рассматривать вместе. Каталог ведет себя как обыкновенный файл, с тем отличием, что записывать в него могут только привилегированные программы, т. е. содержимое каталогов контролируется системой. Однако любой пользователь, имеющий необходимые права, может читать каталог, как любой другой файл.

Система поддерживает несколько каталогов для своих собственных нужд. Один из них – корневой каталог. Любой хранящийся в системе файл можно найти, следуя по цепочке каталогов. Начальной точкой для всех таких поисков часто является корень. Другой системный каталог содержит все программы, предназначенные для общего пользования, т. е. все *команды*. Однако, как мы увидим, программа вовсе не обязана находиться в этом каталоге, чтобы ее можно было выполнить.

Файлы именуются последовательностями, содержащими 14 символов или менее. Имя файла, сообщаемое системе, должно иметь форму *путевого имени*, т. е. представлять собой последовательность имен каталогов, разделенных знаками косой черты «/», которая заканчивается именем файла. Если последовательность начинается знаком /, то поиск начинается с корневого каталога. Имя */alpha/beta/gamma* заставляет систему искать в корне каталог *alpha*, затем искать в *alpha* каталог *beta* и, наконец, искать в *beta* имя *gamma*. *Gamma* может быть обыкновенным файлом, каталогом или специальным файлом. Самое короткое из возможных имя «/» относится к самому корню.

Если путевое имя не начинается с «/», то система начинает поиск в текущем каталоге пользователя. Таким образом, имя *alpha/beta* определяет файл *beta* в подкаталоге *alpha* текущего каталога. Простейший вид имени, например *alpha*, относится к файлу, который находится прямо в текущем каталоге. Другой крайний случай, пустое имя, относится к текущему каталогу.

Один и тот же файл, не являющийся каталогом, может находиться в нескольких каталогах, возможно, под разными именами. Эта возможность называется *созданием ссылок*; запись о файле в каталоге иногда называют *ссылкой*. Unix отличается от других систем, в которых разрешены ссылки, тем, что все ссылки на файл равноправны. То есть файл не закреплен за каким-то конкретным каталогом; запись о файле в каталоге содержит просто его имя и ссылку на информацию, фактически описывающую файл. Таким образом, файл существует независимо от записей в каталогах, хотя на практике файл исчезает после удаления последней ссылки на него.

В каждом каталоге всегда имеется по крайней мере две записи. Имя «.» ссылается на сам каталог. Таким образом, программа может прочитать текущий каталог по имени «.», даже не зная его полного путевого имени. Имя «..», по соглашению, ссылается на родителя того каталога, в котором находится, т. е. на каталог, в котором было создано.

Структура каталогов имеет форму дерева с одним корнем. За исключением специальных записей «.» и «..», каждый каталог должен встречаться в виде записи ровно в одном каталоге, который является его родителем. Причина – упростить написание программ, которые посещают поддеревья структуры каталогов, и, что еще важнее, избежать отделения частей иерархии. Если бы были разрешены произвольные ссылки на каталоги, то было бы очень трудно обнаружить перерезание последней связи корня с каталогом.

37.3.3. Специальные файлы. Специальные файлы – самая необычная особенность файловой системы Unix. С каждым устройством ввода-вывода, поддерживаемым Unix, ассоциирован по крайней мере один такой файл. Специальные файлы можно читать и записывать так же, как обыкновенные дисковые

файлы, но запросы на чтение или запись приводят к активации соответствующего устройства. Записи обо всех специальных файлах находятся в каталоге */dev*, хотя на такие файлы можно создавать ссылки, как на обыкновенные файлы. Так, например, для перфорации бумажной ленты можно произвести запись в файл */dev/ppt*. Специальные файлы существуют для всех линий связи, для всех дисков, для всех ленточных накопителей и для физической оперативной памяти. Конечно, специальные файлы, ассоциированные с активными дисками и памятью, защищены от непривилегированного доступа.

У такой трактовки устройств ввода-вывода есть три преимущества: файл и устройство ввода-вывода сделаны настолько похожими, насколько это возможно; имена файлов и устройств имеют одинаковый синтаксис и семантику; наконец, к специальным файлам применяется тот же механизм защиты, что и к регулярным.

37.3.4. Съемные файловые системы <...>

37.3.5. Защита. Хотя схема контроля доступа в Unix очень проста, у нее есть некоторые необычные черты. Каждому пользователю системы присваивается уникальный идентификационный номер. В момент создания файл помечается идентификатором пользователя, являющегося его владельцем. Кроме того, для новых файлов задается семь бит защиты. Шесть из них независимо определяют права чтения, записи и выполнения для владельца файла и всех остальных пользователей.

Если седьмой бит поднят, то при каждом выполнении файла система временно изменяет идентификатор текущего пользователя, делая его равным идентификатору создателя файла. Такое изменение идентификатора действует только на время выполнения программы, хранящейся в нем. Функция смены идентификатора пользователя (set-user-id) открывает возможность создавать привилегированные программы, которые могут использовать файлы, недоступные другим пользователям. Например, программа может вести учетный файл, который запрещено читать и изменять всем остальным программам. Если для программы установлен бит «set-user-id», то она может обратиться к файлу, пусть даже доступ к нему запрещен другим программам, вызываемым от имени пользователя с данным идентификатором. Поскольку фактический идентификатор пользователя, вызвавшего программу, всегда доступен, программа с установленным битом «set-user-id» может предпринять любые меры, которые сочтет нужным, чтобы проверить учетные данные того, кто ее вызвал. Этот механизм используется, чтобы разрешить пользователям выполнять аккуратно написанные команды, которые обращаются к привилегированным системным вызовам. Например, системный вызов, создающий пустой каталог, доступен только «суперпользователю» (см. ниже). Как уже было сказано выше, ожидается, что любой каталог содержит запись для «.» и «..». Командой создания каталога владеет суперпользователь, и для нее установлен бит «set-user-id». Убедившись, что пользователю, вызвавшему ее, разрешено создавать указанный каталог, она создает его и помещает в него запись для «.» и «..».

Поскольку любой пользователь может установить бит «set-user-id» для своих файлов, этот механизм обычно доступен без вмешательства со стороны

администратора. Например, эта схема защиты легко решает проблему учета в многопользовательской игре MOO, описанную в работе Aleph-Null (1971).

Система считает один конкретный идентификатор пользователя (а именно «суперпользователя») исключением из общих правил ограничения доступа к файлам; это позволяет (например) записывать дамп памяти программ и перезагружать файловую систему без нежелательного вмешательства со стороны системы защиты.

37.3.6. Вызовы ввода-вывода. Системные вызовы для выполнения ввода-вывода спроектированы так, чтобы устранить различия между разнообразными устройствами и стилями доступа. Не существует различия между «произвольным» и последовательным вводом-выводом, и система не навязывает никакого размера логической записи. Размер обыкновенного файла определяется последним записанным в него байтом; предварительное задание размера файла необязательно и даже невозможно. Для иллюстрации основ ввода-вывода в Unix ниже перечислены некоторые базовые системные вызовы на безымянном языке, который позволяет описать обязательные параметры, не вдаваясь в сложности программирования на машинном языке. Каждый вызов системы потенциально может вернуть ошибку, но для простоты они не показаны в последовательности вызова. Чтобы прочитать или записать в уже существующий файл, его необходимо сначала открыть с помощью следующего вызова:

```
filep = open (name, flag)
```

Здесь *name* – имя файла. Можно указать произвольное путевое имя. Аргумент *flag* говорит, что сделать с файлом: прочитать, записать или «обновить», т. е. читать и записывать. Возвращенное значение *filep* называется дескриптором файла. Это небольшое целое число, которое позволяет идентифицировать файл в последующих вызовах для чтения, записи или выполнения иных операций.

Для создания нового файла или полного перезаписывания старого имеется системный вызов *create*, который создает указанный файл, если его не существует, или усекает до нулевой длины, если он существует. *Create* также открывает новый файл для записи и, как и *open*, возвращает дескриптор файла.

В файловой системе не существует видимых пользователю блокировок и нет никаких ограничений на количество пользователей, открывающих файл для чтения или записи; хотя теоретически содержимое файла может быть искажено, когда два пользователя пишут в него одновременно, на практике таких трудностей не возникает. Мы считаем, что в нашем окружении блокировки не являются ни необходимыми, ни достаточными, чтобы предотвратить взаимовлияние пользователей, работающих с одним и тем же файлом. Они не являются необходимыми, потому что мы не сталкиваемся с большими однофайловыми базами данных, к которым обращаются независимые процессы. А достаточными они не могут считаться, потому что блокировки в общепринятом смысле, когда один пользователь не может писать в файл, который другой читает, не способны предотвратить недоразумения, когда, например, оба пользователя редактируют файл в редакторе, кото-

рый делает копию редактируемого файла. Следует добавить, что в системе имеется достаточно внутренних механизмов блокировки для поддержания логической согласованности файловой системы в случаях, когда два пользователя одновременно выполняют такие вызывающие затруднения операции, как запись в один и тот же файл, создание файлов в одном и том же каталоге или удаление файлов, открытых другим пользователем.

Кроме случаев, оговоренных ниже, чтение и запись производятся последовательно. Это означает, что если некоторый байт в файле был последним записанным (или прочитанным), то следующая операция неявно обращается к первому следующему за ним байту. Для каждого открытого файла система поддерживает указатель на следующий байт, подлежащий чтению или записи. Если прочитано или записано *n* байт, то указатель сдвигается вперед на *n* байт.

После открытия файла можно выполнять следующие вызовы:

n = read(filep, buffer, count)

n = write(filep, buffer, count)

Между файлом, заданным дескриптором *filep*, и байтовым массивом, заданным аргументом *buffer*, передается не более *count* байт. Возвращенное значение *n* равно числу фактически переданных байтов. В случае записи *n* совпадает с *count*, если только не возникли исключительные условия, например ошибка ввода-вывода или исчерпание места на физическом носителе, определяемом специальным файлом. Однако в случае чтения *n* может оказаться меньше *count*, даже если ошибки не было. Если указатель чтения находится так близко к концу файла, что чтение *count* символов привело бы к выходу за границу файла, то передается лишь столько байтов, сколько осталось до конца файла; кроме того, устройства типа пишущей машинки не могут возвращать больше символов, чем помещается в одной строке ввода. Если вызов *read* возвращает *n*, равное нулю, значит, достигнут конец файла. Для дисковых файлов это бывает, когда указатель чтения равен текущему размеру файла. Конец файла при чтении с пишущей машинки можно сгенерировать с помощью управляющей последовательности, которая зависит от конкретного устройства.

Запись байтов в файл затрагивает только те байты, которые определяются позицией указателя записи и счетчиком; больше никакие части файла не изменяются. Если последний байт оказывается за концом файла, то размер файла соответственно увеличивается.

Для выполнения произвольного ввода-вывода (с прямым доступом) необходимо только переместить указатель чтения или записи в нужное место файла:

location = seek(filep, base, offset)

Указатель, ассоциированный с *filep*, перемещается в позицию, отстоящую на *offset* байт от начала файла, от текущей позиции указателя или от конца файла, в зависимости от значения *base*. Аргумент *offset* может быть отрицательным. Для некоторых устройств (например, бумажной ленты и пишущих

машинок) вызовы *seek* игнорируются. В *location* возвращается фактическое смещение от начала файла позиции, в которой оказался указатель.

37.3.6.1. Вызовы ввода-вывода. Имеются дополнительные системные вызовы, относящиеся к вводу-выводу и файловой системе, которые здесь не обсуждаются, например закрыть файл, получить состояние файла, изменить режим защиты или владельца файла, создать каталог, создать ссылку на существующий файл, удалить файл.

37.4. Реализация файловой системы

Как было сказано в § 37.3.2 выше, запись в каталоге содержит только имя соответствующего файла и указатель на сам файл. Этот указатель представляет собой целое число, называемое *i-номером* (индексным номером) файла. При обращении к файлу его *i-номер* используется как индекс в системную таблицу (*i-список*), хранящуюся в известной части устройства, на котором находится каталог. Найденная в результате запись (*индексный дескриптор*, или *i-node* файла) содержит следующее описание файла.

1. Его владелец.
2. Его биты защиты.
3. Адреса содержимого файла на диске или ленте.
4. Его размер.
5. Время последней модификации.
6. Число ссылок на файл, т. е. сколько раз он встречается в каталоге.
7. Бит, показывающий, является ли файл каталогом.
8. Бит, показывающий, является ли файл специальным файлом.
9. Бит, показывающий, является ли файл «большим» или «малым».

Задача системного вызова *open* или *create* – преобразовать путевое имя, заданное пользователем, в i-номер, найдя явно или неявно поименованные каталоги. После того как файл открыт, его устройство, i-номер и указатель чтения-записи сохраняются в записи системной таблицы с индексом, равным дескриптору файла, возвращенному *open* или *create*. Таким образом, дескриптор файла, переданный последующему системному вызову для чтения или записи файла, можно легко связать с информацией, необходимой для доступа к файлу.

При создании нового файла для него выделяется индексный дескриптор и в каталоге создается запись, содержащая имя файла и номер индексного дескриптора. При создании ссылки на существующий файл необходимо создать в каталоге запись с новым именем, скопировать в нее i-номер из записи об оригинальном файле и увеличить на единицу поле счетчика ссылок в индексном дескрипторе. Для удаления файла уменьшается на единицу счетчик ссылок в индексном дескрипторе, который описывается записью о нем в каталоге, и стирается запись в каталоге. Если счетчик ссылок обратился в 0, то все блоки, занятые файлом на диске, освобождаются, как и его индексный дескриптор.

Место на всех фиксированных или съемных дисках, содержащих файловую систему, разбито на 512-байтовые блоки, логически адресуемые от 0 и до верхнего предела, зависящего от устройства. В индексном дескрипторе каждого файла есть место для восьми адресов устройства. *Малый* (неспециальный) файл помещается в восемь или менее блоков; в этом случае хранятся адреса самих блоков. Для *больших* (неспециальных) файлов каждый из восьми адресов устройства может указывать на косвенные блоки, содержащие 256 адресов блоков самого файла. Такие файлы могут занимать до $8 \cdot 256 \cdot 512 = 1\,048\,576$ (2^{20}) байт.

Предшествующее обсуждение относилось к обыкновенным файлам. Если сделан запрос ввода-вывода к файлу, в индексном дескрипторе которого указано, что он специальный, то последние семь слов, содержащих адреса устройства, не важны, а список интерпретируется как пара байтов, составляющих внутреннее имя *устройства*. Эти байты определяют соответственно тип устройства и номер подустройства. Тип устройства указывает, какая системная процедура отвечает за ввод-вывод для этого устройства; номер подустройства выбирает, например, дисковод, подключенный к конкретному контроллеру или одному из нескольких схожих интерфейсов пишущей машинки.

В этом окружении реализация системного вызова *mount* (§ 37.3.4) вполне прямолинейна. *Mount* поддерживает системную таблицу, в которой ключом является i-номер и имя устройства обыкновенного файла, заданного при обращении к *mount*, а значением – имя устройства указанного специального файла. В этой таблице производится поиск каждой пары (i-номер, устройство), встретившейся при просмотре путевого имени, указанного в *open* или *create*; если соответствие найдено, то i-номер заменяется на 1 (это i-номер корневого каталога во всех файловых системах), а имя устройства – значением, прочитанным из таблицы.

Пользователю обе операции – чтение и запись файлов – представляются синхронными и небуферизованными. То есть данные становятся доступны сразу после возврата из вызова *read* и наоборот – сразу после вызова *write* пользовательскую рабочую область можно использовать повторно. На самом деле система поддерживает довольно сложный механизм буферизации, который значительно уменьшает количество операций ввода-вывода, необходимых для доступа к файлу. Предположим, что выполнен вызов *write*, требующий передачи одного байта.

Unix произведет поиск в своих буферах, чтобы узнать, находится ли уже требующий изменения дисковый блок в оперативной памяти; если нет, его необходимо прочитать с устройства. Затем нужный байт заменяется в буфере и вносится элемент в список блоков, подлежащих записи на диск. После этого может быть произведен возврат из вызова *write*, хотя фактический ввод-вывод, возможно, будет завершен спустя некоторое время. Обратно, если читается один байт, то система определяет, находится ли уже в ее буферах тот блок внешней памяти, в котором этот байт расположен. Если да, то байт можно вернуть немедленно. Если нет, то блок читается в буфер, и оттуда выбирается нужный байт.

Программа, которая читает и записывает файлы блоками по 512 байт, превосходит программу, читающую или записывающую по одному байту за

раз, но выигрыш не так уж велик; он проистекает в основном из упразднения системных накладных расходов. Программа, используемая редко или выполняющая умеренный объем ввода-вывода, вполне может читать и записывать сколь угодно малыми блоками.

Идея i-списка – необычная особенность Unix. На практике такой метод организации файловой системы доказал высокую надежность и простоту использования. Для самой системы одно из его преимуществ заключается в том, что у каждого файла имеется короткое, недвусмысленное имя, которое простым способом соотносится с защитой, адресацией и прочей информацией, необходимой для доступа к файлу. Он также допускает простой и быстрый алгоритм проверки согласованности файловой системы, например что те части каждого устройства, которые содержат полезную информацию, и те, что свободны и доступны для выделения, не пересекаются и в совокупности исчерпывают все имеющееся на устройстве место. Этот алгоритм не зависит от иерархии каталогов, поскольку должен просматривать только линейно организованный i-список. В то же время идея i-списка привносит некоторые тонкости, не встречающиеся при других способах организации файловой системы.

Например, возникает вопрос, кто должен платить за место, занятое файлом, потому что все записи каталога, в которых этот файл встречается, равноправны. Взимать плату с владельца файла в общем случае несправедливо, потому что один пользователь может создать файл, затем другой создаст на него ссылку, после чего первый пользователь может файл удалить. Владельцем файла остается первый пользователь, но начислять плату за него нужно второму. Простейший более-менее справедливый алгоритм, по-видимому, состоит в том, чтобы равномерно распределять плату между всеми пользователями, создавшими ссылки на файл. В текущей версии Unix эта проблема не возникает, потому что плата не взимается вовсе.

37.4.1. Эффективность файловой системы <...>

37.5. Процессы и образы

Образ – это окружение, в котором компьютер производит выполнение. Он включает образ оперативной памяти, значения общих регистров, состояние открытых файлов, текущий каталог и т. п. Образ – это текущее состояние псевдокомпьютера.

Процесс – это исполнение образа. Когда процессор выполняет инструкции от имени процесса, образ должен находиться в оперативной памяти; во время выполнения других процессов он остается в памяти, если только появление активного высокоприоритетного процесса не вынуждает выгрузить его на фиксированный диск.

Часть образа, находящаяся в пользовательской памяти, делится на три логических сегмента. Сегмент текста программы начинается с адреса 0 в виртуальном адресном пространстве. Во время выполнения этот сегмент защищен от записи и существует его единственная копия, разделяемая между всеми

процессами, исполняющими одну и ту же программу. На первой границе, кратной 8K байтам, после сегмента текста программы в виртуальном адресном пространстве начинается неразделяемый, допускающий запись сегмент данных, размер которого можно увеличить с помощью системного вызова. Начиная с самого старшего адреса в виртуальном адресном пространстве расположен сегмент стека, который автоматически растет вниз при изменении аппаратного указателя стека.

37.5.1. Процессы. Если не считать начальной загрузки самой Unix, то новый процесс появляется на свет только в результате системного вызова *fork*:

processid = fork (label)

Когда процесс вызывает *fork*, он расщепляется на два независимо исполняемых процесса. Оба они имеют независимые копии исходного образа в памяти и разделяют все открытые файлы. Два процесса различаются только тем, что один из них считается родительским: в родительском процессе управление возвращается прямо из *fork*, а в дочернем передается по адресу *label*. Значение *processid*, возвращенное вызовом *fork*, – идентификатор другого процесса. Поскольку точки возврата в родительском и дочернем процессах различаются, каждый образ, который оставляет за собой *fork*, может определить, является процесс родительским или дочерним.

37.5.2. Каналы. Процессы могут взаимодействовать со связанными процессами, применяя те же системные вызовы *read* и *write*, которые используются для ввода-вывода в файловой системе. Вызов

filep = pipe( )

возвращает файловый дескриптор *filep* и создает межпроцессный *канал*. Этот канал, как и другие открытые файлы, передается от родительского процесса дочернему в образе, созданном вызовом *fork*. Операция чтения из файлового дескриптора канала ждет, пока другой процесс что-то запишет, пользуясь дескриптором того же канала. В этот момент данные передаются между образами обоих процессов. Ни одному процессу не нужно знать, что он имеет дело с каналом, а не обыкновенным файлом.

Хотя межпроцессное взаимодействие посредством каналов – весьма ценный инструмент (см. § 37.6.2), этот механизм недостаточно общий, потому что канал должен быть организован общим предком взаимодействующих по нему процессов.

37.5.3. Исполнение программ. Еще один важный системный примитив

execute(file, arg_1, arg_2, ..., arg_n)

просит систему прочитать и выполнить программу, находящуюся в файле *file*, передав ей строковые аргументы arg_1, arg_2, ..., arg_n. Обычно arg_1 должен совпадать с *file*, так чтобы программа могла определить, под каким именем она вызвана. Весь код и данные процесса, выполнившего *execute*, заменяются кодом и данными, прочитанными из файла, но открытые файлы, текущий каталог и межпроцессные связи не изменяются. Только если вызов

завершается неудачно, например потому что не удалось найти файл или для него не установлен бит выполнения, системный вызов *execute* возвращает управление; это напоминает скорее машинную команду «jump», а не вызов подпрограммы.

37.5.4. Синхронизация процессов. Еще один системный вызов для управления процессами

processid = wait( )

заставляет вызывающий процесс приостановить выполнение, пока один из дочерних процессов не закончит работу. Затем *wait* возвращает идентификатор завершившегося процесса. Если у вызывающего процесса нет дочерних, возвращается код ошибки. Можно также получить определенную информацию о состоянии дочернего процесса. Иногда *wait* предоставляет состояние внука или еще более отдаленного потомка; см. § 37.5.5.

37.5.5. Завершение. Наконец, вызов

exit (status)

завершает процесс, уничтожает его образ, закрывает все его открытые файлы и вообще стирает все следы его присутствия. Если родитель уведомляется посредством примитива *wait*, то ему передается значение *status*; если родитель к этому моменту уже завершился, то состояние делается доступным деду и т. д. Процессы могут также завершаться в результате различных недопустимых действий или получения сгенерированных пользователем сигналов (§ 37.7 ниже).

37.6. Оболочка

Для большинства пользователей взаимодействие с Unix опосредовано программой, называемой оболочкой. Оболочка – это интерпретатор командной строки: она читает строки, введенные пользователем, и интерпретирует их как запросы на выполнение других программ. В простейшей форме командная строка состоит из имени команды, за которым следуют аргументы команды, разделенные пробелами:

command arg_1 arg_2 ... arg_n

Оболочка выделяет имя команды и аргументы, представляя их отдельными строками. Затем она ищет файл с именем *command*; *command* может быть путевым именем, включающим символ «/», и задавать любой файл в системе. Если файл *command* найден, он загружается в память и выполняется. Аргументы, разобранные оболочкой, делаются доступными команде. По завершении команды оболочка возобновляет выполнение и индицирует готовность к приему следующей команды, печатая символ приглашения.

Если файл *command* не найден, оболочка дописывает строку /bin/ в начало команды и пробует найти файл еще раз. Каталог /bin содержит все команды, предназначенные для использования всеми желающими.

37.6.1. Стандартный ввод-вывод. После обсуждения ввода-вывода в § 37.3 выше могло сложиться впечатление, что любой файл, используемый программой, необходимо сначала открыть или создать, чтобы получить дескриптор файла. Однако программы, запускаемые оболочкой, начинают выполнение с двумя открытыми файлами, имеющими дескрипторы 0 и 1. При этом файл 1 открыт для записи и его проще всего считать стандартным выводом. За исключением случаев, описанных ниже, этот файл является пишущей машинкой пользователя. Таким образом, программы, которые хотят вывести информационное или диагностическое сообщение, обычно используют файловый дескриптор 1. С другой стороны, файл 0 изначально открыт для чтения, и его обычно используют программы, желающие читать сообщения, введенные пользователем. Оболочка может изменить стандартные назначения этим файловым дескрипторам – пишущую машинку и клавиатуру. Если одному из аргументов команды предшествует знак «>», то файловый дескриптор 1 на все время выполнения этой команды будет связан с файлом, имя которого указано после «>». Например, команда

```
ls
```

обычно выводит на пишущую машинку список имен файлов в текущем каталоге. А команда

```
ls >there
```

создает файл *there* и помещает список в него. Таким образом, аргумент «>there» означает «помести вывод в there». С другой стороны, команда

```
ed
```

обычно запускает редактор, который принимает команды пользователя, набранные на пишущей машинке. Команда же

```
ed <script
```

интерпретирует *script* как файл, содержащий команды редактора; именно «<script» означает «возьми ввод из script».

Хотя имя файла, следующее за знаком «<» или «>», выглядит как аргумент команды, на самом деле оно интерпретируется оболочкой и вообще не передается команде. Поэтому внутри команды не нужно как-то специально обрабатывать перенаправление ввода-вывода; команда просто должна использовать стандартные дескрипторы файлов 0 и 1 там, где это необходимо.

37.6.2. Фильтры. Расширение идеи стандартного ввода-вывода позволяет направить вывод одной команды на вход другой. Последовательность команд, разделенных вертикальной чертой, заставляет оболочку запустить все команды одновременно и связать стандартный вывод каждой команды (кроме последней) со стандартным вводом следующей. Так, в командной строке

```
ls | pr -2 | opr
```

ls выводит список файлов в текущем каталоге, а ее вывод передается *pr*, которая разбивает свой ввод на страницы и добавляет заголовки с датой. Аргумент «-2» означает печать в две колонки. Аналогично вывод *pr* подается

на ввод *opr*. Эта команда копирует свой ввод в файл в каталоге спулинга для последующей печати.

Этот процесс можно было бы записать более громоздко:

```
ls > temp1
pr -2 <temp1 >temp2
opr <temp2
```

а затем удалить временные файлы. Если бы не было возможности перенаправлять ввод и вывод, то пришлось бы применить еще более громоздкий метод – потребовать, чтобы команда *ls* принимала запросы пользователя на разбиение вывода на страницы, форматирование их в несколько колонок и организацию последующей печати результата. Было бы странно и даже глупо, по причинам, связанным с эффективностью, требовать от авторов таких команд, как *ls*, предоставления столь широкого спектра опций.

Программа типа *pr*, которая копирует свой стандартный ввод на стандартный вывод (после обработки), называется *фильтром*. Некоторые фильтры, которые мы сочли полезными, выполняют транслитерацию символов, сортировку ввода, шифрование и дешифрирование.

37.6.3. Разделители команд: многозадачность. Еще одна возможность, предоставляемая оболочкой, довольно проста. Команды необязательно должны размещаться в разных строках, вместо этого их можно разделять точками с запятой. Команда

```
ls; ed
```

сначала печатает содержимое текущего каталога, а затем запускает редактор.

Более интересна другая связанная с этим возможность. Если за командой следует знак «&», то оболочка не ждет завершения команды, прежде чем вывести приглашение, а готова принимать новую команду сразу же. Например, команда

```
as source > output &
```

запускает ассемблирование файла *source* и выводит диагностическую информацию в *output*; вне зависимости от того, сколько времени займет ассемблирование, оболочка возвращается немедленно. Если оболочка не ждет завершения команды, то печатается идентификатор процесса, исполняющего эту команду. Впоследствии этот идентификатор можно использовать, чтобы дождаться завершения команды или снять ее. Знак «&» можно использовать в строке несколько раз. Команда

```
as source > output & ls > files &
```

выполняет и ассемблирование, и вывод списка файлов в фоновом режиме. В приведенных выше примерах использования «&» вывод направлялся в файл, а не на печатную машинку; в противном случае вывод разных команд беспорядочно чередовался бы.

Оболочка также допускает использование скобок. Например, команда

```
(date; ls) > x &
```

печатает текущую дату и время, а затем список файлов в текущем каталоге – и все это направляется в файл *x*. При этом оболочка сразу же возвращается, чтобы принять следующий запрос.

37.6.4. Оболочка как команда: командные файлы. Оболочка сама является командой, которую можно вызывать рекурсивно. Предположим, что файл *tryout* содержит строки

```
as source
mv a.out testprog
testprog
```

Команда *mv* переименовывает файл *a.out* в *testprog*. Файл *a.out* – это (двоичный) вывод ассемблера, готовый к выполнению. Таким образом, если ввести все три строки на консоли, то файл *source* будет ассемблирован, результирующая программа получит имя *testprog* и файл *testprog* будет выполнен. Если эти строки находятся в файле *tryout*, то команда

```
sh < tryout
```

заставит оболочку *sh* выполнить команды последовательно.

У оболочки есть и другие возможности, например она умеет подставлять параметры и строить списки аргументов по заданному подмножеству имен файлов в каталоге. Можно также выполнять команды условно в зависимости от результата сравнения строк или существования определенных файлов и передавать управление внутри файла команд.

37.6.5. Реализация оболочки. Теперь можно в общих чертах описать механизм работы оболочки. Большую часть времени оболочка ждет ввода команды пользователем. Получив символ новой строки, завершающий строку, вызванная оболочкой функция *read* возвращает управление. Оболочка анализирует командную строку и преобразует аргументы в форму, пригодную для выполнения. Затем она вызывает *fork*. Дочерний процесс, который, конечно же, тоже является оболочкой, пытается вызвать *execute* с подходящими аргументами. Если все хорошо, то будет загружена и запущена программа с указанным именем. А тем временем другой процесс, образовавшийся в результате выполнения *fork*, т. е. родительский процесс, ждет завершения дочернего. Когда это произойдет, оболочка будет знать, что команда завершилась, поэтому напечатает приглашение и будет читать следующую команду с пишущей машинки.

В рамках этой общей схемы реализация фоновых процессов тривиальна; когда в командной строке встречается символ «&», оболочка не ждет завершения процесса, созданного ей для выполнения команды.

Весь этот механизм очень хорошо сочетается с понятиями файлов стандартного ввода и вывода. Процесс, созданный примитивом *fork*, наследует не только образ своего родителя в оперативной памяти, но также и все файлы, открытые родителем, в т. ч. с дескрипторами 0 и 1. Конечно, оболочка использует эти файлы для чтения командных строк и записи приглашений и диагностических сообщений, и в базовом случае ее потомки – программы, реализующие команды, – наследуют их автоматически. Но если заданы

аргументы, содержащие «<» или «>», то дочерний процесс, прежде чем вызвать *execute*, связывает дескриптор ввода-вывода 0 или 1 с поименованным файлом. Это легко, потому что, по соглашению, при открытии вызовом *open* (или создании вызовом *create*) нового файла ему назначается наименьший неиспользуемый дескриптор; поэтому необходимо только закрыть файл 0 (или 1) и открыть поименованный файл. Поскольку процесс, в котором выполняется команда, по окончании работы просто завершается, ассоциация между файлом, указанным после «<» или «>», и файловым дескриптором 0 или 1 автоматически разрывается при уничтожении процесса. Поэтому оболочке не нужно знать истинных имен файлов, связанных с ее собственным стандартным вводом и выводом, т. к. ей никогда не приходится заново открывать их.

Фильтры реализованы прямолинейно, как расширения перенаправления стандартного ввода-вывода, только вместо файлов используются каналы.

При обычных обстоятельствах главный цикл оболочки никогда не завершается. (Главный цикл включает ветвь *fork*, относящуюся к родительскому процессу, т. е. ту ветвь, которая вызывает *wait*, а затем читает следующую командную строку.) Завершиться оболочка может, например, встретив конец входного файла. Таким образом, когда оболочка выполняется как команда с заданным входным файлом, например

```
sh < comfile
```

команды в файле *comfile* будут выполняться до достижения конца *comfile*, после чего экземпляр оболочки, запущенный *sh*, завершится. Так как этот процесс оболочки является дочерним по отношению к другому экземпляру оболочки, вызов *wait* в последнем вернет управление, и может начаться обработка следующей команды.

37.6.6. Инициализация. Экземпляры оболочки, в которых пользователи вводят команды, сами являются потомками некоторого процесса. На последнем шаге инициализации Unix создается один процесс, в котором исполняется (с помощью *execute*) программа с именем *init*. Роль *init* заключается в том, чтобы создать по одному процессу для каждого канала пишущей машинки, к которому может подключиться пользователь. Дочерние процессы *init* открывают соответствующие пишущие машинки для ввода и вывода. Поскольку в момент вызова *init* еще не было открытых файлов, в каждом процессе клавиатура пишущей машинки получит файловый дескриптор 0, а ее печатающее устройство – дескриптор 1. Каждый процесс печатает сообщение, предлагающее пользователю войти, и ждет ответа, читая с пишущей машинки. В самом начале ни один пользователь еще не вошел в систему, поэтому каждый процесс просто висит. Наконец кто-то вводит свое имя или иной идентификатор. Соответствующий экземпляр *init* просыпается, получает данные с подключенной линии и читает файл паролей. Если имя пользователя в нем найдено и если пароль правилен, то *init* переходит в текущий каталог по умолчанию данного пользователя; в качестве идентификатора пользователя, связанного с процессом, устанавливает идентификатор вошедшего пользователя и вызывает *execute* для оболочки. В этот момент обо-

лочка готова принимать команды, и протокол входа в систему можно считать завершенным.

Тем временем главный экземпляр *init* (родитель всех дочерних экземпляров, ставших впоследствии оболочками) вызывает *wait*. Если какой-нибудь дочерний процесс завершится – потому ли, что оболочка встретила конец файла, или потому, что пользователь ввел неправильное имя или пароль, то этот экземпляр просто заново создаст почивший процесс, который, в свою очередь, заново откроет файлы ввода и вывода и напечатает приглашение войти. Таким образом, пользователь может выйти из системы, просто набрав последовательность символов, означающую конец файла, вместо команды оболочки.

37.6.7. Другие программы в роли оболочки. Описанная выше оболочка предоставляет пользователям полный доступ к средствам системы, поскольку умеет выполнять любую программу с соблюдением мер защиты. Однако иногда желателен другой интерфейс к системе, и это легко организовать.

Напомним, что после того как пользователь успешно вошел в систему, введя свое имя и пароль, init запускает оболочку для интерпретации командных строк. В записи о пользователе в файле паролей может быть указано имя программы, запускаемой после входа в систему вместо стандартной оболочки. Эта программа вправе интерпретировать сообщения, введенные пользователем, как ей будет угодно.

Например, для пользователей из секретариата в файле паролей вместо оболочки может быть указан редактор *ed*. Тогда сразу после входа пользователь оказывается в редакторе и может немедленно приступить к работе; заодно ему запрещено запускать программы Unix, не связанные с его функциями. На практике оказалось желательно разрешить временный выход из редактора для запуска программ форматирования и других утилит.

Некоторые игры (например, шахматы, блэкджек, трехмерные крестики и нолики), имеющиеся в Unix, могут служить примером гораздо более строго ограниченного окружения. Для каждой из них в файле паролей имеется запись, в которой указано, какую игровую программу следует запустить вместо оболочки. Люди, входящие в систему, чтобы поиграть в одну из таких игр, оказываются ограниченными только этой игрой и не могут исследовать другие возможности, предлагаемые Unix, – быть может, более интересные.

37.7. Ловушки

Оборудование PDP-11 обнаруживает ряд программных ошибок, например обращение к несуществующей памяти, нереализованные инструкции и нечетные адреса там, где ожидаются четные. При возникновении такой ошибки процессор переходит к системной процедуре-ловушке. Если перехвачено недопустимое действие и не предусмотрено никаких других мер, то система завершает процесс и записывает образ пользовательской памяти в файл *core* в текущем каталоге. Для исследования состояния программы в момент ошибки можно использовать отладчик.

Программы, которые зациклились, порождают нежелательный вывод или вызывают у пользователя сомнения, можно остановить с помощью сигнала прерывания, который генерируется нажатием клавиши «delete». Если не предпринято специальных мер, то этот сигнал просто вызывает прекращение программы без записи файла *core* с образом памяти.

Существует также сигнал останова *quit*, который позволяет принудительно создать образ памяти. Таким образом, неожиданно зациклившуюся программу можно снять и исследовать образ ее памяти без всякой предварительной подготовки.

Аппаратные ошибки, а также сигналы прерывания и останова процесс может по запросу игнорировать или перехватить. Например, оболочка игнорирует сигналы останова, чтобы предотвратить выход пользователя из системы. Редактор перехватывает сигналы прерывания и возвращается на уровень ввода команд. Это полезно, когда нужно прервать длинную распечатку, не потеряв проделанной работы (редактор работает с копией редактируемого файла). В системах, не оснащенных аппаратной реализацией операций с плавающей точкой, нереализованные инструкции перехватываются и инструкции с плавающей точкой интерпретируются программно.

37.8. Перспективы

Как ни парадоксально это звучит, своим успехом Unix в значительной степени обязана тому, что при ее проектировании не ставились какие-то заранее определенные цели. Первая версия была написана, когда один из нас (Томпсон) оказался неудовлетворен имеющими средствами компьютера, набрел на мало используемую машину PDP-7 и задался целью создать более дружелюбное окружение. Эта, по сути дела личная, инициатива оказалась достаточно успешной, чтобы вызвать интерес у другого автора, и не только, а впоследствии обосновать приобретение PDP-11/20, конкретно для поддержки системы редактирования и форматирования текста. Затем, когда и возможностей 11/20 стало не хватать, Unix уже стала настолько полезной, что удалось убедить руководство потратить деньги на PDP-11/45. На протяжении всей работы наши цели, если вообще озвучивались, неизменно были связаны с построением удобной среды для работы с машиной и с исследованием идей и изобретений в области операционных систем. Перед нами не стояла задача удовлетворить чьи-то требования, и за эту свободу мы очень благодарны. Оглядываясь назад, мы можем назвать три соображения, повлиявших на дизайн Unix.

Во-первых, будучи программистами, мы, естественно, проектировали систему, которая упростила бы написание, тестирование и выполнение программ. Самым важным выражением нашего стремления к удобству программирования было обустройство системы для интерактивной работы, пусть даже первоначальная версия поддерживала только одного пользователя. Мы полагаем, что правильно спроектированная интерактивная система открывает возможности для гораздо более продуктивной и приносящей больше

удовлетворения работы, чем «пакетная». К тому же такую систему сравнительно легко адаптировать к неинтерактивному использованию, тогда как обратное неверно. Во-вторых, всегда существовали довольно серьезные размерные ограничения на систему и ее программное обеспечение. При наличии конфликтущих стремлений к разумной эффективности и высокой выразительной способности размерные ограничения поощряют не только экономию, но и некоторую элегантность дизайна. Возможно, это слегка замаскированная философия «спасения через страдание», но в нашем случае она сработала.

В-третьих, почти с самого начала система могла и действительно сопровождала сама себя. Это более важно, чем могло бы показаться. Если проектировщики системы вынуждены ее использовать, то быстро узнают о ее функциональных и поверхностных недостатках и имеют сильные побудительные мотивы исправить их, пока не стало слишком поздно. Поскольку весь исходный текст программ был в любой момент доступен и его оперативная модификация не вызывала трудностей, мы были готовы пересматривать и переписывать систему и ее программное обеспечение всякий раз, как новые идеи рождались у нас или предлагались другими.

Те аспекты Unix, которые обсуждались в этой статье, наглядно демонстрируют первые два из описанных соображений. Например, интерфейс к файловой системе очень удобен с точки зрения программирования. Самый нижний уровень интерфейса предназначен для стирания различий между устройствами и файлами и между прямым и последовательным доступами. Не нужны никакие большие подпрограммы «методов доступа», изолирующие программиста от системных вызовов; на самом деле все пользовательские программы либо вызывают систему напрямую, либо пользуются небольшой библиотечной программой, содержащей всего несколько десятков инструкций, которая буферизует символы и читает или записывает их за один присест.

Еще один важный аспект удобства программирования – отсутствие «блоков управления» со сложной структурой, которая частично обслуживается и на которую полагается файловая система или другие системные вызовы. Вообще говоря, содержимое адресного пространства программы – собственность самой программы, и мы старались избежать любых ограничений на структуры данных в этом пространстве.

С учетом требования, что любую программу должно быть возможно использовать с любым файлом или устройством в роли ввода или вывода, желательно также, с точки зрения эффективности использования памяти, переместить все зависящие от устройств вопросы внутрь самой операционной системы. Похоже, единственные альтернативы – загружать подпрограммы работы с каждым устройством в каждую программу, что требует много памяти, или опираться на какие-то средства динамического связывания с подпрограммами для каждого устройства в тот момент, когда оно оказывается фактически необходимо, – но это обойдется дорого с точки зрения накладных расходов или оборудования.

Аналогично схема управления процессами и командный интерфейс оказались удобными и эффективными. Поскольку оболочка – обыкновенная выгружаемая пользовательская программа, она не занимает специально зарезервированного места в самой системе и может быть сделана сколь угодно мощной при скромных затратах. В частности, принимая во внимание тот факт, что оболочка работает как процесс, запускающий другие процессы для выполнения команд, реализовать идеи перенаправления ввода-вывода, фоновые процессы, командные файлы и выбираемый самим пользователем интерфейс к системе оказывается тривиально.

<...>

38 Протокол взаимодействия сетей с коммутацией пакетов (1974)

Винтон Серф и Роберт Кан

Компьютерные сети стали ответом на потребности разных ведомств и обрели форму под влиянием коммерческих, государственных и научных инициатив. Запуск советского космического спутника в 1957 году привел к образованию Управления перспективных исследований (ARPA) при Министерстве обороны США. На него была возложена задача пестовать исследования, которые позволили бы избежать таких конфузов в будущем. Еще в 1960 году Пол Бэран, исследователь из корпорации RAND, получивший грант от ВВС США, предположил, что избыточная, распределенная сеть с коммутацией пакетов могла бы пережить ядерный удар, способный вывести из строя всю телекоммуникационную структуру США (Baran 1964). Тем временем Дж. К. Р. Ликлайдер, после публикации работы «Симбиоз человека и компьютера» (глава 20), стал размышлять о «межгалактической компьютерной сети», а встав во главе отдела технологий обработки информации (IPTO) в ARPA в 1964 году, начал щедро финансировать исследования по компьютерным сетям, и эту инициативу подхватил его преемник на посту главы IPTO Роберт Тэйлор. При Тэйлоре начались работы, приведшие в итоге к созданию сети ARPAnet, которой суждено было стать интернетом; миллион долларов из бюджета был потрачен на обеспечение удаленного доступа к компьютерам, построенным на деньги ARPA. Тэйлор поручил Лоуренсу (Ларри) Робертсу проектирование сети, и Робертс, связавшись с Бэраном и британским ученым Дональдом

Дэвисом (работавшим над сходными идеями), остановился на проекте сети с коммутацией пакетов, а в 1968 году выпустил техническое задание на построение аппаратного и программного обеспечения ARPAnet. Робертс сменил Тэйлора на посту главы IPTO и обеспечил непрерывность инициатив по развитию сетей.

Тем временем IBM, DEC и другие компании-производители компьютеров разрабатывали собственные сети для обмена данными и связывания периферийных устройств, но у производителей не было стимула согласовывать стандарт сети, который позволил бы машинам конкурентов присоединиться к их собственной сети. Винтон Серф (родился в 1943 году) и Роберт Кан (родился в 1938 году) превратили ARPAnet в интернет, разработав набор протоколов для соединения сетей.

И Серф, и Кан работали над ранним проектом ARPAnet, Серф – в Калифорнийском университете в Лос-Анджелесе, а Кан – в компании Bolt Beranek and Newman (BBN), которая выиграла первый контракт на ARPAnet. В 1973 году Серф и Кан начали совместную работу над проектированием стандартов и протоколов взаимодействия, итогом которой стала эта основополагающая статья. Описанный в ней базовый проект до сих пор живет, подвергшись лишь одной крупной переработке. То, что в статье названо TCP, в 1978 году было преобразовано в набор из двух протоколов: TCP/IP – протокол узла TCP – и IP – отдельный протокол интернета. TCP обеспечивает надежную сквозную связь между узлами, опираясь на IP – более простой и ненадежный протокол, исполняемый шлюзами.

Серф и Кан сыграли важную роль в продвижении не только технологий интернета, но и самого духа интернета как глобального ресурса для совместного использования информации. В статье, сопровождающей присуждение им премии Тьюринга (ACM 2004), признается как их технический вклад, так и «вдохновляющее лидерство в развитии сетей».

Представлен протокол, обеспечивающий разделение ресурсов, находящихся в различных сетях с коммутацией пакетов. Протокол поддерживает различные размеры отдельных сетевых пакетов, сбои при передаче, восстановление порядка следования пакетов, управление потоком, сквозной контроль ошибок, а также создание и уничтожение логических межпроцессных соединений. Рассмотрены некоторые вопросы реализации и такие проблемы, как межсетевая маршрутизация, учет трафика и обработка тайм-аутов.

38.1. Введение

За последние несколько лет было приложено немало усилий к проектированию и реализации сетей с коммутацией пакетов (Roberts and Wessler 1970a; Pouzin 1973b; Dell 1971; Scantlebury and Wilkinson 1971; Barber 1972; Despres 1972; Kahn and Crowther 1972). Основная причина разработки таких сетей –

желание обеспечить совместное использование вычислительных ресурсов. Сеть с коммутацией пакетов включает механизм передачи для доставки данных от одного компьютера другому или от компьютера к оконечным устройствам. Чтобы придать смысл данным, компьютер и оконечные устройства пользуются общим протоколом (т. е. набором соглашений). Для этой цели уже разработано несколько протоколов (Chambon et al. 1973; Carr et al. 1970; McKenzie 1972; Pouzin 1973a; Walden 1972; McKenzie 1973). Однако эти протоколы решали лишь задачу взаимодействия в пределах одной сети. В этой статье мы представляем проект и философию протокола, который поддерживает разделение ресурсов, принадлежащих разным сетям с коммутацией пакетов.

После краткого введения в проблематику протоколов взаимодействия мы опишем функцию ШЛЮЗА – интерфейса между сетями – и обсудим его роль в протоколе. Затем мы рассмотрим различные детали протокола, как то: адресация, форматирование, буферизация, восстановление порядка следования пакетов, управление потоком, контроль ошибок и т. д. Мы завершим статью описанием механизма межпроцессного взаимодействия и покажем, как его можно поддержать с помощью межсетевого протокола.

Несмотря на то что при проектировании отдельных сетей с коммутацией пакетов необходимо решить много разных и трудных проблем, все эти проблемы многократно усложняются, когда требуется связать между собой непохожие сети. Возникающие проблемы могут не иметь прямых аналогов в отдельных сетях и серьезно влияют на возможные формы межсетевого взаимодействия.

Типичная сеть с коммутацией пакетов состоит из набора вычислительных ресурсов, называемых УЗЛАМИ, одного или нескольких *пакетных коммутаторов* и набора средств связи, соединяющих между собой пакетные коммутаторы. Мы предполагаем, что на каждом УЗЛЕ существуют *процессы*, которые должны взаимодействовать с процессами на своем собственном или других УЗЛАХ. Любое имеющееся определение процесса подойдет для наших целей (Lampson 1968). В общем случае эти процессы и являются конечными источниками и получателями данных в сети. Обычно в отдельной сети существует протокол взаимодействия между любым процессом-источником и процессом-получателем. Только эти процессы и должны знать о таком соглашении, этого достаточно для осуществления взаимодействия. Процессы в двух разных сетях обычно применяют для этой цели различные протоколы. Совокупность пакетных коммутаторов и средств связи называется *подсетью с коммутацией пакетов*. Эти идеи иллюстрируются на рис. 38.1.

В типичной подсети с коммутацией пакетов от узла-источника принимаются данные фиксированного максимального размера вместе с адресом приемника, который используется для маршрутизации данных по принципу сохранения и дальнейшей передачи. Время передачи этих данных обычно зависит от внутренних параметров сети, в частности скорости передачи данных в коммуникационной среде, стратегий буферизации и сигнализирования, маршрутизации, задержек распространения сигнала и т. д. Кроме того, обычно присутствует какой-то механизм обработки ошибок и определения состояния компонентов сети.

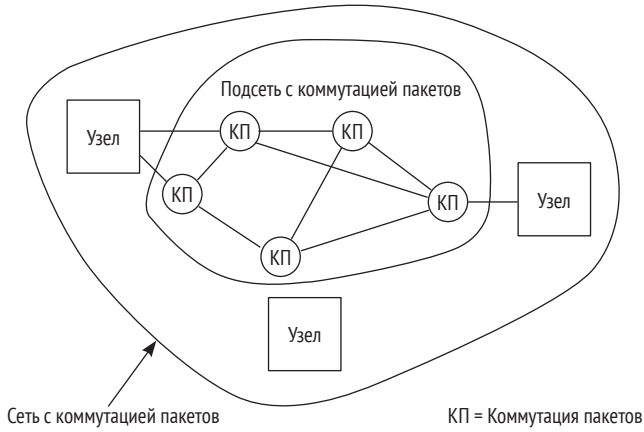


Рис. 38.1. Типичная сеть с коммутацией пакетов

Реализации отдельных сетей с коммутацией пакетов могут различаться в следующих отношениях.

1. У каждой сети могут быть свои способы адресации получателя, поэтому необходимо создать единую схему адресации, которая была бы понятна всем сетям.
2. Для каждой сети может быть определен свой максимальный размер принимаемых данных, поэтому требуется, чтобы сети работали с блоками наименьшего максимального размера (который может оказаться слишком мал для практических целей), или же необходимы процедуры, которые позволят переформатировать данные в порции меньшего размера при пересечении границы сети.
3. Успех или неудача передачи, а также производительность каждой сети определяются различными временными задержками при приеме, доставке и транспортировке данных. Поэтому необходимо тщательно проработать процедуры временно́го согласования сетей, чтобы данные можно было гарантированно доставлять через различные сети.
4. Внутри каждой сети связь может прерываться из-за не подлежащего восстановлению изменения данных или пропажи данных. Желательны сквозные процедуры, которые позволили бы осуществить полное восстановление в таких ситуациях.
5. Информация о состоянии, маршрутизация, механизмы обнаружения сбоев и изоляции, как правило, неодинаковы в разных сетях. Поэтому, чтобы верифицировать некоторые условия, например недоступность или неработоспособность узла-получателя, необходимы различные виды координации взаимодействующих сетей.

Было бы очень удобно, если бы все различия между сетями можно было экономически целесообразным способом разрешить путем введения подходящих интерфейсов на границах сетей. Для многих различий эта цель может быть достигнута. Однако по экономическим и техническим соображениям предпочтительно, чтобы этот интерфейс был максимально прост и надежен

и ориентирован преимущественно на пропуск данных между сетями, в которых используются различные стратегии коммутации пакетов.

Теперь возникает вопрос, должен ли этот интерфейс учитывать различия на уровне узлов или процессов, преобразуя соглашения источника в соответствующие соглашения приемника. Очевидно, что мы хотим разрешить преобразование между стратегиями коммутации пакетов в точке интерфейса, чтобы можно было осуществлять взаимодействие с существующими и планируемыми сетями. Однако сложность и неоднородность протоколов уровня узла или процесса диктует стремление избегать необходимости их преобразования в точке интерфейса, даже если бы такое преобразование всегда было возможно. Вместо этого следует разработать совместимые протоколы уровня узла и процесса, которые позволили бы достичь эффективного межсетевого разделения ресурсов. Неприемлемая альтернатива – реализовывать каждый протокол для каждого узла или процесса (число которых потенциально не ограничено). Поэтому мы будем предполагать, что между узлами или процессами в разных сетях используется общий протокол и что интерфейс между сетями должен играть в этом протоколе как можно более скромную роль.

Чтобы сети, принадлежащие разным владельцам, можно было объединить между собой, несомненно, необходимо как-то учитывать трафик, проходящий через интерфейс. В самых простых словах это означает учет обрабатываемых каждой сетью пакетов, плата за которые передается от одной сети к другой, пока, наконец, не будет начислена пользователю или его представителю. Кроме того, объединение не должно нарушать целостность внутренних операций каждой отдельной сети. Этого легко добиться, если две сети объединены так, будто каждая является узлом другой, но без использования каких-либо изощренных преобразований протокола узла. Таким образом, представляется, что интерфейс между сетями должен играть центральную роль в разработке любой стратегии объединения сетей. Мы дадим интерфейсу, выполняющему эти функции, специальное название – шлюз.

38.2. Понятие шлюза

На рис. 38.2 изображены три отдельные сети, *A*, *B* и *C*, соединенные шлюзами *M* и *N*. Шлюз *M* служит интерфейсом между сетями *A* и *B*, а шлюз *N* – интерфейсом между сетями *B* и *C*. Мы предполагаем, что отдельная сеть может иметь более одного шлюза (например, сеть *B*) и что между любой парой сетей может быть несколько путей со шлюзами. Ответственность за правильную маршрутизацию данных возлагается на шлюз.

На практике шлюз между двумя сетями может состоять из двух половин, каждая из которых ассоциирована со своей сетью. Каждую половину шлюза можно реализовать так, что она будет лишь включать межсетевые пакеты в локальный пакет или извлекать их из локального пакета. Мы предлагаем сделать так, чтобы шлюз обрабатывал межсетевые пакеты в стандартном формате, но не предлагаем никакой конкретной процедуры передачи между двумя половинами шлюза.

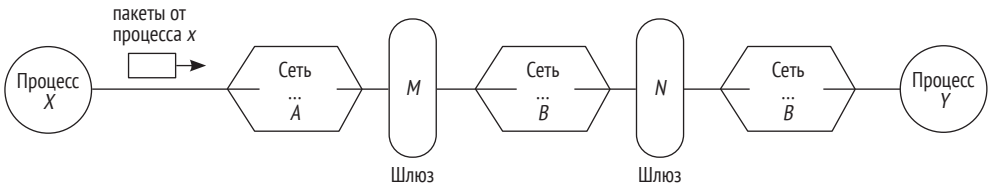


Рис. 38.2. Три сети, объединенные двумя шлюзами

Давайте теперь проследим поток данных через объединенные сети. Мы предполагаем, что пакет данных от процесса *X* поступает в сеть *A* и должен быть доставлен процессу *Y* в сети *C*. Адрес *Y* первоначально задается процессом *X*, а адрес шлюза *M* выводится из адреса процесса *Y*. Мы не пытаемся диктовать, должен ли выбор шлюза производиться процессом *X*, его узлом или одним из пакетных коммутаторов в сети *A*. Пакет путешествует по сети *A*, пока не достигнет шлюза *M*. В шлюзе пакет переформатируется в соответствии с требованиями сети *B*, эта единица учитывается в трафике между *A* и *B*, и шлюз доставляет пакет в сеть *B*. На основе конечного адреса *Y* определяется адрес следующего шлюза. В данном случае следующим является шлюз *N*. Пакет путешествует по сети *B*, пока, наконец, не достигнет шлюза *N*, где он переформатируется в соответствии с требованиями сети *C*. Эта единица еще раз учитывается в трафике между сетями *B* и *C*. Попав в сеть *C*, пакет маршрутизируется узлу, на котором расположен процесс *Y*, и там доставляется конечному получателю.

Поскольку шлюз должен понимать адреса узлов источника и приемника, эта информация должна быть представлена в стандартном формате в каждом пакете, приходящем в шлюз. Она содержится в межсетевом заголовке, добавляемом в начало пакета узлом-источником. Формат пакета, включающего межсетевой заголовок, показан на рис. 38.3. Поля источник и приемник единообразно и однозначно идентифицируют адрес каждого узла в объединенной сети. Адресация – весьма сложный вопрос, он более подробно обсуждается в следующем разделе. Следующие два поля заголовка содержат порядковый номер и счетчик байтов, которые можно использовать для правильного упорядочения пакетов по прибытии в место назначения; они также позволяют шлюзам обнаруживать ошибки, влияющие на пакет. Поле флагов несет специальную управляющую информацию и обсуждается в разделе о повторной передаче и обнаружении дубликатов ниже. Оставшаяся часть пакета содержит подлежащий доставке текст и заключительную контрольную сумму, используемую программным обеспечением для сквозной проверки. Шлюз не изменяет текст и просто передает контрольную сумму, не вычисляя и не пересчитывая ее.

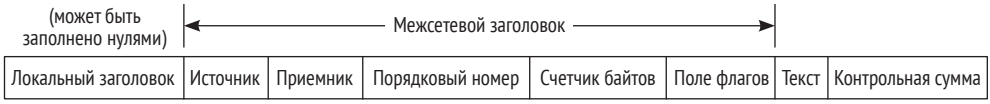


Рис. 38.3. Формат межсетевого пакета (длина каждой клетки не отражает размер поля)

Каждая отдельная сеть может при необходимости расширить формат пакета, прежде чем передавать его через себя. Мы указали на рисунке локальный заголовок, добавляемый в начало пакета. *Локальный заголовок* введен просто для иллюстрации идеи встраивания межсетевого пакета в формат отдельной сети, через которую должен пройти пакет. Понятно, что его точная форма будет изменяться от сети к сети, и в некоторых случаях он даже может оказаться ненужным. Хотя на рисунке это явно не показано, возможно также добавление локального трейлера в конец пакета. Если не введено формального ограничения на длину передаваемого пакета, с тем чтобы пакет могла принять любая отдельная сеть, то не исключено, что шлюз должен будет разбить пакет на два или более меньших пакетов. Это действие называется фрагментацией и должно выполняться таким образом, чтобы приемник мог снова собрать фрагментированный пакет. Ясно, что формат межсетевого заголовка диктует минимальный размер пакета, который должны принимать все сети (нет сомнений, что все сети захотят транспортировать пакеты большего размера). Мы полагаем, что в перспективе рост и развитие межсетевого взаимодействия будут серьезно тормозиться тем, насколько больше минимального может быть размер пакета, и тому есть следующие причины.

1. Если задан максимально допустимый размер пакета, то становится невозможно полностью изолировать параметры размера внутреннего пакета одной сети от параметров размера внутреннего пакета всех остальных сетей.
2. Было бы очень трудно увеличить максимально допустимый размер пакета в ответ на появление новых технологий (например, систем с большим объемом памяти, средств связи с большей скоростью передачи данных и т. д.).
3. Ассоциативная адресация и шифрование пакетов могут потребовать увеличения размера конкретного пакета во время транспортировки для включения в него новой информации.

Возможность фрагментации (независимо от того, где она производится) позволяет варьировать размер пакета на уровне отдельной сети, не нуждаясь в глобальном администрировании, а также дает возможность изолировать узлы и процессы от изменений размеров пакетов, разрешенных в сетях, через которые должны проходить их данные.

Если фрагментация необходима, то, по-видимому, лучше делать это при входе в следующую сеть на шлюзе, т. к. только этот шлюз (а не прочие сети) должен знать о параметрах размера внутреннего пакета, из-за которых возникла необходимость во фрагментации.

Если шлюз фрагментирует входящий пакет на два или более, то в конечном итоге они должны быть доставлены узлу-приемнику в виде фрагментов или заново собраны для этого узла. Можно предположить, что кто-то захочет, чтобы сборку выполнял шлюз, дабы упростить задачу принимающего узла (или процесса) и (или) воспользоваться преимуществами большего размера пакета. Наша позиция заключается в том, что шлюзы не должны выполнять эту функцию, потому что сборка пакетов шлюзом может привести к серьезным проблемам с буферизацией, потенциальным взаимоблокировкам, не-

обходимости передавать все фрагменты одного пакета через один и тот же шлюз, а также к увеличению времени задержки передачи. Кроме того, присутствия этой функции только в шлюзе все равно недостаточно, потому что последний шлюз может также фрагментировать пакет для передачи. И таким образом, узел-приемник должен быть готов выполнить эту задачу.

Теперь немного поговорим о несколько необычной ситуации в учете, которая возникает, когда пакет может быть фрагментирован одним или несколькими шлюзами. Предположим для простоты, что каждая сеть первоначально взимает фиксированную плату за каждый переданный пакет, не зависящую от расстояния, и если одна сеть может обрабатывать пакеты большего размера, чем другая, то и плата пропорционально увеличивается. Предположим также, что последующее увеличение размера пакета в любой сети не приводит к увеличению платы за его передачу для конечных пользователей. Таким образом, сумма, предъявляемая к оплате пользователю, остается постоянной при транспортировке любой сетью, которая вынуждена фрагментировать пакет. Необычный эффект возникает, когда пакет фрагментируется на меньшие пакеты, которые должны по отдельности пройти через следующую сеть, в которой размер пакета больше, чем размер исходного нефрагментированного пакета. Мы ожидаем, что большинство сетей будут естественно выбирать близкие размеры пакетов, но в любом случае увеличение размера пакета в одной сети, даже если это приводит к фрагментации, не должно увеличивать стоимость передачи и фактически может даже снизить ее. В случае принятия любой другой политики тарификации (отличной от предлагаемой нами) разницу в стоимости можно использовать в качестве экономического рычага, направленного на оптимизацию производительности отдельной сети.

38.3. Взаимодействие на уровне процессов

Мы предполагаем, что процессы хотели бы взаимодействовать в полнодуплексном режиме, когда обе стороны могут использовать сообщения неограниченной, но конечной длины. Текст сообщения от процесса терминалу и обратно может состоять из одного-единственного символа. Текст сообщения от файла процессу может содержать целую страницу символов. Поток данных (например, непрерывно генерируемая строка битов) может быть представлен последовательностью сообщений конечной длины.

Мы предполагаем, что внутри узла существует программа управления передачей (transmission control program – TCP), которая обрабатывает передачу и прием сообщений от имени обслуживаемых ей процессов. TCP по очереди обслуживается одним или несколькими пакетными коммутаторами, подключенным к узлу, на котором размещена эта TCP. Процессы, желающие взаимодействовать, предъявляют TCP сообщения для передачи, а TCP доставляет входящие сообщения соответствующим процессам-получателям. Мы разрешаем TCP разбивать сообщения на сегменты, поскольку приемник может ограничивать количество поступающих данных, потому что локаль-

ная сеть может ограничивать максимальный размер передачи или потому что TCP должна конкурентно разделять свои ресурсы между несколькими процессами. Кроме того, мы ограничиваем длину сегмента целым числом 8-битовых байтов. Это единообразие особенно полезно, чтобы упростить программное обеспечение, необходимое на машинах-узлах с разной естественной длиной слова. На уровне процесса могут быть предприняты меры для дополнения сообщения, содержащего нецелое число байтов, и для решения о том, какие из поступающих байтов текста содержат информацию, интересную процессу-получателю.

Мультиплексирование и демультиплексирование сегментов между процессами – важнейшие задачи TCP. При передаче TCP должна мультиплексировать сегменты от разных процессов-источников и порождать межсетевые пакеты для доставки одному из обслуживающих пакетных коммутаторов. При приеме TCP должна принимать последовательность пакетов от обслуживающего ее коммутатора (или нескольких коммутаторов). По этой последовательности поступающих пакетов (обычно от разных узлов) TCP должна уметь реконструировать сообщения и доставлять их нужным процессам-получателям.

Мы предполагаем, что в каждый сегмент включена дополнительная информация, позволяющая передающей и принимающей TCP идентифицировать процессы-приемники и процессы-источники соответственно. В этот момент мы должны ответить на важный вопрос. Как TCP источника форматируют сегменты, предназначенные для одной и той же TCP приемника? Мы рассмотрим два случая.

Случай 1. Заняв позицию, согласно которой границы сегмента несущественны и поток байтов может быть образован сегментами, предназначенными для одной и той же TCP, мы сможем улучшить эффективность передачи и степень разделения ресурсов, произвольным образом разбив поток на пакеты и допустив, что много сегментов может находиться в заголовке одного межсетевого пакета. Однако такая позиция ведет к необходимости точно и по порядку реконструировать поток байтов текста, порождаемый TCP источника. На стороне приемника этот поток нужно сначала разложить на сегменты, а те, в свою очередь, использовать для реконструкции сообщений и их доставки нужным процессам. Это фундаментальные проблемы данной стратегии, и связаны они с тем, что пакеты могут приходить в точку назначения не по порядку. Самая критичная проблема – возможное взаимовлияние процессов, разделяющих один и тот же поток байтов TCP–TCP. Особенно это относится к принимающей стороне. Во-первых, TCP может оказаться в затруднительном положении из-за того, что нужно разобрать поток на сегменты, а затем распределить их по буферам, где заново собираются сообщения. Если сразу не очевидно, что пришел весь сегмент (напомним, он может находиться в нескольких пакетах), то TCP, возможно, придется временно приостановить разбор, пока не придет больше пакетов. Во-вторых, если пакет отсутствует, то не ясно, можно ли передавать принимающему процессу последующие пакеты, даже когда они допускают идентификацию, если только TCP не знает о какой-то схеме упорядочения на уровне процесса. Такое знание позволило бы TCP решить, можно ли доставлять очередной

сегмент ожидающему процессу. Нахождение начала сегмента в случае, когда в потоке байтов имеются лакуны, также может представлять трудности.

Случай 2. С другой стороны, мы могли бы занять позицию, согласно которой TCP должна иметь возможность определить, сразу после поступления и без дополнительной информации, какому процессу или процессам адресован принятый пакет и следует ли его доставлять.

Если в обязанности TCP входит определение процесса, которому адресован поступивший пакет, то каждый пакет должен содержать заголовок процесса (отличный от межсетевого заголовка), который полностью идентифицирует процесс-получатель. Для простоты предположим, что каждый пакет содержит текст от одного процесса, предназначенный одному процессу. Тогда в каждом пакете должен быть только один заголовок процесса. Чтобы решить, можно ли доставлять поступившие данные процессу-получателю, TCP должна уметь определять, пришли ли данные в правильном порядке (мы можем наделить процесс-получатель возможностью проинструктировать TCP игнорировать порядок, но это считается особым случаем). В предположении, что каждый поступающий пакет содержит заголовок процесса, TCP приемника сразу же получает в свое распоряжение необходимые средства упорядочения и идентификации процесса-получателя.

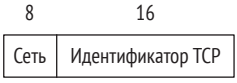


Рис. 38.4. TCP-адрес

В обоих случаях 1 и 2 обеспечиваются демультиплексирование и доставка сегментов процессу-получателю, но только в случае 2 это делается без потенциального нежелательного межпроцессного взаимодействия. Кроме того, в случае 1 необходим дополнительный механизм управления потоком на уровне узел–узел, потому что должны быть также приняты меры для управления на уровне процессов, и этот механизм используется нечасто, поскольку мала вероятность того, что внутри данного узла по стечению обстоятельств будет запланирована отправка сообщений одному и тому же узлу-приемнику со стороны двух процессов. По этой причине мы выбираем случай 2 для включения в протокол межсетевой передачи.

38.4. Форматы адреса

Выбор форматов адреса в случае межсетевого обмена – проблема, потому что адреса в локальных сетях TCP могут существенно различаться по формату и размеру. Единое адресное пространство TCP, которое понимали бы шлюз и TCP, абсолютно необходимо для маршрутизации и доставки межсетевых пакетов. Аналогичные трудности возникают с адресацией процессов и, более общо, с адресацией портов. Мы вводим понятие порта, чтобы дать процессу возможность различать несколько потоков сообщений. Порт – это

просто обозначение одного такого потока сообщений, ассоциированного с процессом. Средства идентификации портов в общем случае различны в разных операционных системах, и потому, чтобы получить единообразную адресацию, необходим также стандартный формат адреса порта. Адрес порта ассоциируется с полнодуплексным потоком сообщений.

38.5. TCP-адресация

TCP-адресация тесно связана с вопросами маршрутизации, потому что узел или шлюз должны выбрать подходящий узел или шлюз для исходящего межсетевого пакета. Постулируем следующий формат TCP-адреса (рис. 38.4). Выбор идентификатора сети (8 бит) позволяет адресовать до 256 различных сетей. На обозримое будущее это представляется достаточным. Аналогично поле идентификатора TCP позволяет адресовать до 65 536 различных TCP; этого, по всей видимости, более чем достаточно для любой сети.

Когда пакет проходит через шлюз, тот смотрит на идентификатор конечной сети, чтобы определить, как маршрутизировать пакет. Если конечная сеть подключена к шлюзу, то младшие 16 бит TCP-адреса используются для формирования локального TCP-адреса в конечной сети. Если же конечная сеть не подключена к шлюзу, то старшие 8 бит используются для выбора следующего шлюза. Мы не пытаемся специфицировать, как каждая отдельная сеть должна ассоциировать межсетевой идентификатор TCP с его локальным TCP-адресом. Мы также не исключаем возможность, что локальная сеть понимает схему межсетевой адресации и снимает со шлюза ответственность за маршрутизацию.

38.6. Адресация портов

Перед принимающей TCP стоит задача демультиплексирования получаемых межсетевых пакетов и реконструкции исходных сообщений для каждого процесса-получателя. У каждой операционной системы имеются собственные внутренние средства идентификации процессов и портов. Мы предполагаем, что 16 бит достаточно для идентификации межсетевых портов. Процессу-отправителю не нужно знать, как будет использоваться идентификатор порта получателя. TCP приемника сможет разобрать это число и определить, в какой буфер помещать поступающие пакеты. Мы предусматриваем широкое поле для номера порта, чтобы поддержать процессы, которые хотят различать много одновременных потоков сообщений. В действительности нам не важно, как эти 16 бит будут интерпретироваться участвующей в схеме TCP.

Несмотря на то что поле имени порта в адресе широкое, это все же компактное внешнее имя для внутреннего представления порта. Использование коротких имен для идентификаторов портов часто желательно, чтобы сни-

зить накладные расходы на передачу и, возможно, уменьшить время обработки пакета TCP приемника. Однако назначение коротких имен каждому порту требует начального согласования между источником и приемником, которые должны прийти к общему решению о выборе короткого имени, о последующем сопровождении таблиц преобразования на стороне источника и приемника и о финальной операции освобождения короткого имени. При динамическом назначении имен портов такое согласование, вообще говоря, необходимо в любом случае.

38.7. Форматы сегментов и пакетов

Как показано на рис. 38.5, TCP разбивает сообщения на сегменты, формат которых более подробно представлен на рис. 38.6. Длины полей ориентировочные. Первые два поля (на рисунке – порт источника и порт приемника) уже обсуждались в предыдущем разделе, посвященном адресации. Использование третьего и четвертого полей (на рисунке – окно и подтверждение ACK) будет рассмотрено ниже в разделе о повторной передаче и обнаружении дубликатов. Напомним (см. рис. 38.3), что межсетевой заголовок содержит порядковый номер и счетчик байтов, а также поле флагов и контрольную сумму. Порядок использования этих полей объясняется в следующем разделе.

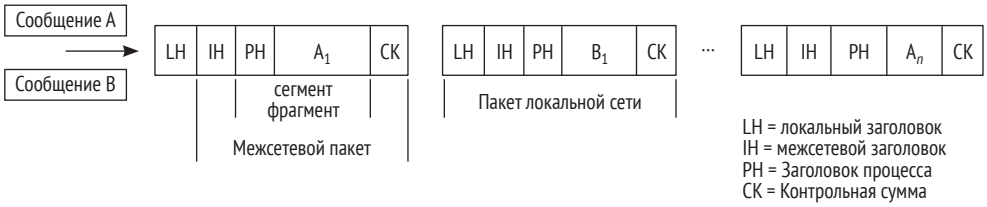


Рис. 38.5. Создание сегментов и пакетов из сообщений

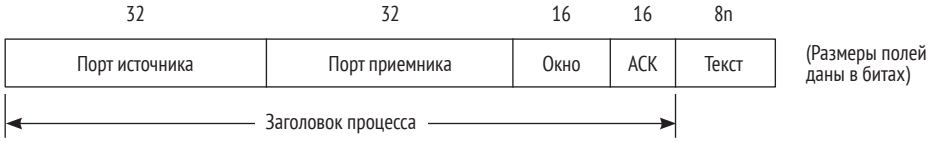


Рис. 38.6. Формат сегмента (заголовок процесса и текст)

38.8. Сборка и восстановление порядка следования

Для реконструкции сообщения на стороне принимающей TCP, очевидно, необходимо, чтобы в каждом межсетевом пакете присутствовал порядковый номер, уникальный в пределах потока сообщений, адресованных конкретному порту приемника. Порядковые номера должны монотонно возрастать

(или убывать), поскольку используются для переупорядочения поступающих пакетов и сборки из них сообщения. Если бы пространство порядковых номеров было бесконечно, то мы могли бы просто присваивать каждому новому пакету следующий номер. Понятно, однако, что это пространство бесконечным быть не может, а проблемы, возникающие из-за его конечности, мы рассмотрим при обсуждении повторной передачи и обнаружения дубликатов в следующем разделе. Мы предлагаем следующую схему упорядочения пакетов, а стало быть, и реконструкции сообщений TCP приемника.

Пара портов обменивается одним или несколькими сообщениями на протяжении некоторого периода времени. Последовательность сообщений, порожденных одним портом, можно было бы рассматривать так, будто она вложена в бесконечно длинный поток байтов. У каждого байта сообщения имеется уникальный порядковый номер, который мы будем считать положением байта относительно начала потока. Когда TCP источника извлекает сегмент из сообщения и форматирует его для межсетевой передачи, относительное положение первого байта текста сегмента используется как порядковый номер пакета. Поле счетчика байтов в межсетевом заголовке учитывает весь текст в сегменте (но не включает байты контрольной суммы и байты, входящие в состав межсетевого заголовка и заголовка процесса). Подчеркнем, что порядковый номер, ассоциированный с данным пакетом, уникален только в пределах пары взаимодействующих портов (см. рис. 38.7). При поступлении пакета проверяется, какому порту он адресован. Затем порядковый номер используется, чтобы определить относительное положение текста пакета в реконструируемом сообщении. Отметим, что это позволяет узнать точную позицию данных в реконструируемом сообщении, даже если некоторые его части еще отсутствуют.

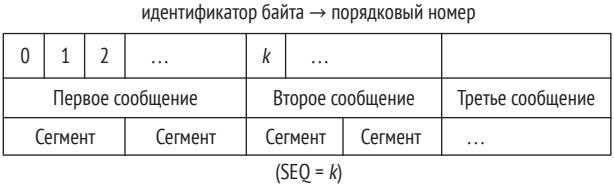


Рис. 38.7. Присваивание порядковых номеров

Каждый сегмент, порожденный TCP источника, упаковывается в один межсетевой пакет, а контрольная сумма вычисляется по тексту и заголовку процесса, ассоциированному с сегментом. Разбиение сообщений на сегменты, производимое TCP, и потенциальное разбиение сегментов на меньшие части, производимое шлюзами, приводит к необходимости указывать TCP приемника, когда поступил конец сегмента (end of a segment – ES), а когда – конец сообщения (end of message – EM). Для этой цели служит поле флагов в межсетевом заголовке (см. рис. 38.8).

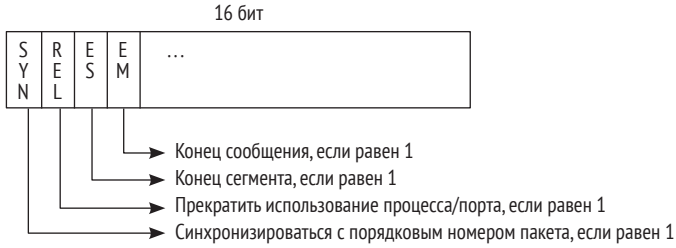


Рис. 38.8. Поле флагов в межсетевом заголовке

Поле ES устанавливается TCP источника всякий раз при подготовке сегмента к передаче. Если оказалось, что сообщение целиком содержится в сегменте, то устанавливается также флаг EM. Флаг EM устанавливается и в последнем сегменте сообщения, если оно не помещается в одном сегменте. Эти два флага используются TCP приемника, чтобы определить наличие контрольной суммы для данного сегмента и понять, что сообщение прибыло полностью.

Флаг ES и EM в межсетевом заголовке известны шлюзу и особенно важны, когда пакеты необходимо разбивать для прохождения через следующую сеть. Их использование иллюстрируется на рис. 38.9.

Исходное сообщение A на рис. 38.9 разбито на два сегмента A_1 и A_2 и отформатировано TCP на два межсетевых пакета. В пакетах A_1 и A_2 установлены биты ES, а в пакете A_2 – еще и бит EM. Когда пакет A_1 проходит через шлюз, он разбивается на две части: пакет A_{11}, в котором не установлен ни один из битов EM и ES, и пакет A_{12}, в котором установлен бит ES. Аналогично пакет A_2 разбивается так, что в первой части, пакете A_{21}, не установлен ни один бит, а во второй, A_{22}, установлены оба. Поле порядкового номера (SEQ) и поле счетчика байтов (CT) в каждом пакете модифицируются шлюзом, так чтобы были правильно отражены байты текста в нем. Для выполнения фрагментации шлюзу нужно исследовать только межсетевой заголовок.

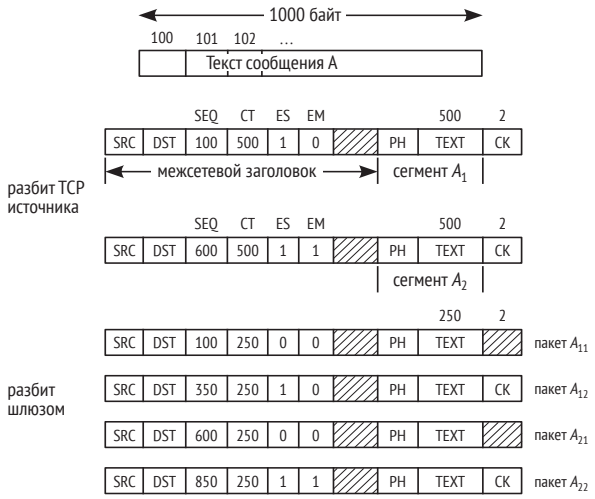


Рис. 38.9. Разбиение сообщения и разбиение пакета

TCP приемника, осуществляя сборку сегмента A_1, обнаружит флаг ES и проверит контрольную сумму, которая, как она знает, находится в пакете A_{12}. По получении пакета A_{22}, в предположении, что все остальные пакеты прибыли, TCP приемника обнаружит, что собрала все сообщение, и может уведомить процесс-получатель о его приеме.

Не бывает стопроцентно надежной передачи. Мы предлагаем механизм тайм-аута и положительного квитирования, который позволит TCP восстановиться после потери пакетов от того или иного узла. TCP передает пакеты и ждет ответов (подтверждений), которые передаются в составе обратного потока пакетов. Если на какой-то пакет не пришло подтверждения, то TCP передает его повторно. Мы ожидаем, что механизм повторной передачи на уровне узла, описанный в последующих параграфах, на практике будет вызываться не слишком часто. Уже имеются свидетельства (Pouzin 1973b) в пользу того, что отдельные сети можно эффективно построить без использования этой возможности. Однако включение повторной передачи на уровне узла позволяет восстановиться после случайных сетевых проблем и допускает использование широкого спектра стратегий реализации протоколов узла. Мы предвидим, что этот механизм иногда будет активироваться, когда узлу необходимо адаптироваться к редким всплескам спроса на ограниченные буферные ресурсы, но и только.

Любая политика повторной передачи требует каких-то средств обнаружения дубликатов приемником. Даже если бы было доступно бесконечное число различных порядковых номеров пакетов, все равно возникла бы проблема – как долго приемник должен хранить полученные пакеты, чтобы избежать дубликатов. Ситуация осложняется еще и потому, что на самом деле число различных порядковых номеров конечно, а если они используются повторно, то приемник должен как-то отличать новую передачу от повторной.

Здесь предлагается стратегия *окна*, аналогичная той, что была применена во французской системе cyclades (режим передачи *voie virtuelle*[1] [Chambon et al. 1973]) и при подключении сильно удаленного узла в сети ARPANET (BBN 1973). См. рис. 38.10.

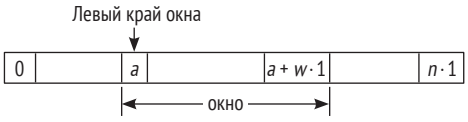


Рис. 38.10. Концепция окна

Предположим, что в поле порядкового номера в межсетевом заголовке могут находиться числа от 0 до $n - 1$. Предположим также, что отправитель не посылает более w байт, не получив подтверждения. Эти w образуют окно (см. рис. 38.11). Очевидно, что w должно быть меньше n. Правила для отправителя и получателя следующие.

Отправитель: пусть L – порядковый номер, ассоциированный с левым краем окна.

[1] Виртуальный путь. – *Прим. перев.*

1. Отправитель передает байты из сегментов, текст которых расположен между позициями L и $L + w - 1$ включительно.
2. По истечении тайм-аута (его продолжительность не специфицируется) отправитель повторно передает неподтвержденные байты.
3. После получения подтверждения, содержащего текущий левый край окна получателя, отправитель сдвигает левый край своего окна вправо на подтвержденное число байтов (правый край при этом тоже неявно сдвигается).

Получатель:

1. Поступающие пакеты, порядковые номера которых совпадают с левым краем текущего окна отправителя, подтверждаются путем отправки источнику следующего ожидаемого порядкового номера. Тем самым подтверждаются и все промежуточные байты. Левый край окна сдвигается вправо до следующего ожидаемого порядкового номера.
2. Поступающие пакеты, порядковые номера которых находятся левее края окна (фактически вне окна), отбрасываются, и в подтверждении возвращается текущий левый край окна.
3. Пакеты, порядковые номера которых находятся внутри окна получателя, но не совпадают с его левым краем, могут быть сохранены или отброшены, но в любом случае подтверждаются. Это случай, когда пакеты приходят не по порядку. <…>

1	Адрес источника	
2	Адрес приемника	
3	Порядковый номер следующего пакета	
4	Текущий размер буфера	
5	Следующая позиция записи	
6	Следующая позиция чтения	
7	Позиция конца чтения	
8	Количество повторных передач	Макс. количество повторных передач
9	Тайм-аут	Флаги
10	Текущее подтверждение	Окно

Рис. 38.11. Концептуальный формат блока TCB
[Примечание редактора: TCB = блок управления передачей]

38.9. Управление потоком

Каждый сегмент, поступающий TCP приемника, рано или поздно подтверждается путем возврата порядкового номера следующего сегмента, который должен быть передан процессу (возможно, что он еще не пришел).

Выше мы описали, как использование пространства порядковых номеров и окна позволяет обнаруживать дубликаты. Подтверждения передаются в заголовке процесса (см. рис. 38.6), и вместе с ними есть возможность пере-

дать «рекомендуемое окно», которое получатель может использовать для управления потоком данных от отправителя. Предполагается, что это станет основным компонентом механизма управления потоком. Получатель вправе изменять размер окна по собственному алгоритму, при условии что размер окна никогда не превышает половины пространства порядковых номеров.

Этот механизм управления потоком чрезвычайно мощный и гибкий и не подвержен проблемам синхронизации, которые могут возникнуть в схемах инкрементного выделения буферов (Carr et al. 1970; McKenzie 1972). Однако он опирается на эффективную стратегию повторной передачи. Получатель может уменьшить окно даже тогда, когда пакеты находятся на пути от отправителя, окно которого в настоящий момент больше. При таком уменьшении получатель может отбросить входящие пакеты (поскольку они оказались вне окна) и повторить передачу текущего размера и текущего левого края окна в подтверждении. Равным образом и отправитель может, исходя из обстоятельств, послать больше данных, чем позволяет окно, надеясь, что получатель расширит окно, чтобы принять их (конечно, отправитель никогда не должен посылать больше половины размера пространства порядковых номеров). Обычно мы ожидаем, что отправитель будет соблюдать ограничения на размер окна. Расширение окна получателем просто позволяет принять больше данных. <...>

39 Программирование с абстрактными типами данных (1974)

Барбара Лисков и Стивен Зиллес

Начиная с середины 1960-х и на протяжении всех 1970-х годов, по мере того как устремления ученых, занимающихся информатикой, становились все более экстравагантными, программные проекты – все более крупными, ошибки – все более тонкими, случалось – и не раз, – что программную систему, стоившую миллионы долларов, так и не удавалось заставить работать и приходилось отправлять на свалку. Был провозглашен «кризис программного обеспечения», и как грибы начали расти инициативы, направленные на управление сложностью путем ограничения свободы выражения программистов. Одни средства включались в языки, другие исключались – все для того, чтобы побудить (или принудить) программистов скрывать сложность машинного кода, стоявшего за реализацией высокоуровневых программных абстракций. Сформировалось два пути канализации этих усилий: контроль над потоком программ и манипулирование данными. В первом случае программное обеспечение надлежало разбивать на модули, а управляющие конструкции ограничивались несколькими примитивами для организации циклов и подпрограмм (см. главу 29 об атаке Дейкстры на оператор «go to», а также статью [Dijkstra 1972] с полным изложением философии структурного программирования). Второй подход, ярким представителем которого является эта статья, предлагал включить в язык соглашения, которые позволили бы выражать только минимально необходимые функциональные операции над данными, но не внутреннюю структуру данных. Этот вид языков программирования «сверхвысокого уровня» теперь стал практически стандар-

том; своими корнями объектно-ориентированное программирование уходит именно сюда.

Барбара Лисков (родилась в 1939 году) защитила докторскую диссертацию в Стэнфорде под руководством Джона Маккарти в 1968 году; она стала одной из первых женщин, имеющих степень PhD в этой области. С тех пор большая часть ее карьеры прошла в МТИ, где она является Институтским профессором, высшее звание в преподавательской иерархии МТИ. Премию Тьюринга она получила в 2008 году, частично за труды в области абстрагирования данных, хотя также много работала над проблемами распределенных и отказоустойчивых вычислений. Стивен Зиллес, бывший аспирантом в МТИ во время написания этой статьи, ныне находится на пенсии после карьеры в IBM и Adobe.

39.0. Краткое описание

За этой работой по языкам сверхвысокого уровня стоит стремление облегчить задачу программирования, предоставив программистам язык, содержащий примитивы или абстракции, подходящие для интересующей его предметной области. Тогда программист сможет прикладывать усилия в нужной точке; он будет сосредоточен на решении своей задачи, а получившаяся в итоге программа окажется более надежной. Очевидно, что это достойная цель.

К сожалению, проектировщику очень трудно заранее выбрать все абстракции, которые могут понадобиться пользователям его языка. Если язык вообще будет использоваться, то вполне вероятно, что для решения задач, которые проектировщик не предусмотрел и для которых встроенных в язык абстракций недостаточно.

В этой работе рассматривается подход, который позволяет дополнять набор встроенных в язык абстракций, когда возникает необходимость в новой абстракции данных. Этот подход к работе с абстракциями – естественное развитие работы по проектированию языка для структурного программирования. Описываются относящиеся к делу аспекты языка и приводятся примеры использования и определения абстракций.

39.1. Введение

В этой работе описывается подход к представлению абстракций в компьютере. Подход, разработанный при проектировании языка, поддерживающего структурное программирование, может также найти применение в работе над языками сверхвысокого уровня. Мы начнем с объяснения его релевантности и сравнения работ по структурному программированию и языкам сверхвысокого уровня.

Цель структурного программирования – повысить надежность и понятность программ. Можно ожидать, что языки сверхвысокого уровня, которые изначально предназначались для повышения продуктивности программиста путем упрощения стоящей перед ним задачи, также улучшат надежность и понятность кода. Таким образом, от работы в обоих направлениях можно ожидать похожих преимуществ.

Однако работа в этих двух областях развивается по трем разным линиям. Язык сверхвысокого уровня старается предоставить пользователю абстракции (операции, структуры данных и управляющие конструкции), полезные в его предметной области. Пользователь может работать с этими абстракциями, не беспокоясь о том, как они реализованы, – его должно интересовать только, что они делают. Поэтому он может игнорировать детали, не относящиеся к его предметной области, и сконцентрироваться на решении своей проблемы.

Структурное программирование стремится ввести дисциплину в задачу программирования, с тем чтобы получающиеся программы были «хорошо структурированы». При такой дисциплине для решения проблемы применяется процесс последовательной декомпозиции. Первый шаг – написать программу, которая решает проблему, но работает на некоей абстрактной машине, предоставляющей только те объекты данных и операции, которые идеально подходят для решения. Некоторые или даже все такие объекты данных и операции действительно абстрактны, т. е. не представлены примитивами используемого языка программирования. Пока что объединим их, неформально назвав «абстракцией».

Программист изначально озабочен тем, чтобы убедить себя (или доказать), что написанная им программа правильно решает проблему. В процессе анализа он смотрит, как программа использует абстракции, но не вникает в детали реализации этих абстракций. Каждая абстракция представляет новую проблему, для решения которой требуются дополнительные программы. Новую программу также можно написать для абстрактной машины, что приведет к появлению новых абстракций. Первоначальная проблема будет окончательно решена, когда все абстракции, порожденные в процессе конструирования программы, будут реализованы последующими программами.

Теперь понятно, что подходы к языкам сверхвысокого уровня и к структурному программированию связаны между собой: каждый основан на идее использования таких абстракций, которые подходят для решения поставленной проблемы. Более того, обоснование использования абстракций в обоих подходах одинаково: освободить программиста от заботы о деталях, не имеющих отношения к решаемой проблеме.

В языках сверхвысокого уровня проектировщик пытается выявить набор полезных абстракций заранее. С другой стороны, язык структурного программирования не содержит составленных заранее представлений о конкретном наборе полезных абстракций, но зато должен предоставить механизм, посредством которого язык можно будет расширить, включив в него нужные пользователю абстракции. Язык, содержащий такой механизм, можно считать языком общего назначения неопределенно высокого уровня.

В этой статье мы опишем подход, который позволяет расширять набор встроенных абстракций, когда возникает необходимость в новой абстракции. Мы начнем с анализа абстракций, используемых при написании программ, и выявим потребность в абстрагировании данных. Язык, поддерживающий использование и определение абстракций данных, описывается неформально, и приводятся примеры программ на нем. В последующих разделах статьи обсуждается связь этого подхода с предшествующими работами, а также некоторые аспекты реализации языка.

39.2. Семантика абстракции

Описание структурного программирования в предыдущем разделе расплывчато, потому что сформулировано в таких не определенных нами терминах, как «абстракция» и «абстрактная машина». В этом разделе мы проанализируем смысл термина «абстракция», чтобы понять, какого рода абстракции нужны программисту и как язык структурного программирования может эти требования поддержать.

От абстрагирования нам нужен механизм, который допускает выражение релевантных деталей и сокрытие нерелевантных. В случае программирования способ возможного использования абстракции релевантен, а способ ее реализации нерелевантен. Взглянув на традиционные языки программирования, мы обнаружим, что они предлагают эффективное подспорье для абстрагирования: функцию или процедуру. Используя процедуру, программист интересуется (или должен интересоваться) только тем, что она делает – чем ему может быть полезна предоставляемая ей функциональность. Его не интересует алгоритм, исполняемый процедурой. Ко всему прочему, процедуры дают способ декомпозиции проблемы – выполнение одной части задачи программирования в процедуре, а другой – в программе, вызывающей эту процедуру. Таким образом, существование процедур – довольно большой шаг к улавливанию смысла абстракции.

К сожалению, одни лишь процедуры не образуют достаточно богатый словарь абстракций. Упомянутые выше абстрактные объекты данных и управляющие конструкции абстрактной машины невозможно адекватно представить независимыми процедурами. Поскольку мы рассматриваем абстракции в контексте структурного программирования, опустим обсуждение абстракций управления.

Это подводит нас к концепции абстрактного типа данных, центрального при проектировании языка. *Абстрактный тип данных* определяет класс абстрактных объектов, который полностью характеризуется операциями, применимыми к этим объектам. Это означает, что абстрактный тип данных можно определить характерными для этого типа операциями.

Мы полагаем, что описанная выше концепция улавливает фундаментальные свойства абстрактных объектов. Когда программист использует абстрактный объект данных, его интересует только демонстрируемое этим

объектом поведение, но не детали реализации этого поведения. Поведение же объекта отражено в наборе характерных операций. Информация о реализации, например как объект представлен в памяти, нужна только при определении того, как реализуются характерные операции. Пользователю объекта не нужно ни знать, ни предоставлять такую информацию.

Абстрактные типы должны быть очень похожи на встроенные типы, предоставляемые языком программирования. Пользователю встроенного типа, например **integer** или **integer array**, интересно только создание объектов этого типа и выполнение операций над ними. Обычно его не интересует, как представлены объекты данных, а операции над объектами он рассматривает как неделимые и атомарные, хотя в действительности для их реализации может потребоваться несколько машинных инструкций. Кроме того, ему (вообще говоря) не разрешено производить декомпозицию объектов. Рассмотрим, к примеру, встроенный тип **integer**. Программист хочет объявлять объекты типа **integer** и выполнять над ними обычные арифметические действия. Обычно его не интересует представление целочисленного объекта в виде строки битов, и он не может как-то воспользоваться форматом машинного слова, составленного из битов. Кроме того, он хотел бы, чтобы язык защищал его от досадного неправильного использования типов (например, сложения целого числа с литерой): либо обрабатывал такую операцию как ошибку, либо выполнял какое-то автоматическое преобразование типов.

В случае встроенного типа данных программист пользуется концепцией или абстракцией, реализованной на нижнем уровне детализации – в самом языке программирования и его компиляторе. Аналогично абстрактный тип данных используется на одном уровне, а реализуется на более низком, но этот более низкий уровень не появляется автоматически как часть языка, а должен быть реализован путем написания программы особого рода, называемой *кластером операций*, или просто кластером, которая определяет тип в терминах операций, которые можно над ним производить. Язык облегчает эту деятельность, позволяя использовать абстрактный тип данных и не требуя немедленного определения. Языковой процессор поддерживает абстрактные типы данных тем, что строит ссылки между использованием типа и его определением (которое может находиться раньше или позже) и навязывает представление о типе данных как об эквиваленте набора операций посредством очень строгой формы типизации данных.

Мы видим, что из концепции абстрактных типов данных следует, что большинство абстрактных операций в программе принадлежат наборам операций, характеризующих абстрактные типы. Мы будем использовать термин *функциональная абстракция* для обозначения тех абстрактных операций, которые не принадлежат никакому характерному набору. Функциональная абстракция будет реализована в виде композиции характерных операций одного или нескольких типов данных и будет поддерживаться обычным образом с помощью процедуры. Примером такой функциональной абстракции может служить процедура вычисления синуса. Реализацией синуса могло бы быть разложение в ряд Тейлора, выраженное в терминах характерных операций типа **real**.

39.3. Язык программирования

Теперь мы дадим неформальное описание языка программирования, который допускает использование и определение абстрактных типов данных. Он является упрощенным вариантом языка структурного программирования, разрабатываемого в МТИ. В его основе лежит преимущественно язык Pascal (Wirth, 1971), и во многих отношениях он является традиционным, однако есть и несколько важных отличий.

Язык предлагает *два* вида модулей, соответствующих двум формам абстракции: процедуры, которые поддерживают функциональные абстракции, и кластеры операций, которые поддерживают абстрактные типы данных. Каждый модуль транслируется (компилируется) отдельно.

В языке нет свободных переменных в обычном смысле. Внутри модулей единственные имена, которые свободны, т. е. определены вне модуля, – это имена других модулей: кластеров и процедур. Эти имена связываются на этапе трансляции с помощью каталога имен модулей, созданного программистом специально для этой цели. В оттранслированном модуле никаких имен, подлежащих связыванию, не остается.

В языке имеются только структурные управляющие конструкции. Не существует ни предложений **goto**, ни меток, есть только варианты конкатенации, выбора (**if**, **case**) и итерации (**while**). Структурный механизм обработки ошибок находится в процессе разработки. В этой работе о нем напоминает только наличие зарезервированного слова **error**.

Как именно язык допускает использование и определение абстрактных типов данных, лучше всего проиллюстрировать на примере. Мы выбрали следующую задачу: написать программу, Polish_gen, которая транслирует инфиксный язык в постфиксный (обратную польскую запись). Polish_gen должна быть универсальной программой, не делающей никаких предположений об устройствах ввода-вывода (или файлах). Она делает только следующие предположения о входном языке.

1. Входной язык имеет грамматику с операторным предшествованием.
2. Символом входного языка является либо произвольная строка букв и цифр, либо одна не буквенно-цифровая литера; пробелы разграничивают символы, но во всех остальных отношениях игнорируются.

Например, если Polish_gen получает на входе строку $a + b * (c + d)$, то на выходе должна породить строку $a\ b\ c\ d + * +$. Мы выбрали для примера эту задачу, потому что она сама и ее решение хорошо известны всем, кто интересуется языками программирования, и в то же время она достаточно сложна, чтобы проиллюстрировать использование многих абстракций.

39.4. Использование абстрактных типов данных

Процедура Polish_gen, показанная на рис. 39.1, выполняет описанную выше трансляцию. Она принимает три аргумента: input, объект абстрактного типа

infile, содержащий предложение на входном языке; output, объект абстрактного типа outfile, в который будет помещено предложение на выходном языке; и g, объект абстрактного типа grammar, используемый для распознавания символов входного языка и определяющий для них правила предшествования. Кроме того, Polish_gen использует локальные переменные абстрактных типов stack и token. Заметим, что все имена-типов-данных в Polish_gen являются свободными, равно как и «scan» – имя единственной функциональной абстракции в Polish_gen.

```
Polish_gen: procedure(input: infile,
                      output: outfile, g: grammar);

t : token;
mustscan: boolean;
s: stack(token) ;

mustscan := true;
stack$push (s, token (g, grammar$eof (g)));
while ¬stack$empty (s) do
   if mustscan
      then t := scan(input, g)
      else mustscan := true;
   if token$is op(t)
      then
         case token$prec rel(stack$top(s), t) of
            "<":: stack$push(s, t);
            "=":: stack$erasetop(s);
            ">"::
               begin
                  outfile$out str(output,
                     token$symbol(stack$pop(s)));
                  mustscan := false;
               end
         otherwise error;
      else outfile$out str(output, token$symbol(t));
   end
   outfile$close(output);
   return;
end Polish_gen
```

Рис. 39.1

В языке используется один и тот же синтаксис для объявления переменных абстрактных типов данных и переменных примитивных типов. Синтаксически различаются объявления, подразумевающие и не подразумевающие создание объекта. Например,

t: token

означает, что t – имя переменной, содержащей объект абстрактного типа token, но при этом никакой объект token не создается, поэтому значение t в начальный момент не определено. Таким образом, переменная t объявляется так же, как переменная mustscan в предложении

mustscan: boolean

Круглые скобки после имени типа означают создание объекта. Например, предложение

```
s: stack(token)
```

означает, что s – имя переменной, содержащей объект абстрактного типа stack, и что объект stack следует создать и сохранить в s. Информация, необходимая для создания объекта, передается в списке параметров; в данном примере единственный параметр token определяет тип элемента, который можно поместить в стек s. Объявление stack похоже на объявление массива, например «**array**[1..10] **of characters**», тем, что и там, и там необходимо задавать тип элементов.

Язык строго типизирован, т. е. существует всего три допустимых способа использования абстрактного объекта:

1. К абстрактному объекту можно применять операции, определяющие его абстрактный тип.
2. Абстрактный объект можно передавать в качестве параметра процедуры. В этом случае тип фактического аргумента, передаваемого вызывающей процедурой, должен в точности совпадать с типом соответствующего формального параметра вызываемой процедуры.
3. Абстрактный объект можно присвоить переменной, но только если в ее объявлении было указано, что она содержит объект этого типа.

Применение операции к абстрактному объекту обозначается вызовом операции, в котором используется составное имя, например:

```
grammar$eof(g)
stack$push(s, t)
token$is op(t)
```

Первая часть составного имени определяет абстрактный тип, которому принадлежит операция, а вторая – саму операцию. У вызова операции всегда имеется по крайней мере один параметр – объект абстрактного типа, которому операция принадлежит.

Есть несколько причин для включения имени-типа в вызов операции. Во-первых, поскольку вызов операции может иметь несколько параметров разных абстрактных типов, отсутствие имени-типа может привести к неоднозначности: к какому объекту операция применяется. Во-вторых, использование составных имен позволяет определять одноименные операции в разных типах данных, не опасаясь конфликта идентификаторов. В-третьих, мы полагаем, что префикс в виде имени-типа улучшает удобочитаемость программ, коль скоро читатель освоился с нотацией. Мало того что сразу виден тип операции, так еще и вызовы операций очевидным образом отличаются от вызовов процедур.

Предложение

```
t := scan(input, g)
```

демонстрирует как передачу абстрактных объектов в параметрах, так и присваивание абстрактного объекта переменной. Процедура scan, показанная

на рис. 39.2, ожидает объекты типа infile и grammar в качестве аргументов и возвращает объект типа token, который затем сохраняется в переменной r типа token.

```
scan: procedure(input: infile, g: grammar) returns token;
  newsymb: string;
  ch: char;
  ch := infile$get(input)
  while ch = " " do ch := infile$get(input); end
  if infile$eof(input)
    then return token(g, grammar$eof(g));
  newsymb := unit string(ch);
  if alphanumeric(ch) then
    while alphanumeric(infile$peek(input)) do
      newsymb := newsymb concat infile$get(input);
    end
  return token(g, newsymb);
end scan
```

Рис. 39.2

Мы объяснили, что объекты можно создавать прямо в объявлении переменной. Но их можно создавать и независимо от объявления переменной. О создании объекта (не важно, внутри или вне объявления) свидетельствует появление имени типа, за которым следуют круглые скобки. Например, последняя строка процедуры scan

```
token(g, newsymb)
```

означает, что следует создать объект типа token, представляющий только что просканированный символ; информация, необходимая для создания объекта (грамматика и только что просканированный символ), передается в списке параметров.

Теперь можно привести краткое описание логики Polish_gen. В Polish_gen используется функциональная абстракция scan для получения символа грамматики из входной строки. Scan возвращает символ в виде объекта типа token, который предназначен специально для эффективного выполнения без раскрытия информации о том, как символы представлены в грамматике.

Polish_gen stores сохраняет лексему (token), содержащую просканированный символ, в переменной t. Если в t хранится лексема, представляющая идентификатор (например, «a»), а не оператор (например, «+»), то этот идентификатор помещается в выходной файл немедленно. В противном случае лексема на вершине стека сравнивается с t, чтобы определить, что чему предшествует. Если они связаны отношением предшествования «<», то t помещается в стек (например, «+» < «*»). Если они связаны отношением предшествования «=», то и t, и лексема на вершине стека отбрасываются (например, «(» = «)»). Если они связаны отношением предшествования «>», то оператор, хранящийся в лексеме на вершине стека, добавляется в конец выходного файла, открывая для доступа новую лексему на вершине стека. Поскольку эта операторная лексема может предшествовать t, то булева переменная mustscan используется, чтобы предотвратить сканирование нового

символа и гарантировать, что следующее сравнение производится с текущим значением t. Так как зависящее от грамматики представление символа конца файла (grammar$eof(g)) было помещено в стек в начальный момент, то стек может стать пустым, что приведет к завершению Polish_gen, только когда совпадение с лексемой вызвано исчерпанием входа. (Мы сделали упрощающее предположение о том, что вход содержит допустимое предложение на инфиксном языке.)

Процедура scan получает литеры из входного файла с помощью операций, определяющих абстрактный тип infile. Она пользуется типами данных **char** и **string** и операциями этих типов. Хотя эти типы показаны как встроенные, они вполне могли бы быть и абстрактными. Тогда, например, встроенный предикат **alphanumeric** следовало бы записать в виде char$alphanumeric. Изменился бы только синтаксис; семантика и использование типов в любом случае остались бы прежними.

Подводя итоги, можно отметить, что в Polish_gen используется пять абстракций данных: infile, outfile, grammar, token и stack, плюс одна функциональная абстракция, scan. Мощь абстракций данных иллюстрируется типами infile и outfile, которые нужны, чтобы изолировать Polish_gen от любых физических фактов, касающихся ее входа и выхода соответственно. Polish_gen ничего не знает ни об используемых устройствах ввода-вывода, если ввод-вывод действительно имеет место, ни о том, как литеры представлены в устройствах. Знает она ровно столько, сколько необходимо для ее целей. Что касается вывода, то она знает, как добавить строку литер (outfile$outstr) и как сказать, что вывод завершен (outfile$close). Что касается ввода, то она знает, как получить следующую литеру (infile$get), как посмотреть на следующую литеру, не удаляя ее из ввода (infile$peek), и как распознать конец ввода (infile$eof). (Отметим, что для правильной работы scan тип infile должен возвращать отличный от пробела, не буквенно-цифровой символ при любом вызове infile$get или infile$peek после достижения конца файла.) В любом случае она знает имена операций, предоставляющих эти услуги.

39.5. Определение абстрактных типов данных

В этом разделе мы опишем программный объект – кластер операций, – трансляция которого дает реализацию типа. Кластер содержит код, реализующий каждую из характерных операций, и тем самым воплощает в жизнь идею о том, что тип данных определяется своим набором операций.

В качестве примера рассмотрим абстрактный тип данных stack, используемый в программе Polish gen. Кластер, поддерживающий стеки, показан на рис. 39.3. Этот кластер реализует очень общий вид стекового объекта, в котором тип элементов стека заранее неизвестен. Параметр element_type операции cluster определяет тип элементов, который будет содержать конкретный объект стека. В первой части определения кластера находится очень краткое описание интерфейса, который кластер предоставляет своим поль-

зователям. В этом интерфейсе определено имя кластера, параметры, необходимые для создания экземпляра кластера (объект абстрактного типа, реализуемого кластером) и список операций, определяющий тип, который реализован кластером, например:

stack: **cluster**(element type: **type**)
  is push, pop, top, erasetop, empty

```
stack: cluster(element type: type)
                    is push, pop, top, erasetop, empty ;
   rep(type param: type) = (tp: integer;
      e_type: type; stk: array[1..10] of type_param);
   create
      s: rep(element_type);
      s.tp := 0;
      s.e_type := element_type;
      return s;
      end
   push: operation(s: rep, v: s.e_type);
      s.tp := s.tp+1;
      s.stk[s.tp] := v;
      return;
      end
   pop: operation(s: rep) returns s.e_type;
      if s.tp = 0 then error;
      s.tp := s.tp–1;
      return s.stk[s.tp+1];
      end
   top: operation(s: rep) returns s.e_type;
      if s.tp = 0 then error;
      return s.stk[s.tp];
      end
   erasetop: operation(s: rep);
      if s.tp = 0 then error;
      s.tp := s.tp–1;
      return;
      end
   empty: operation(s: rep) returns boolean;
      return s.tp = 0;
      end
end stack
```

Рис. 39.3

Использование зарезервированного слова **is** подчеркивает идею о том, что тип данных характеризуется группой операций.

Далее в определении кластера описывается, как именно поддерживается абстрактный тип. Здесь мы видим три части: представление объекта, код создания объектов и определения операций.

39.5.1. Представление объекта. Пользователи абстрактного типа данных видят объекты этого типа как неделимые сущности. Однако внутри кластера объекты разлагаются на элементы более примитивных типов. *Описание* **rep**

rep[(⟨параметры-rep⟩)] = ⟨определение-типа⟩

определяет новый тип, обозначаемый зарезервированным словом **rep**, доступным только внутри кластера, и описывает, как здесь выглядят объекты. ⟨определение-типа⟩ определяет шаблон, который позволяет создавать и подвергать декомпозиции объекты данного типа. В нем используются методы структурирования данных, предоставляемые языком: массивы (возможно, неограниченные) или записи Pascal. Факультативная часть («[ ]») ⟨параметры-rep⟩ позволяет отложить задание некоторых аспектов ⟨определения-типа⟩ до момента создания экземпляра rep. Рассмотрим описание **rep** для кластера стека:

rep(type param: **type**) = (tp: **integer**; e_type: **type**; stk: **array**[1..] **of** type_param)

⟨определение-типа⟩ говорит, что объект стека представлен записью, содержащей три компонента с именами tp, stk и e_type. Параметр type_param определяет тип элемента, который может храниться в неограниченном массиве stk, содержащем элементы, помещенные в стек. Тот же самый тип хранится в компоненте e_type и используется для проверки типов, как будет описано ниже. В компоненте tp хранится индекс элемента на вершине стека.

39.5.2. Создание объекта. Зарезервированное слово **create** обозначает секцию create-code – код, который будет выполняться при создании объекта абстрактного типа. Кластер можно рассматривать как процедуру, телом которой является секция create-code. Когда пользователь говорит, что нужно создать объект абстрактного типа, например

s: stack(token)

вызывается (во время выполнения) create-code, что приводит к исполнению тела процедуры. Параметры кластера – это на самом деле параметры create-code. Поскольку никаких свободных переменных, кроме ссылок на внешние модули, нет, эти параметры недоступны ни операциям, ни ⟨определению-типа⟩ в **rep**. Поэтому любую информацию о параметрах, которую необходимо сохранить, необходимо явно вставлять в каждый экземпляр **rep**.

Код в кластере stack типичен для create-code. Во-первых, создается объект типа rep, т. е. выделяется память для хранения объекта, определенного в **rep**. Затем в объекте сохраняются начальные значения. И наконец, объект возвращается вызывающей стороне. При возврате объекта его тип изменяется с **rep** на абстрактный тип, определенный кластером.

39.5.3. Операции. Оставшаяся часть кластера содержит группу определений операций, где находятся реализации допустимых операций над типом данных. Определения операций похожи на определения обыкновенных процедур, но имеют доступ к представлению **rep** кластера, что позволяет им осуществлять декомпозицию объекта типа кластера. Сами операции не являются модулями; транслятор принимает их только в составе кластера.

Любая операция имеет по крайней мере один параметр – типа **rep**. Поскольку кластер может одновременно поддерживать много объектов определяемого им типа, этот параметр сообщает операции о том конкретном объекте, к которому она применяется. Заметим, что тип этого параметра меняется с абстрактного типа на тип **rep** в момент передачи между вызывающей стороной и операцией.

Поскольку язык строго типизирован, тип объектов, помещаемых в стек, необходимо проверять на согласованность с типом элементов, которые могут в этом стеке храниться. Это требование согласованности синтаксически задается путем объявления, говорящего, что тип второго аргумента push должен совпадать с компонентом e_type представления **rep** объекта stack, который передается в первом аргументе push. Транслятор может сгенерировать код, проверяющий совпадение типов во время выполнения и выдающий ошибку, если это не так.

39.6. Управление использованием информации

Абстрактные типы данных задумывались как способ освободить программиста от забот о несущественных деталях при использовании своих абстракций данных. Но на самом деле мы пошли дальше. Поскольку язык строго типизирован, пользователь вообще не может воспользоваться никакими деталями реализации. В этом разделе мы обсудим, какие преимущества вытекают из данного ограничения: программы становятся более модульными, их проще понять, модифицировать и сопровождать, а также доказывать их правильность.

Token – хороший пример типа, созданного для управления доступом к деталям реализации. Можно было бы не создавать новый тип, а написать Polish_gen так, чтобы она принимала строки от scan, сохраняла их в строке и сравнивала строки для определения предшествования (с помощью подходящей операции grammar$prec_rel). Такое решение было бы неэффективно. Поскольку матрицу предшествования можно индексировать положениями операторов в таблице зарезервированных слов грамматики, эффективная реализация могла бы искать строку литер только один раз, чтобы выяснить, является ли данный символ символом оператора, и если да, то использовать в Polish_gen индекс оператора.

Однако это раскрывает информацию о представлении грамматики типом grammar. Если Polish_gen или еще какой-то модуль, работающий с grammar, пользуется этой информацией, то при нормальном сопровождении и модификации кластера grammar могут возникнуть ошибки, которые будет трудно найти (Parnas, 1971). Поэтому введен новый тип token, который призван ограничить распространение информации о том, как представлена грамматика. Теперь изменение кластера grammar может повлиять только на кластер token, который не делает никаких предположений об индексе, полученном от grammar. Если возникнет ошибка при поиске информации о предшествовании (например, индекс выйдет за допустимые границы), то ее причиной может быть только что-то в кластере token или grammar.

На самом деле выбор реализации лексем – к примеру, представлять ли лексему целым числом или строкой литер, – проектное решение. Это решение можно отложить до момента определения кластера для лексем, и его не нужно принимать на этапе кодирования Polish_gen. Поэтому при программировании Polish_gen можно применять один из принципов Дейкстры: при

построении программы принимать только одно решение за раз (Dijkstra, 1972). Следование этому принципу упрощает логику Polish_gen, а значит, также ее понимание и сопровождение.

Запрещая доступ к представлению, мы также облегчаем доказательство правильности программы. Оно разбивается на две части: доказать, что кластер правильно реализует тип, и доказать, что программа правильно использует этот тип. Только при доказательстве первого утверждения нужно учитывать детали реализации объектов типа; доказательство же второго основывается лишь на абстрактных свойствах типов, которые можно выразить в терминах соотношений между характерными операциями каждого типа.

<...>

40 Мифический человеко-месяц (1956)[1]

Фредерик Ф. Брукс

По мере того как компьютерные системы становились все больше и сложнее, со всей очевидностью проявился неприятный факт. Писать программное обеспечение было трудно, а еще труднее было оценить, сколько времени займет его написание. Протоколы, разработанные для управления техническими проектами, казались неприменимыми к программам. Специалист по информатике Том Читэм говаривал: «Разница между гражданским строительством и разработкой программного обеспечения заключается в том, что если вам говорят, что мост наполовину построен, то вы можете пойти и посмотреть». (См. синдром 90%-ной готовности Ройса, стр. 409.)

Фредерик Ф. (Фред) Брукс (родился в 1931 году) был докторантом Говарда Эйкена в Гарварде и работал над докторской диссертацией (Brooks 1956) на тему анализа обработки деловых данных. В 1956 году он поступил на работу в компанию IBM, которая в то время производила серию компьютеров все возрастающей мощности, каждый из которых был несовместим со своими предшественниками. Осознав, что (как и предсказывала Хоппер) стоимость программного обеспечения станет неподъемной, если клиентам придется переписывать свои программы всякий раз при переходе на новую машину, он спроектировал линейку System/360, так что программы, работавшие на одной машине, могли быть легко перенесены на более новые и мощные мо-

---

[1] Текст печатается по изданию: *Брукс-мл. Ф. П.* Как проектируются и создаются программные комплексы. Мифический человеко-месяц / пер. с англ. Н. А. Черемных; под ред. А. П. Ершова (с незначительной правкой). М.: Наука, 1979.

дели, даже если в них использовалась другая реализация оборудования (тут помогло микропрограммирование, см. стр. 221). Брукс предложил термин «компьютерная архитектура», означающий структуру компьютерной системы с точки зрения программного обеспечения.

Когда в 1964 году он уходил из IBM, Томас Уотсон мл. спросил его, почему программными проектами так трудно управлять. «Мифический человеко-месяц» – часть ответа Брукса на этот вопрос. Это эссе – одно из нескольких в одноименной книге; все их стоит прочитать, но именно это стало культовым. Удивительно, но и до сих пор можно услышать, что для скорейшего завершения проекта нужно добавить рабочую силу; при этом не берут в расчет ни время, которое новым работникам придется потратить, чтобы войти в курс дела, ни проблемы координации большего количества участников.

После ухода из IBM Брукс основал факультет информатики в университете Северной Каролины и там работал до завершения своей карьеры. Его группа внесла значительный вклад в машинную графику, но большая часть его нетленного наследия связана с прозорливостью в области программной инженерии. (До сих пор «все программисты – оптимисты».) Десять лет спустя он опубликовал еще один глубокий анализ «асфальтовой топи» программной инженерии, «Серебряной пули нет: существенное и случайное в программной инженерии» (Brooks 1987), в которой пытался объяснить, почему – вопреки напряженным усилиям и грандиозным обещаниям – ни одна из технологий, предложенных для ускорения процесса производства программного обеспечения, похоже, так и не смогла сделать его ни проще, ни быстрее и не добилась заметных успехов в повышении качества результата. Модульность, повторное использование и библиотеки с открытым исходным кодом в последние годы помогли, но ни в коей мере не отправили в утиль смиренную мудрость этого эссе.

Почти все программистские проекты страдают скорее из-за нехватки времени, нежели из-за отсутствия каких-либо других ресурсов. Почему эта причина бедствий является столь всеобщей?

Во-первых, наши методы оценки весьма несовершенны. Строго говоря, они отражают некоторое неявно высказываемое и в корне неверное допущение, что все будет идти хорошо.

Во-вторых, наши методы оценки ошибочно путают усилия с достижениями, прячась за допущение, что человек и месяц взаимозаменяемы.

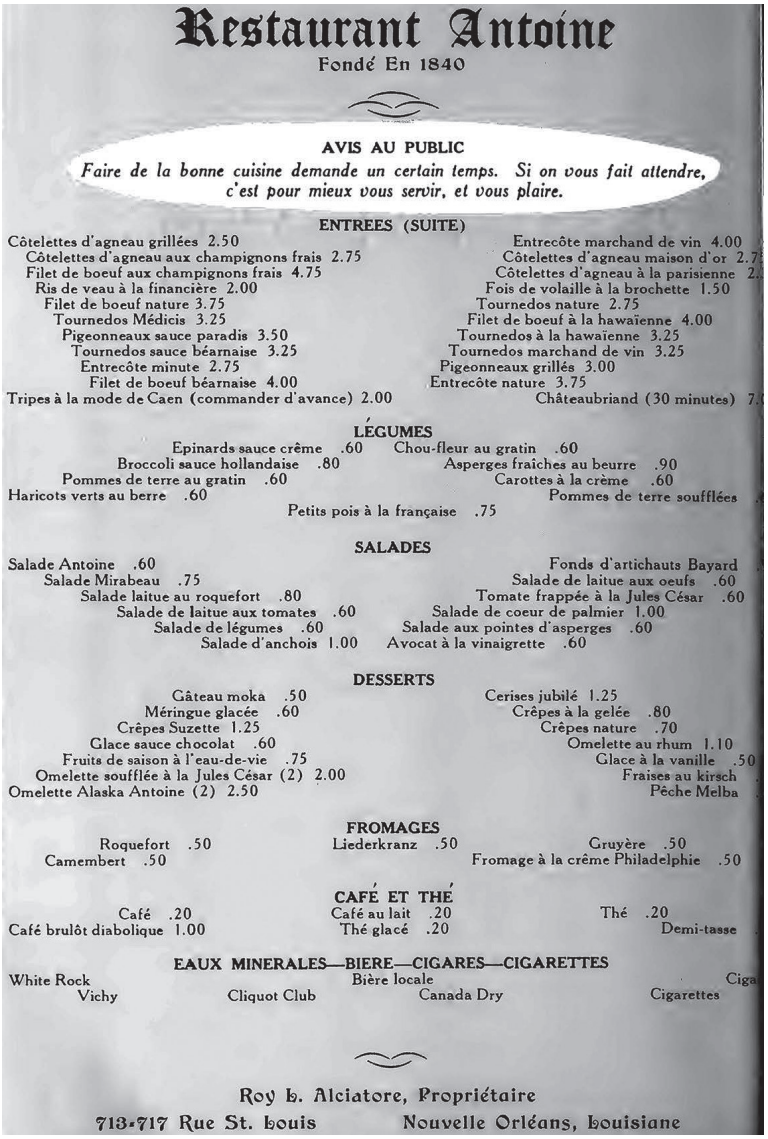
В-третьих, отсутствие уверенности в наших оценках ведет к отсутствию у руководителей программистских проектов вежливого упрямства, свойственного шеф-повару ресторана «Антуан».

В-четвертых, управление ходом разработки плохо организовано. Методы, давно опробованные и даже рутинные в других технических дисциплинах, в технологии программирования рассматриваются как радикальные новшества.

В-пятых, когда обнаруживается отставание от графика, естественная (и традиционная) реакция руководителя – добавить рабочей силы. А это,

аналогично попытке заливать огонь бензином, только ухудшает дело, причем значительно. Чем сильнее огонь, тем больше требуется бензина, круг замыкается, и последствия плачевны.

Наблюдение за выполнением графиков будет темой отдельной главы. Давайте пока подробнее рассмотрим другие аспекты проблемы.



Restaurant Antoine

Fondé En 1840

AVIS AU PUBLIC

Faire de la bonne cuisine demande un certain temps. Si on vous fait attendre, c'est pour mieux vous servir, et vous plaire.

ENTREES (SUITE)

Côtelettes d'agneau grillées 2.50
Côtelettes d'agneau aux champignons frais 2.75
Filet de boeuf aux champignons frais 4.75
Ris de veau à la financière 2.00
Filet de boeuf nature 3.75
Tournedos Médicis 3.25
Pigeonneaux sauce paradis 3.50
Tournedos sauce béarnaise 3.25
Entrecôte minute 2.75
Filet de boeuf béarnaise 4.00
Tripes à la mode de Caen (commander d'avance) 2.00
Entrecôte marchand de vin 4.00
Côtelettes d'agneau maison d'or 2.7
Côtelettes d'agneau à la parisienne 2.
Fois de volaille à la brochette 1.50
Tournedos nature 2.75
Filet de boeuf à la hawaïenne 4.00
Tournedos à la hawaïenne 3.25
Tournedos marchand de vin 3.25
Pigeonneaux grillés 3.00
Entrecôte nature 3.75
Châteaubriand (30 minutes) 7.

LÉGUMES

Epinards sauce crême .60
Broccoli sauce hollandaise .80
Pommes de terre au gratin .60
Haricots verts au berre .60
Chou-fleur au gratin .60
Asperges fraîches au beurre .90
Carottes à la crème .60
Pommes de terre soufflées .
Petits pois à la française .75

SALADES

Salade Antoine .60
Salade Mirabeau .75
Salade laitue au roquefort .80
Salade de laitue aux tomates .60
Salade de légumes .60
Salade d'anchois 1.00
Fonds d'artichauts Bayard .
Salade de laitue aux oeufs .60
Tomate frappée à la Jules César .60
Salade de coeur de palmier 1.00
Salade aux pointes d'asperges .60
Avocat à la vinaigrette .60

DESSERTS

Gâteau moka .50
Méringue glacée .60
Crêpes Suzette 1.25
Glace sauce chocolat .60
Fruits de saison à l'eau-de-vie .75
Omelette soufflée à la Jules César (2) 2.00
Omelette Alaska Antoine (2) 2.50
Cerises jubilé 1.25
Crêpes à la gelée .80
Crêpes nature .70
Omelette au rhum 1.10
Glace à la vanille .50
Fraises au kirsch .
Pêche Melba .

FROMAGES

Roquefort .50
Camembert .50
Liederkranz .50
Gruyère .50
Fromage à la crême Philadelphie .50

CAFÉ ET THÉ

Café .20
Café brulôt diabolique 1.00
Café au lait .20
Thé glacé .20
Thé .20
Demi-tasse .

EAUX MINERALES—BIERE—CIGARES—CIGARETTES

White Rock
Vichy
Cliquot Club
Bière locale
Canada Dry
Ciga
Cigarettes

Roy b. Alciatore, Propriétaire

713-717 Rue St. Louis Nouvelle Orléans, Louisiane

[Примечание редактора: текст в верхней части меню гласит: «Хорошая кухня требует времени. Если Вы готовы подождать, мы обслужим Вас гораздо лучше, и Вы получите большее удовольствие».]

40.1. Оптимизм

Все программисты – оптимисты. Может быть, это современное чародейство особенно привлекает тех, кто верит в добрых фей и счастливый конец. Может быть, стократное крушение надежд способен пережить только тот, кто привык добиваться поставленной цели. Или, может быть, все дело в том, что вычислительные машины молоды, а юность всегда оптимистична. Однако как ни объясняй, но слышим мы одно и то же:

«К этому сроку программа обязательно пройдет» или «Я только что нашел последнюю ошибку».

Итак, первое ложное допущение, лежащее в основе планирования деятельности системных программистов, заключается в том, что *все будет в порядке* – т. е. что *выполнение каждого задания займет ровно столько времени, сколько оно «должно» занять*.

Широкое распространение оптимизма среди программистов заслуживает более глубокого анализа. Дороти Сейерс в своей прекрасной книге «Мысль творца» (The Mind of the Maker) подразделяет творческую деятельность на три этапа: идея, реализация и взаимодействие. Книга, вычислительная машина или программа сначала существуют как идея, вне времени и пространства, только в мозгу своего создателя, но в совершенно законченном виде. Замысел реализуется во времени и пространстве посредством пера, чернил и бумаги или же с помощью проводов, полупроводниковых схем и ферритовых сердечников. Процесс создания завершается, когда кто-то другой читает книгу, использует вычислительную машину, пропуская через нее программу, тем самым взаимодействуя с замыслом творца.

Это описание творческой деятельности человека поможет нам при решении нашей сегодняшней задачи. Для людей творческого труда неполнота и противоречивость их идей становятся ясными только в процессе реализации; таким образом, описание, эксперимент, «решение» весьма важны для теоретика.

Во многих видах творческой деятельности средства реализации несовершенны. Древесина раскалывается, масляные краски засыхают, в электрических цепях происходит замыкание. Такое физическое несовершенство средств накладывает свои ограничения на предлагаемые идеи и, кроме того, может вызвать непредвиденные трудности в процессе реализации.

Реализация дастся нам потом и кровью как из-за несовершенства физических средств, так и вследствие неадекватности наших основополагающих идей. Мы склонны сваливать вину за большинство затруднений на средства реализации, поскольку они не «наши», в отличие от «наших» идей, которые нам трудно оценивать беспристрастно.

Программист, однако, имеет дело с очень податливым материалом: концепциями и весьма гибкими представлениями. Поскольку материал столь послушен, мы не ожидаем особых затруднений при его реализации, и отсюда наш всепроникающий оптимизм. Так как наши идеи неверны, мы получаем ошибки. Следовательно, наш оптимизм не обоснован.

Допущение о том, что все будет в порядке, имеет вполне определенный вероятностный смысл для отдельно взятой задачи. Действительно, все может

идти по плану, поскольку существует вероятностное распределение появления отставания от графика, а стало быть, есть конечная вероятность, что отставания не будет. Большой программистский проект, однако, включает в себя много отдельных задач, каждая из которых может зависеть от окончания другой. Вероятность того, что каждая задача будет идти нормально, становится исчезающе малой.

40.2. Человеко-месяц

Вторая ложная предпосылка нашла свое отражение в самой единице, используемой при оценке производительности и составлении графиков, а именно в человеко-месяце. Стоимость проекта действительно зависит от числа людей и от числа месяцев, но его успешность – нет. *Следовательно, человеко-месяц как единица измерения объема работы является опасным и вводящим в заблуждение мифом.* Этот миф основывается на предпосылке, что люди и месяцы взаимозаменяемы.

Человек и месяц взаимозаменяемы только в том случае, когда задание можно распределить между несколькими работниками, никак не зависящими друг от друга (рис. 40.1). Это справедливо на уборке пшеницы или сборе хлопка, но даже приблизительно неверно в системном программировании.

Когда задание нельзя распределить между несколькими работниками из-за ограничений на последовательность выполняемых работ, привлечение дополнительных сил не влияет на график его выполнения (рис. 40.2). Чтобы выносить ребенка, нужно девять месяцев, независимо от того, сколько женщин будет к этому привлечено. Это характерно для многих программных заданий из-за последовательной природы отладки. В заданиях, допускающих разбиение на взаимосвязанные подзадачи, следует прибавлять ко всему объему предстоящей работы затраты на обеспечение связи. Поэтому здесь даже лучшие результаты всегда несколько хуже, чем в случае эквивалентного обмена человека на месяц (рис. 40.3).

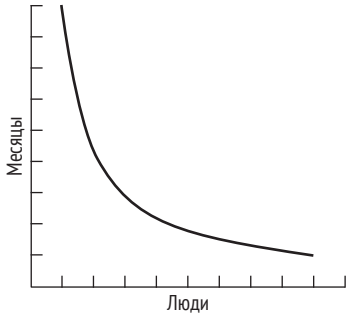


Рис. 40.1. Время и число работников – полностью распределяемое задание

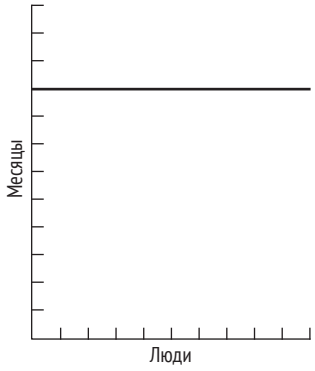


Рис. 40.2. Время и число работников – нераспределяемое задание

Дополнительные затраты на обеспечение связи слагаются из двух частей: обучения и взаимосвязи. Каждый работник должен получить определенные технические навыки, познакомиться с целями и задачами, общей стратегией и планом работы. Такое обучение нельзя разбить на отдельные части, так что эта часть дополнительных затрат изменяется линейно в зависимости от числа работников.

С установлением взаимосвязи дело обстоит хуже. Если каждая часть задачи должна отдельно координироваться с каждой другой частью, затраты возрастают как $n(n-1)/2$. Между тремя работниками в три раза больше попарных взаимосвязей, чем между двумя; между четырьмя – в шесть раз больше. Если, кроме того, для совместного решения вопросов нужно проводить совещания трех, четырех и более работников, ситуация становится еще хуже. Дополнительные затраты на обеспечение связи могут полностью свести на нет эффект разбиения первоначальной задачи на части, что приводит нас к ситуации, показанной на рис. 40.4.

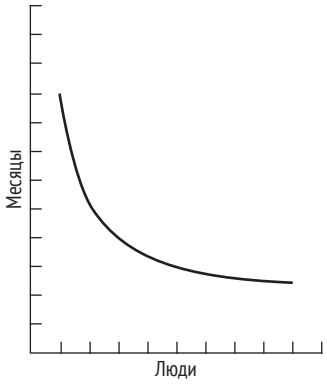


Рис. 40.3. Время и число работников – распределяемое задание, требующее связей между частями

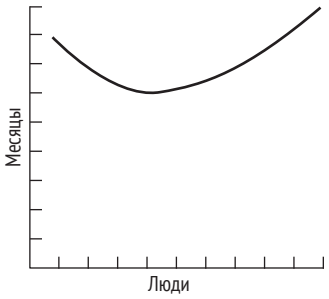


Рис. 40.4. Время и число работников – задача со сложными взаимосвязями

Так как разработка программного обеспечения – по самой своей сути деятельность системная, т. е. задача на сложные взаимосвязи, то затраты на их обеспечение велики и быстро перевешивают ту экономию затрачиваемого на задачу времени, которая достигается благодаря разбиению ее на части. Добавление людей лишь удлиняет сроки выполнения работ, а не укорачивает их.

40.3. Комплексное тестирование

Ни одна часть графика работ не связана так сильно ограничениями на их последовательность, как отладка компонент и комплексное тестирование. Очевидно, что требуемое время зависит от числа встречающихся ошибок и легкости их обнаружения. Будучи оптимистами, мы обычно ожидаем, что

ошибок будет меньше, чем это оказывается в действительности. Именно поэтому тестирование чаще всего не укладывается в график.

В течение нескольких лет я успешно применял следующее практическое правило планирования работ по созданию программного обеспечения:

планирование – 1/3,
кодирование –1/6,
тестирование компонент и раннее комплексное тестирование – 1/4,
системное (комплексное) тестирование всех компонент – 1/4.

Это разбиение отличается от общепринятого по нескольким важным показателям:

1. Доля, отведенная планированию, больше обычного. Но даже в этом случае времени едва хватает на написание подробных и надежных спецификаций и вовсе недостает на разработку и внедрение существенно новых методов.
2. Половина времени отводится на отладку написанной программы, что гораздо больше обычного.
3. Часть, которую легко оценить, т. е. собственно кодирование, занимает только одну шестую графика.

Знакомясь с проектами, планируемыми по общепринятой методике, я обнаружил, что по графику только в некоторых из них половина времени отводилась на тестирование, но в действительности так получалось в большинстве проектов. Многие проекты до этапа комплексного тестирования еще как-то укладывались в график.

Проект оказывается в особенно бедственном положении, если отводится недостаточно времени на комплексное тестирование. Так как задержка приходится на конец графика, то никто и не ожидает никаких неприятностей почти до самой даты окончания работ. Плохие новости обрушиваются на заказчиков и руководителей внезапно и чаще всего слишком поздно.

Кроме того, задержка в этот период влечет особенно суровые финансовые и психологические последствия. Штаты проекта полностью укомплектованы, и затраты уже достигли предела. И что еще серьезнее, разрабатываемые программы должны обеспечивать другие виды деятельности, например поставку вычислительных машин, ввод в действие новой системы, работу новых устройств и т. д., а так как почти всегда именно поставка программного обеспечения является последним этапом разработки, то эта задержка обходится очень дорого и вызванные ею дополнительные расходы на практике могут значительно превышать все остальные. Поэтому так важно в первоначальном графике проекта отводить достаточно времени на комплексное тестирование.

40.4. Объективность оценки

Отметим, что настойчивость руководителя может определить график выполнения задания, но не в состоянии определить срок его действительного завершения. Омлет, обещанный через две минуты, может быть подан в срок, но

если за две минуты он еще не готов, у заказчика два выбора – подождать или съесть его сырым. Заказчики программного обеспечения находятся перед таким же выбором. Но у повара есть другой выход – он может прибавить огня. И зачастую омлет тогда уже ничто не может спасти – он подгорел с одной стороны и остался сырым с другой.

Я не считаю, что руководители программистских проектов обладают меньшей смелостью и настойчивостью, чем шеф-повар или же чем руководители других технических проектов, но составление фиктивных графиков, соответствующих установкам начальства, в нашей области распространено гораздо больше, чем где-либо еще в технике. Дело в том, что энергичная и убедительная защита своих оценок, которые выводятся не на основе количественных методов, подтверждается малым количеством данных и основывается преимущественно на интуиции руководителя, – вещь очень трудная и сопряженная со многими неудобствами.

По-видимому, нужно систематизировать и публиковать данные о производительности, о частоте ошибок, правила выведения оценок и т. д. Профессионалы только выиграют от возможности совместного использования таких данных.

До тех пор пока методы оценки не станут более надежными, руководителям придется проявлять твердость характера и защищать собственные оценки, следуя своей интуиции.

40.5. Нарастающие катастрофы с графиком

Что нужно предпринять, когда важный программистский проект не укладывается в график? Естественно, добавить рабочей силы. Как видно на рисунках (40.1–40.4), иногда это помогает, а иногда – нет.

Давайте рассмотрим пример. Допустим, что трудоемкость задачи оценена в 12 человеко-месяцев и троим сотрудникам отвели на нее 4 месяца, причем установили количественные вехи A, B, C и D, которых в соответствии с графиком нужно достичь в конце каждого месяца (рис. 40.5).

Допустим теперь, что первая контрольная точка достигнута только через два месяца (рис. 40.6). Перед какими альтернативами оказался руководитель?

1. Допустим, что задачу нужно сделать вовремя. Допустим также, что неверно оценено только время выполнения первой части (см. рис. 40.6). Тогда остается 9 человеко-месяцев работы и два месяца времени, т. е. задача требует 4,5 человека. Добавим двух людей к трем первоначальным.
2. Допустим, что задачу нужно сделать вовремя. Допустим также, что время выполнения было занижено, а реальное положение дел представлено на рис. 40.7. Тогда остается 18 человеко-месяцев работы и два месяца времени, т. е. понадобится 9 человек. Добавим к трем первоначальным работникам еще шестерых.
3. Составление нового графика. Мне нравится совет П. Фагга, опытного инженера, специалиста по вычислительной технике: «Не исправляйте

понемногу». Другими словами, делайте новый график достаточно свободным, чтобы работу можно было сделать тщательно и основательно без последующего перепланирования.

4. Ослабление задания. На практике это происходит довольно часто, когда рабочий коллектив вдруг обнаруживает отставание от графика. Если вторичная стоимость задержки очень высока, такое решение является единственно приемлемым. У руководителя есть только две возможности: или тщательно и формально сократить задание, составив новый график, или же просто наблюдать, как поспешное проектирование и неполное тестирование мало-помалу сокращают объем задачи.

Рис. 40.5

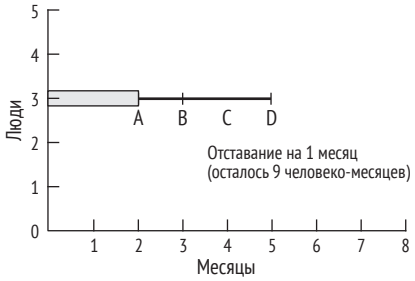


Рис. 40.6

В двух первых случаях настаивать на том, чтобы задача безо всяких изменений была закончена за четыре месяца, по меньшей мере ошибочно. Рассмотрим, например, какие помехи возникают в первом случае (рис. 40.8).

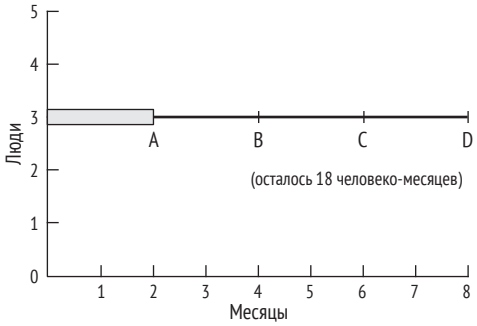


Рис. 40.7

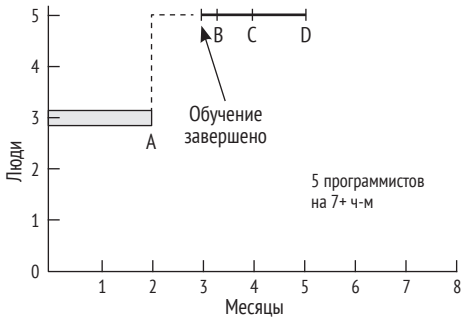


Рис. 40.8

Для того чтобы ввести в курс дела двух новых людей, пусть даже вполне компетентных и опытных, нужен один опытный работник. Если ему на это потребуется месяц, то *3 человеко-месяца будут отданы работе, никак не учтенной в первоначальных планах*. Кроме того, задачу, ранее поделенную на три части, теперь придется поделить на пять частей, следовательно, часть уже проделанной работы пропадет, а комплексное тестирование значительно удлинится. Значит, к концу третьего месяца останется больше 7 человеко-

месяцев работы, 5 обученных людей и месяц времени. Как видно по рис. 40.8, сроки выполнения задания не сократились, несмотря на появление новых людей (см. рис. 40.6).

Чтобы выйти из положения, даже рассматривая только время на обучение и не учитывая перераспределения задачи и дополнительного комплексного тестирования, потребовалось бы в конце второго месяца добавить не двоих, а четверых людей. Чтобы покрыть затраты на перераспределение и комплексное тестирование, нужен еще один человек. Теперь, однако, рабочий коллектив состоит не из троих, а, по крайней мере, из семерых людей, и принципы организации работы, распределения заданий и прочее уже отличаются от прежних не только количественно, но и качественно.

Заметим, что к концу третьего месяца дела обстоят очень плохо. Отметка «1 марта» все еще не достигнута, несмотря на усилия руководителя. Очень велик соблазн повторить весь цикл и привлечь дополнительных работников. Но это безумный путь.

В данном примере предполагалось, что только первая контрольная отметка была установлена неправильно. Если же 1 марта принять более осторожное допущение, что весь график (см. рис. 40.7) чрезмерно оптимистичен, то нужно к первоначальному коллективу добавить еще шесть человек. Предоставляем читателю в качестве упражнения вычислить, во что выльется обучение, перераспределение заданий, комплексное тестирование. Но нет никаких сомнений в том, что полученный продукт будет хуже, а его осуществление потребует больше времени, чем в случае, когда исходная группа также состояла из 3 человек, но график был бы другим.

Крайне упрощая, мы сформулируем закон Брукса:

Если программистский проект не укладывается в сроки,
то добавление рабочей силы только задержит его окончание.

Таким образом, мы развенчиваем миф о человеко-месяце. Число месяцев, отводимых на проект, зависит от ограничений на его линейность. Максимальное число людей зависит от числа независимых подзадач. Исходя из этих двух величин, можно построить график, рассчитанный на меньшее число людей и большее количество месяцев. (Единственная опасность заключается в том, что конечный продукт устареет.) Нельзя, однако, составить работоспособный график, используя больше людей и меньше месяцев. В большинстве программистских проектов дела шли скверно, скорее, из-за нехватки календарного времени, нежели по всем другим причинам, вместе взятым.

41 Ethernet: распределенная коммутация пакетов для локальных вычислительных сетей (1976)

Роберт Меткалф и Дэвид Р. Роджерс

Сети – вещь хитрая. Чтобы обеспечить безотказную доставку сообщений по ненадежным каналам связи, соединяющим потенциально враждебные узлы, протоколы должны предусматривать ситуации, которые даже трудно вообразить. К временным зависимостям нужно подходить скептически, а реализацию, которая может опираться на специализированное оборудование, не всегда легко верифицировать. Трудности нарастали по мере того, как компьютеры становились меньше и предприятия начали монтировать локальные сети с оборудованием от разных производителей.

Роберт (Боб) Меткалф (родился в 1946 году) получил степень бакалавра в МТИ в 1969 году, а затем поступил в аспирантуру в Гарварде как раз тогда, когда университеты начали присоединяться к сети ARPAnet. Гарвард отказался от предложения Меткалфа взять на себя обслуживание их подключения к ARPAnet, поэтому он согласился на аналогичную работу в МТИ. Этот опыт воспламенил в Меткалфе страсть к сетям, но ему пришлось уйти из препо-

давательского состава тогда только зарождавшегося в Гарварде факультета информатики. К тому же он остро переживал редкое по тем временам унижение – провал на защите докторской диссертации. Он поступил на работу в исследовательский центр компании Xerox Palo Alto (PARC), где разрабатывался первый персональный компьютер с возможностью подключения к сети, Alto. (Позже он представил переработанную диссертацию в Гарвард и получил степень доктора.)

В PARC Меткалф узнал о беспроводной пакетной сети Aloha Network, соединяющей Гавайские острова, и понял, что для решения тонких проблем, возникающих при соединении компьютеров проводами, следует ослабить контроль. Компьютеры должны быть объединены в сеть путем подключения их к общему коаксиальному кабелю с помощью физического отвода – шипа, врезаемого в медный сердечник в любом удобном месте кабеля. Для взаимодействия компьютеры должны были бы отправлять пакеты данных в пассивную сеть, где сообщения распространялись бы в обоих направлениях, и прослушивать сеть, отбирая предназначенные для них пакеты. Дэвид Боггс (родился в 1950 году), молодой радиоинженер, тоже работавший в PARC, применил свой опыт работы с беспроводной техникой, к идее Меткалфа. Поскольку подключенные к сети компьютеры не были синхронизированы, неизбежно возникали коллизии, или перекрытия пакетов, и для обработки этой ситуации была необходима повторная передача. Таким образом, коаксиальный кабель был аналогичен «эфиру», гипотетической среде, которой в XVII веке пытались объяснить волновую природу распространения света в вакууме. Название прижилось, а поскольку поддержкой идеи пассивной сети стали низкие лицензионные отчисления в пользу Xerox, Ethernet превратился в вездесущий стандарт. Меткалф и Боггс пошли по стезе предпринимательства.

41.1. Предварительные сведения

Распределенные вычисления можно охарактеризовать как спектр операций, различающихся по степени децентрализации, на одном конце которого находится сеть удаленных компьютеров, а на другом – многопроцессорность. Сеть удаленных компьютеров представляет собой слабо связанную совокупность ранее изолированных, территориально широко разнесенных и довольно крупных вычислительных систем. Многопроцессорность – это построение ранее монолитных и последовательных вычислительных систем из все большего количества меньших частей, работающих параллельно. Посередине этого спектра находятся локальные сети, т. е. совокупность взаимосвязанных компьютеров, организованная с целью добиться, с одной стороны, разделения ресурсов, присущего сетям компьютеров, а с другой – параллелизма, присущего многопроцессорности.

Расстояние между компьютерами и скорость передачи информации можно использовать, чтобы разбить спектр распределенных вычислений на несколько широких классов. Произведение расстояния на скорость передачи,

в настоящее время равное 1 гигабит-метр в секунду (1 Гбм/с), – показатель развитости современной техники связи, и можно ожидать, что со временем он будет расти (рис. 41.1).

Класс	Расстояние	Скорость передачи
Удаленные сети	> 10 км	< 0,1 Мб/с
Локальные сети	10–0,1 км	0,1–10 Мб/с
Многопроцессорность	< 0,1 км	> 10 Мб/с

Рис. 41.1

41.1.1. Сети удаленных компьютеров. Компьютерные сети начинались со взаимодействия *терминал–компьютер*, целью которого было подключение удаленных терминалов к центральному вычислительному устройству. По мере развития технологии связи *компьютер–компьютер* уже сами компьютеры стали использоваться для организации связи (Abramson 1970; Baran 1964; Rustin 1970). Как связь с использованием компьютеров в качестве пакетных коммутаторов (Heart et al. 1970, 1972; Kahn 1975; Metcalfe 1972b,c, 1973; Roberts and Wessler 1970b), так и связь между компьютерами для разделения ресурсов (Crocker et al. 1972; Farber 1973) получили развитие в результате разработки компьютерной сети ARPA.

Сеть Aloha в Гавайском университете первоначально создавалась, чтобы применить методы пакетной радиосвязи к организации связи между центральным компьютером и его терминалами, разбросанными по Гавайским островам. Многие терминалы теперь стали мини-компьютерами, сообщающимися между собой с использованием узла Menehune сети Aloha в качестве пакетного коммутатора. Menehune и ARPAnet Imp ныне соединены, что дает терминалам сети Aloha доступ к вычислительным ресурсам материковой части США.

Итак, компьютерные сети шагнули через континенты и океаны, связав крупные вычислительные центры по всему миру. А теперь они точно так же распространяются внутри одного здания и между зданиями, чтобы связать мини-компьютеры в офисах и лабораториях (Ashenhurst and Vonderohe 1975; Farber 1973, 1975a,b; Willard 1973).

41.1.2. Многопроцессорность. Первоначально многопроцессорность выступала в форме подключения контроллера ввода-вывода к большому центральному компьютеру; классическим примером является IBM ASP (Rustin 1970). Затем несколько центральных процессоров стали подключать к общей памяти, чтобы расширить возможность счетных приложений (Thornton 1970). Для некоторых таких приложений были разработаны более экзотические многопроцессорные архитектуры, например Illiac IV (Barnes et al. 1968).

Позже мини-компьютеры стали включать в многопроцессорные конфигурации для экономии, повышения надежности и увеличения модульности системы (Ornstein et al. 1975; Wulf and Levin 1972). Наметилась тенденция к децентрализации ради повышения надежности; слабо связанные многопроцессорные системы меньше зависят от общей центральной памяти и больше от *тонких проводов* для межпроцессного взаимодействия, что увеличивает изоляцию компонентов (Metcalfe 1972b; Roberts and Wessler 1970b). По мере продолжающегося истончения межпроцессорной связи в целях по-

вышения надежности и разработки допускающих распределение приложений многопроцессорность постепенно переходит в локальную форму распределенных вычислений.

41.1.3. Локальные компьютерные сети. Цели Ethernet во многом такие же, как у других локальных сетей: Mitrix компании Mitre, Spider компании Bell Telephone Laboratory и Distributed Computing System (DCS) компании U. C. Irvine (Farber 1973, 1975a,b; Willard 1973). Прототипы всех четырех схем локальной сети работают со скоростью от 1 до 3 мегабит в секунду. В Mitrix и Spider имеется центральный мини-компьютер для коммутации и выделения пропускной способности, тогда как в DCS и Ethernet управление распределенное. В Spider и DCS кольцевой коммуникационный тракт, в Mitrix используется готовая технология кабельного телевидения (CATV) для реализации двух односторонних шин, а в нашей экспериментальной сети Ethernet применяется разветвленная двусторонняя пассивная шина. Различия между этими системами обусловлены различиями предполагаемых приложений, разными финансовыми ограничениями, при которых принимались компромиссные решения, и различием мнений среди ученых.

Прежде чем переходить к подробному описанию Ethernet, мы предлагаем следующий обзор (см. рис. 41.2).

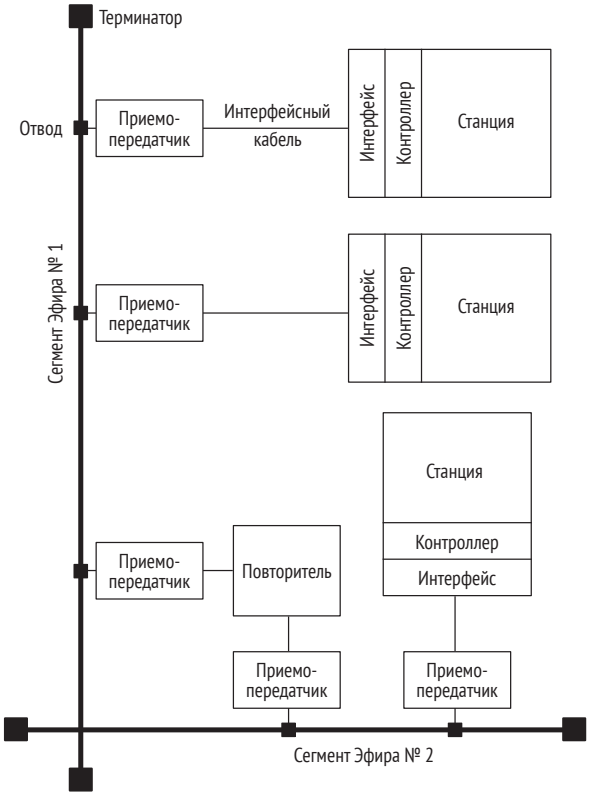


Рис. 41.2. Сеть Ethernet с двумя сегментами

41.2. Краткое описание системы

Ethernet – система для локальной связи между вычислительными станциями. В нашей экспериментальной версии Ethernet используются коаксиальные кабели с отводами для передачи цифровых пакетов данных переменной длины между персональными мини-компьютерами, печатающими устройствами, большими устройствами хранения файлов, резервными накопителями на магнитных лентах, оборудованием для передачи данных на большие расстояния и т. д.

Разделяемая коммуникационная среда, разветвленный Эфир, пассивна. Ethernet-интерфейс станции через последовательный стык по интерфейсному кабелю подключается к приемопередатчику, который отводом подключен к пассивному Эфиру. Пакет передается в Эфир, его видят все станции, но копируют из Эфира только те, которым он предназначен; это определяется по старшим битам в адресе пакета. Это широковещательная коммутация пакетов, ее следует отличать от коммутации пакетов с промежуточным накоплением и передачей, когда маршрутизация осуществляется промежуточными обрабатывающими элементами. Чтобы удовлетворить требованиям в случае роста сети, Ethernet может расширяться с помощью повторителей пакетов для регенерации сигналов, пакетных фильтров для локализации трафика и пакетных шлюзов для преобразования адресов в формат других сетей.

Управление полностью распределено между станциями с пакетной передачей и координируется посредством статистического арбитража. Передачи, инициированные станцией, откладываются до завершения уже начатых. Если после начала передачи возникла помеха со стороны других пакетов, то передача отменяется и перепланируется станцией-источником. После некоторого периода передачи в отсутствие помех пакет будет увиден всеми станциями, после чего будет передан уже до конца, не опасаясь помех. Ethernet-контроллеры в станциях, мешающих друг другу, генерируют случайные интервалы ретрансмиссии, чтобы избежать повторной коллизии пакетов. Средние интервалы ретрансмиссии пакетов корректируются с учетом истории коллизий, чтобы поддерживать близкий к оптимальному уровень использования Эфира в условиях изменяющейся нагрузки на сеть.

Даже в условиях передачи без помех, обнаруженных источником, пакет все же может не достичь места назначения без ошибки; таким образом, пакеты доставляются не гарантированно, а лишь с высокой вероятностью. Станции, для которых требуется, чтобы остаточная частота ошибок была ниже, чем обеспечивает сам механизм транспортировки пакетов в Ethernet, должны следовать взаимно согласованным протоколам передачи пакетов.

41.3. Принципы проектирования

Наша цель – спроектировать систему связи, которая может беспрепятственно расти, охватывая несколько зданий с большим числом персональных компьютеров и вспомогательных средств.

Как и соединяемые вычислительные станции, система связи должна быть недорогой. Мы решили распределить управление между взаимодействующими компьютерами, чтобы исключить проблемы надежности активного центрального контроллера, избавиться от узкого места в системе с высоким уровнем параллелизма и уменьшить постоянные издержки, из-за которых небольшая система может стать экономически невыгодной.

Проектирование Ethernet началось с основополагающей идеи о коллизии пакетов и повторной передаче, воплощенной в сети Aloha (Abramson 1970). Мы ожидали, что, как и в Aloha, в сети Ethernet может присутствовать пульсирующий трафик, поэтому традиционного синхронного мультиплексирования с разделением по времени (STDM) будет недостаточно (Abramson 1970, 1973; Metcalfe 1973; Roberts and Wessler 1970a). Мы сочли многообещающим подход Aloha к распределенному управлению мультиплексированием радиоканалов и понадеялись, что он окажется эффективным применительно к среде, подходящей для организации локальной компьютерной сети. Добавив несколько собственных изобретений, мы реализовали эту надежду.

Название Ethernet было выбрано по ассоциации со *светоносным эфиром*, в котором, как когда-то полагали, распространяется электромагнитное излучение. Как и радиопередатчик в Aloha, передатчик Ethernet помещает в Эфир пакеты, т. е. синхронизированные передатчиком последовательности битов, снабженные полным адресом, в надежде, что их увидят приемники, которым они адресованы. Логически Эфир является пассивной средой распространения цифровых сигналов, которую можно построить на основе различных носителей: коаксиальных кабелей, витых пар и оптоволоконных кабелей.

41.3.1. Топология. По финансовым причинам мы не могли позволить себе избыточных соединений и динамической маршрутизации пакетов с промежуточным хранением и передачей для обеспечения надежной связи, поэтому решили обеспечить надежность с помощью простоты. Мы решили сделать общую среду передачи пассивной, так чтобы выход из строя активного элемента влиял на связь только с одной станцией. Архитектурный план и изменяющиеся потребности офисных и лабораторных зданий побудили нас выбрать такую топологию сети, которая допускала бы постепенное расширение и изменение конфигурации с минимальным временем прерывания обслуживания.

Топологически сеть Ethernet представляет собой дерево без корня. Это *дерево*, поэтому Эфир может разветвиться при входе в коридор здания, избежав при этом помех из-за многолучевого распространения. Должен существовать только один путь через Эфир между любым источником и приемником; если бы путей существовало несколько, то передача создавала бы помехи себе же, т. е. пакет прибывал бы к приемнику, пройдя по нескольким путям разной длины. Эфир *не имеет корня*, потому что должна быть возможность расширять его из любой точки в любом направлении. Любая станция, желающая присоединиться к Ethernet, подключается к среде отводом в ближайшей удобной точке.

Глядя на соотношение взаимосвязанности и управления, мы видим, что Ethernet является двойственной к сети типа «звезда». Вместо организа-

ции *распределенной* связи с помощью отдельных звеньев и центрального блока управления в коммутирующем узле, как в сети типа «звезда», в сети Ethernet связь *центральная* через Эфир, а управление распределено между станциями.

В отличие от Aloha, которая представляет собой сеть типа «звезда» с исходящим широковещательным каналом и входящим каналом со множественным доступом, Ethernet поддерживает связь типа многие-ко-многим с помощью одного широковещательного канала с множественным доступом.

41.3.2. Управление. Управление совместным использованием Эфира осуществляется таким образом, что попытка передачи пакета двумя или более станциями примерно в одно и то же время не только возможна, но и вероятна. Если пакеты перекрываются в Эфире по времени, то говорят о *коллизии* пакетов; они создают друг другу помехи и потому не распознаются приемником. Чтобы восстановиться после обнаруженной коллизии, станция прерывает передачу и передает пакет повторно по истечении некоторого динамически выбираемого случайного времени. Арбитраж запросов на повторную передачу распределенный и статистический.

Если Эфир большую часть времени не используется, то станция передает свои пакеты, когда ей угодно, пакеты принимаются без ошибок – и все идет хорошо. Когда количество передающих станций увеличивается, частота коллизий пакетов возрастает. Контроллеры Ethernet в каждой станции устроены так, чтобы корректировать средний интервал ретрансмиссии пропорционально частоте коллизий; поэтому уровень совместного использования Эфира конкурирующими передачами от одной станции к другой поддерживается близким к оптимальному.

Для справедливого разделения Эфира необходимо определенное сотрудничество между станциями. В случае особо требовательных приложений некоторые передающие станции могут получать приоритет, систематически нарушая правила справедливого распределения. Станция может узурпировать Эфир двумя способами: не подстраивать свой интервал ретрансмиссии под увеличившийся трафик или отправлять очень большие пакеты. В настоящее время то и другое запрещено низкоуровневым программным обеспечением в каждой станции.

41.3.3. Адресация. У каждого пакета есть источник и адресат, оба они прописаны в заголовке пакета. Пакет, помещенный в Эфир, распространяется всем станциям. Любая станция может скопировать пакет из Эфира в свою локальную память, но обычно так поступает только активная станция-адресат, которая видит, что адрес в заголовке проходящего мимо пакета совпадает с ее собственным. По соглашению, адрес, состоящий из одних нулей, является универсальным, т. е. совпадает с любым адресом; пакет с универсальным адресом получателя называется *широковещательным*.

41.3.4. Надежность. Сеть Ethernet по природе своей вероятностная. Пакеты могут теряться из-за помех со стороны других пакетов, импульсных помех в Эфире, неактивности предполагаемого получателя или сознательного отбрасывания. Протоколы, применяемые для связи через Ethernet, должны

предполагать, что пакеты будут правильно доставлены в место назначения лишь *с высокой вероятностью*.

Ethernet старается сделать *все возможное* для успешной передачи пакетов, но окончательная ответственность за желаемую степень надежности связи возлагается на станции источника и адресата (Metcalfe 1972b, 1973). Понимая дороговизну и опасности обещания «безошибочной» связи, мы воздерживаемся от гарантий надежной доставки каждого отдельного пакета ради экономии на стоимости передачи и достижения высокой надежности в среднем по многим пакетам (Metcalfe 1973). Снятие ответственности за надежность связи с механизма транспортировки пакетов позволяет нам настраивать надежность в соответствии с потребностями приложения и помещать процедуру восстановления после ошибок туда, где от нее будет больше пользы. Эта политика становится еще важнее, если сети Ethernet соединяются в иерархию сетей, по которой пакеты должны передаваться дальше, подвергаясь большему риску.

41.3.5. Механизмы. Станция подключается к Эфиру с помощью отвода и приемопередатчика. Отвод – это устройство для физического подключения к Эфиру с минимальным нарушением его передающих характеристик. Проектирование приемопередатчика по необходимости является упражнением в паранойе. Необходимо принять меры к тому, чтобы вероятные отказы приемопередатчика или станции не приводили к засорению Эфира. В частности, перебой в питании приемопередатчика не должен приводить к его отключению от Эфира.

В нашей экспериментальной сети Ethernet предложено пять механизмов уменьшения вероятности и стоимости потери пакета: (1) обнаружение несущей, (2) обнаружение помехи, (3) обнаружение ошибки в пакете, (4) фильтрация усеченных пакетов и (5) принудительный консенсус при коллизиях.

41.3.5.1. Обнаружение несущей. Когда станция помещает биты пакета в Эфир, они подвергаются фазовому кодированию (как биты на магнитной ленте), которое гарантирует, что во временном интервале каждого бита имеется по крайней мере одно изменение состояния Эфира. Поэтому передача пакета в Эфир может быть обнаружена путем прослушивания связанных с ним изменений состояния. По аналогии с радиосвязью мы говорим о присутствии несущей частоты при прохождении пакета через приемопередатчик. Поскольку станция может обнаружить несущую проходящего пакета, она может и отложить отправку собственного пакета, пока обнаруженный пакет не пройдет дальше. В сети Aloha механизма обнаружения несущей нет, поэтому частота коллизий в ней заметно выше. Без обнаружения несущей эффективность использования Эфира уменьшалась бы с увеличением длины пакета. В § 41.6 ниже будет показано, что благодаря обнаружению несущей эффективность использования Эфира возрастает при увеличении длины пакета.

Обнаружение несущей позволяет реализовать задержку: никакая станция не начнет передачу, пока видит несущую. Задержка влечет за собой захват Эфира: на протяжении всего времени распространения пакета через Эфир все станции видят несущую и задерживают передачу, т. е. Эфир захвачен

и передача завершится без коллизий. Коллизия может случиться, только если две или более станций слышат молчание в Эфире и начинают передачу одновременно, точнее на протяжении времени распространения пакета от начала до конца Эфира. Это почти всегда происходит после передачи пакета, вызвавшей задержку у двух или более станций. Поскольку станции в настоящее время не рандомизируют время начала передачи после задержки, то когда передача пакета завершается, ожидающие станции начинают передачу одновременно, пакеты накладываются, происходит рандомизация и повторная передача.

41.3.5.2. Обнаружение помехи. В каждом приемопередатчике имеется детектор помех. Признаком наличия помехи является различие между значением бита, полученного приемопередатчиком из Эфира, и значением бита, которое он пытается передать.

Обнаружение помехи дает три преимущества. Во-первых, станция, обнаружившая коллизию, знает, что ее пакет поврежден. Пакет можно запланировать для повторной передачи немедленно, избежав длительного тайм-аута подтверждения. Во-вторых, период помехи в Эфире ограничен максимум одним периодом кругового обращения. В сети Aloha наложившиеся пакеты передаются до конца, но в Ethernet усеченные в результате коллизии пакеты проводят в Эфире (впустую расходуя ресурсы) лишь малую долю времени передачи полного пакета. В-третьих, частота обнаружения помех используется для оценки трафика в Эфире и, следовательно, корректировки интервалов ретрансмиссии и оптимизации эффективности использования канала.

41.3.5.3. Обнаружение ошибки в пакете. При помещении пакета в Эфир вычисляется и включается в пакет контрольная сумма. При чтении пакета из Эфира контрольная сумма пересчитывается. Если обе контрольные суммы не совпадают, то пакет отбрасывается. Таким образом, ошибки передачи, ошибки из-за импульсных помех и ошибки из-за необнаруженных помех перехватываются в точке назначения пакета.

41.3.5.4. Фильтрация усеченных пакетов. Из-за обнаружения помехи и задержки большинство коллизий приводят к появлению *усеченных пакетов*, содержащих всего несколько битов; станции, обнаружившие помеху, прекращают передачу еще до истечения времени кругового обращения в Эфире. Чтобы уменьшить нагрузку на программное обеспечение прослушивающей станции и избавить ее от обработки таких, очевидно, поврежденных пакетов, усеченные пакеты отфильтровываются аппаратно.

41.3.5.5. Принудительный консенсус при коллизиях. Когда станция решает, что на ее передачу наложилась помеха, она посылает в Эфир специальный сигнал затора, чтобы все остальные участники коллизии обнаружили помеху и, следуя протоколу задержки, были вынуждены прервать передачу. Без такого механизма *принудительного консенсуса при коллизиях* могло бы случиться, что передающая станция, которая должна была бы последней обнаружить коллизию, не сумела бы этого сделать, потому что другие создающие помеху передачи одна за другой прерываются и прекращают вносить помехи. Хотя этому последнему передатчику пакет может показаться нормальным,

различие в длинах путей между наложившимися передатчиками и предполагаемым приемником приведет к тому, что пакет придет поврежденным.

41.4. Реализация

Наш выбор параметров экспериментальной сети Ethernet – протяженность 1 км, скорость передачи 3 Мб/с, 256 станций – был продиктован характеристиками локально распределенной коммуникационной среды и нашими оценками минимально достижимого; безусловно, это не жесткие ограничения, присущие самой концепции Ethernet.

Мы ожидали, что разумный максимальный размер сети будет порядка 1 км кабеля. И использовали эту прикидку при выборе Эфиров с разным затуханием сигнала и при проектировании приемопередатчиков подходящей мощности и чувствительности.

Основными станциями в нашей экспериментальной Ethernet являются мини-компьютеры, для которых 3 Мб/с – удобная скорость передачи данных. Сохраняя пиковую скорость передачи намного меньшей, чем скорость доступа компьютера к оперативной памяти, мы уменьшаем необходимость в дорогостоящей специализированной буферизации пакетов в наших интерфейсах к Ethernet. Поддерживая пиковую скорость на удобном уровне, мы открываем возможность для увеличения количества станций и создания более развитых многопроцессорных приложений связи.

Чтобы ускорить низкоуровневую обработку пакетов на 256 станциях, мы отводим первый 8-битовый байт пакета под адрес получателя, а второй байт – под адрес отправителя (см. рис. 41.3). Число 256 достаточно мало, чтобы каждая станция получала адекватную долю имеющейся пропускной способности, и близко к пределу, достижимому с помощью современной техники врезки в кабели. Число 256 удобно только для самых нижних уровней протокола; на верхних уровнях можно расширить адресное пространство, включив в пакет дополнительные поля, интерпретируемые программным обеспечением.

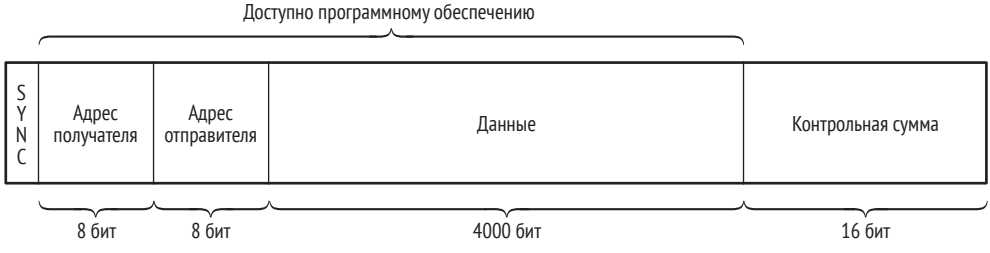


Рис. 41.3. Формат пакета Ethernet

Наша экспериментальная реализация Ethernet состоит из четырех основных частей: Эфир, приемопередатчики, интерфейсы и контроллеры (см. рис. 41.2).

41.4.1. Эфир. Мы решили реализовать экспериментальный Эфир с помощью коаксиального кабеля с малыми потерями и применяемых в кабельном телевидении отводов и соединителей. В одной сети Ethernet можно применять разные Эфиры; мы используем коаксиальный кабель малого диаметра для соединений внутри кластера станций и кабели большого диаметра между кластерами для уменьшения потерь. Стоимость Эфира на коаксиальном кабеле незначительна по сравнению со стоимостью распределенных вычислительных систем, поддерживаемых Ethernet.

41.4.2. Приемопередатчики. Наши экспериментальные приемопередатчики могут обслуживать один километр коаксиального кабеля, в который врезано 256 станций, передающих со скоростью 3 Мб/с. Приемопередатчики не выходят из строя после длительного короткого замыкания, неправильного окончания Эфира и одновременной передачи всеми 256 станциями; они выдерживают (т. е. работают в условиях) дифференциал заземления и бытовые электрические помехи, например от пишущих машинок или электрических дрелей, когда расстояние между станциями достигает километра.

Приемопередатчик Ethernet подключается непосредственно к Эфиру, который проходит по потолку или под полом. Он питается и управляется по 5 витым парам в интерфейсном кабеле, предназначенном для передачи данных, приема данных, обнаружения помех и напряжения источника питания. В отсутствие подачи энергии приемопередатчик электрически отключается от эфира. В этом проявляется наша борьба за надежность; неисправный приемопередатчик может, но не должен, вывести из строя всю сеть Ethernet. Цепь сторожевого таймера в каждом приемопередатчике пытается предотвратить засорение Эфира, выключая выходной каскад, если он ведет себя подозрительно. Чтобы упростить приемопередатчик, мы используем базовую полосу частот Эфира, но Ethernet можно было построить с использованием полосы любого подходящего размера Эфира с частотным разделением каналов. <…>

41.4.3. Интерфейс. Интерфейс с Ethernet сериализует и десериализует параллельные данные, используемые станциями. В нашей Ethernet имеются разные станции, и для каждой нужен свой интерфейс.

В передающем интерфейсе используется адрес пакетного буфера и счетчик слов, чтобы сериализовать и подвергнуть фазовому кодированию переменное число 16-битовых слов, выбираемых из памяти станции и передаваемых приемопередатчику, предпослав им стартовый бит (на рис. 41.3 он назван SYNC) и добавив в конце контрольную сумму. Принимающий интерфейс использует появление несущей, чтобы обнаружить начало пакета, и бит SYNC, чтобы захватить фазу бита. Пока несущая присутствует, интерфейс декодирует и десериализует входной поток битов, помещая 16-битовые слова в пакетный буфер в оперативной памяти станции. Когда несущая пропадает, интерфейс проверяет, что было получено целое число 16-битовых слов и что контрольная сумма правильна. Предполагается, что последнее принятое слово – контрольная сумма, оно не копируется в пакетный буфер.

Обычно эти интерфейсы включают оборудование для приема только пакетов с подходящими адресами в заголовках. Аппаратная фильтрация адресов помогает станции избегать обременительной программной обработки паке-

тов, когда Эфир сильно загружен трафиком, адресованным другим станциям.

41.4.4. Контроллер. Контроллер Ethernet – это зависящая от станции низкоуровневая прошивка или программа для помещения пакетов в Эфир из выборки их из Эфира. При обнаружении источником коллизии на контроллер источника возлагается обязанность сгенерировать новый случайный интервал ретрансмиссии на основе обновленного счетчика коллизий. Мы изучили несколько алгоритмов управления частотой ретрансмиссии на станциях, стремясь обеспечить эффективное использование Эфира (Metcalfe 1972a). Самые практичные из них оценивают интенсивность нагрузки, опираясь на историю недавних коллизий.

Интервалы ретрансмиссии кратны слоту, максимальному времени между началом передачи и обнаружением коллизии, т. е. времени одного кругового обращения. Контроллер Ethernet начинает передачу каждого нового пакета, принимая средний интервал ретрансмиссии, равный одному слоту. Всякий раз, как попытка передачи заканчивается коллизией, контроллер выполняет задержку на интервал случайной продолжительности со средним, в два раза большим, чем для предыдущего интервала, задерживает все проходящие мимо пакеты, а затем пытается передать пакет повторно. Эта эвристика аппроксимирует алгоритм, который мы назвали двоичной экспоненциальной выдержкой (см. рис. 41.4).

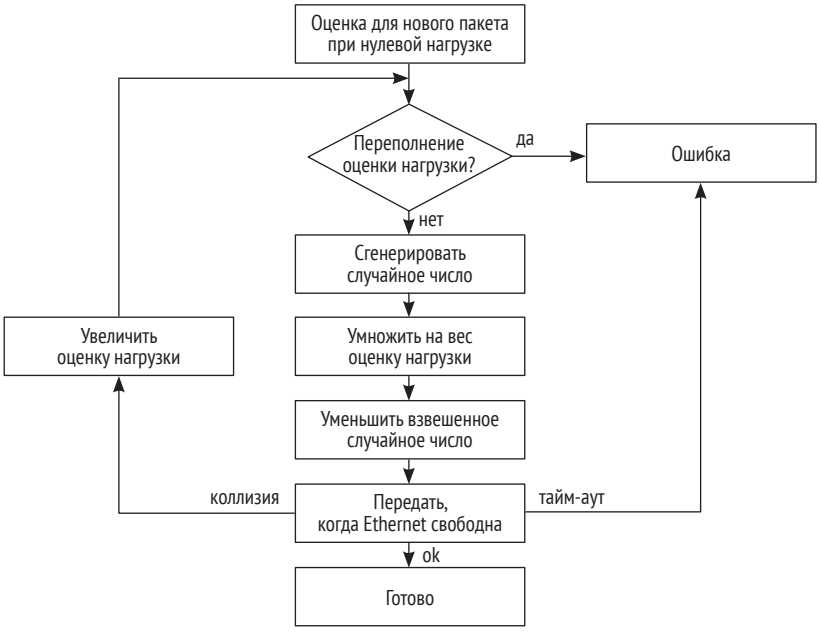


Рис. 41.4. Алгоритм управления коллизиями

Когда сеть не загружена и коллизии случаются нечасто, среднее редко отличается от единицы, и повторная передача производится быстро. По мере нарастания трафика число коллизий увеличивается, на станциях накаплива-

ется очередь задержанных пакетов, интервалы ретрансмиссии растут и повторный трафик снижается для поддержания эффективности канала.

41.5. Рост

41.5.1. Зона покрытия сигналом. Расширять Ethernet путем добавления приемопередатчиков и продления Эфира можно не до бесконечности. Наступает момент, когда приемопередатчики и Эфир уже не способны переносить требуемые сигналы. Зону покрытия сигналом можно расширить с помощью простого небуферизованного *повторителя пакетов*. В нашей экспериментальной сети Ethernet, где по причине простоты приемопередатчика невозможно пассивное разветвление Эфира, простые повторители могут соединять любое число *сегментов* Эфира, разнообразя топологию и расширяя зону покрытия.

Мы эксплуатируем экспериментальный двухсегментный повторитель пакетов, но надеемся избежать зависимости от них. В части разветвления Эфира и расширения его зоны покрытия существует компромисс между использованием более сложных приемопередатчиков и применением повторителей. С увеличением мощности и чувствительности приемопередатчики становятся более дорогими и менее надежными. Включение повторителей в Ethernet делает центральный связующий Эфир активным. Выход из строя приемопередатчика приведет к прерыванию связи только у его владельца; выход из строя повторителя приводит к распадению Эфира на части, лишая связи сразу многих.

41.5.2. Транспортное покрытие. Продление Эфира и добавление повторителей пакетов способны расширить Ethernet лишь до определенного предела. Наступает момент, когда Эфир становится настолько загруженным, что дополнительные станции будут просто делить на более мелкие части полосу пропускания, которой уже и так не хватает. *Транспортное покрытие* можно расширить с помощью небуферизованного повторителя, или *пакетного фильтра*, который передает пакеты из одного сегмента Эфира в другой, только если конечная станция находится в новом сегменте. Пакетный фильтр также расширяет зону покрытия сигналом.

41.5.3. Адресное покрытие. Продление Эфира, добавление повторителей пакетов и пакетных фильтров способно расширить Ethernet лишь до определенного предела. Наступает момент, когда количество станций становится настолько большим, что для их адресации уже не хватает 8-битовых адресов. *Адресное покрытие* можно расширить с помощью пакетных шлюзов и реализуемых ими программных соглашений об адресации (Cerf and Kahn 1974, здесь глава 38). Адреса можно расширять в двух направлениях: вниз, добавляя поля для идентификации конечных портов или процессов внутри станции, и вверх, добавляя поля для идентификации конечных станций в удаленных сетях. Шлюз также расширяет транспортное покрытие и зону покрытия сигналом.

Между двумя сегментами Эфира может находиться только один повторитель или пакетный фильтр; пакет, переданный в сегмент несколькими повторителями, будет создавать помеху самому себе. Однако на количество шлюзов, соединяющих два сегмента, ограничений не накладывается; шлюз лишь повторяет пакеты, адресованные ему самому как посреднику. Выход из строя одного повторителя, соединяющего два сегмента, разделяет сеть на части; выход из строя шлюза не приводит к разделению сети, если существуют другие пути между сегментами.

41.6. Производительность

<…> Мы разработали простую модель производительности нагруженной сети Ethernet, изучив чередующиеся временные периоды работы Эфира. Первый, называемый интервалом передачи, – это время, в течение которого Эфир был захвачен для успешной передачи пакета. Второй, называемый интервалом конкуренции, состоит из слотов ретрансмиссии, описанных в § 41.4.4, в течение которых станции пытаются захватить контроль над Эфиром. Поскольку в модели сеть Ethernet нагружена и поскольку станции задерживают проходящие пакеты перед началом передачи, слоты синхронизируются по концу предшествующего интервала захвата. Слот будет пустым, если ни одна станция не стала ничего передавать в нем, и будет содержать коллизию, если решение о передаче приняло сразу несколько станций. Если в слоте была предпринята только одна попытка передачи, то Эфир захватывается на все время прохождения пакета, интервал конкуренции заканчивается и начинается интервал передачи.

Пусть P – число битов в Ethernet-пакете, C – пиковая пропускная способность в битах в секунду, переносимых Эфиром, а T – продолжительность слота в секундах, т. е. сколько секунд занимает обнаружение коллизии после начала передачи. Предположим, что Q станций постоянно ждут своей очереди передать пакет; либо захватившая Эфир станция имеет новый пакет сразу после успешного захвата, либо уже готова другая станция. Заметим, что Q также дает полную расчетную нагрузку на сеть, которая для этого анализа всегда больше или равна 1. Мы предполагаем, что стоящая в очереди станция пытается передавать в текущем слоте с вероятностью $1/Q$ или задерживает пакет с вероятностью $1 – (1/Q)$; известно, что это оптимальное статистическое правило принятия решения, аппроксимированное в станциях Ethernet с помощью наших алгоритмов управления повторной передачи, основанных на оценивании нагрузки.

41.6.1. Вероятность захвата. Теперь вычислим A – вероятность, что ровно одна станция пытается передавать в слоте и, следовательно, захватывает Эфир. A равна $Q \cdot (1/Q) \cdot ((1 – (1/Q))^{Q–1})$: существует Q способов выбрать станцию, которая решит передавать (с вероятностью $(1/Q)$), тогда как $Q – 1$ решает ждать (с вероятностью $1 – (1/Q)$). После упрощения получаем $A = (1 – (1/Q))^{Q–1}$.

41.6.2. Время ожидания. Теперь вычислим W – среднее число слотов ожидания в интервале конкуренции, прежде чем Эфир будет захвачен передачей какой-то станции. Вероятность, что вообще не придется ждать, равна просто A – вероятности, что одна и только одна станция решит передавать в первом слоте, следующем за передачей. Вероятность ожидания в течение одного слота равна $A(1 - A)$; вероятность ожидания в течение i слотов равна $A(1 - A)^i$. Среднее этого геометрического распределения равно $W = (1 - A)/A$.

41.6.3. Эффективность. Теперь вычислим E – долю времени, в течение которого Эфир переносит хорошие пакеты, т. е. эффективность. Время работы Эфира разделяется на интервалы передачи и интервалы конкуренции. Передача пакета занимает P/C секунд. Среднее время до захвата равно WT. Следовательно, в нашей простой модели $E = (P/C)/((P/C) + (WT))$. <…>

Для пакетов размером свыше 4000 бит эффективность нашей экспериментальной сети Ethernet заметно превышает 95 %. Для пакетов, размер которых приближенно равен размеру слота, эффективность Ethernet стремится к $1/e$, асимптотической эффективности сети Aloha со слотами (Roberts, 1973).

41.7. Протокол

Построение работоспособной системы пакетной связи не исчерпывается предоставлением механизмов транспортировки пакетов. Должны быть также предусмотрены методы исправления ошибок, управления потоком, именования процессов, обеспечения безопасности и учета. Для этого предназначены протоколы верхнего уровня, надстроенные над протоколом управления Эфиром, описанным в § 41.3 и 41.4 выше (Cerf and Kahn 1974; Crocker et al. 1972; Farber 1973; Metcalfe 1973; Rowe 1975; Walden 1972). Управление эфиром включает кадрирование пакетов, обнаружение ошибок, адресацию и контроль множественного доступа. Как и другие процедуры управления линией, Ethernet используется для поддержки многочисленных архитектур сети и многопроцессорности (IBM 1974, 1975).

Ниже приведено краткое описание одного простого пакетного протокола с контролем ошибок. Протокол EFTP (Ethernet File Transfer Protocol) представляет интерес как потому, что его легко понять и правильно реализовать, так и потому, что с его помощью было перенесено много ценных файлов во время разработки более общих и эффективных протоколов.

41.7.1. Общая терминология. При обсуждении пакетных протоколов мы пользуемся следующей терминологией, полезной и в общем случае. Говорят, что у пакета есть *источник* и *адресат*. Говорят, что у потока данных есть *отправитель* и *получатель*, отдавая должное тому факту, что для поддержки потока данных источником некоторых пакетов (как правило, подтверждений) будет получатель, а адресованы они будут отправителю. Говорят, что у соединения есть *прослушиватель* и *инициатор*, а у службы – *сервер* и *пользователь*. Очень полезно рассматривать эти понятия как ортогональные опи-

сания участников связи. Конечно, сервер обычно является прослушивателем, а источником пакетов данных обычно является отправитель.

41.7.2. EFTP. Первые 16 бит всех Ethernet-пакетов содержат интерпретируемые интерфейсом однобайтовые адреса станций источника и адресата (см. рис. 41.3). По принятому в программах соглашению, следующие 16 бит всех Ethernet-пакетов содержат тип пакета. В различных протоколах используются непересекающиеся наборы типов пакетов. В EFTP таких типов пять: data (данные), ack (подтверждение), abort (отмена), end (конец) и endreply (ответ на end). Следующие 16-битовые слова EFTP-пакета: порядковый номер, длина, факультативные слова данных и, наконец, программно вычисляемая контрольная сумма (см. рис. 41.5). Аппаратно вычисляемая контрольная сумма Ethernet видна только Эфиру и на этом уровне протокола не учитывается.

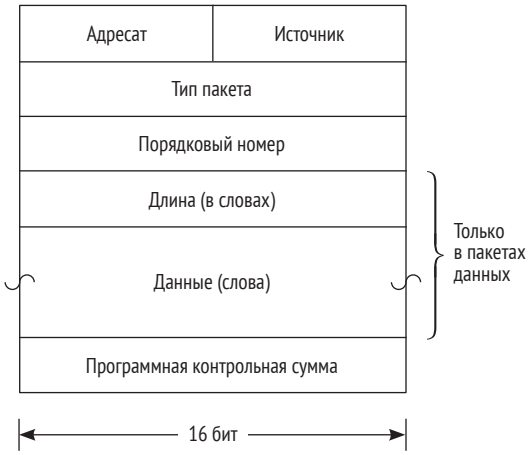


Рис. 41.5. Формат EFTP-пакета

Должно быть понятно, что не уделялось особого внимания наиболее экономному выбору длин некоторых полей. Упор здесь был сделан на простоте и легкости программирования. Но даже с этой оговоркой мы все равно считаем, что лучше отдавать предпочтение более просторным полям; как ни крути, какое-то поле обязательно окажется слишком узким.

Слово программной контрольной суммы используется, чтобы уменьшить вероятность необнаруженной ошибки. Оно не только выступает в роли резервного механизма для вычисляемой последовательным оборудованием экспериментальной Ethernet 16-битовой контрольной суммы циклического избыточного кода (рис. 41.3), но также защищает от ошибок на параллельных путях данных внутри контрольных станций, которые не контролируются с помощью CRC. В качестве контрольной суммы в EFTP используется дополнительный код суммы всех битов пакета, включая заголовок и данные. На свой страх и риск пользователь может игнорировать эту контрольную сумму на любой стороне: отправитель вправе поместить все единицы (невозможное значение) в слово контрольной суммы, показав тем самым, что она не вычислялась.

41.7.2.1. Передача данных. 16-битовые слова файла пересылаются со станции-отправителя на станцию-получатель в пакетах данных, нумеруемых последовательно, начиная с 0. Каждый пакет данных повторно передается отправителем через равные промежутки времени, пока от получателя не придет пакет подтверждения, содержащий такой же порядковый номер. Получатель игнорирует все поврежденные пакеты, пакеты, пришедшие от любой станции, кроме отправителя, и пакеты с порядковым номером, не равным ожидаемому или на единицу меньшим. Если пакет имеет ожидаемый порядковый номер, то он подтверждается, данные принимаются как часть файла и порядковый номер увеличивается на 1. Если порядковый номер пакета на единицу меньше ожидаемого, то он подтверждается и отбрасывается; в этом случае предполагается, что подтверждение было потеряно и пакет передан повторно.

41.7.2.2. Завершение. После передачи всех данных посылается пакет end, содержащий следующий по порядку номер, и отправитель ждет соответственного пакета endreply. Получив пакет end, получатель посылает в ответ пакет endreply с таким же порядковым номером, а затем просто ждет, *ничего не делая*, в течение достаточно длительного времени (10 секунд). Получив endreply, станция-отправитель передает копию endreply и вольна закончить работу в уверенности, что файл был передан успешно. Получив копию своего endreply, ожидающий получатель также может завершиться.

Сравнительно сложная последовательность end–ожидание нужна для того, чтобы отправитель и получатель пришли к соглашению об успешности передачи файла. Если пакет end потерян, то отправитель просто передаст его повторно, как любой пакет с не пришедшим вовремя подтверждением. Если потерян пакет endreply от получателя, то отправитель по истечении таймаута также передаст его повторно, а ожидающий получатель подтвердит его. Если потеряна копия endreply, то ожидающий получатель, конечно, испытает неудобства, поскольку пришлось ждать лишнее время, но по истечении таймаута все равно будет уверен в успешной передаче файла, потому что пакет end был получен.

В любой точке этой процедуры любая сторона вправе решить, что связь нарушена, и отказаться от продолжения; считается проявлением вежливости, если она отправит другой стороне пакет abort, чтобы завершить взаимодействие, например в случае, когда причиной является отмена по инициативе пользователя или из-за ошибки в файловой системе.

41.7.2.3. Недостатки EFTP. Протокол EFTP был очень полезен, но обладает многочисленными недостатками. Во-первых, протокол предусматривает только передачу файла от одной станции другой в пределах одной сети, а не между процессами, работающими на станциях в одной сети или в разных сетях, связанных шлюзом. Во-вторых, механизм рандеву процессов практически отсутствует, поскольку невозможно найти процессы по имени и нет удобного способа обработки нескольких пользователей одним сервером. В-третьих, нет настоящего управления потоком. Если данные приходят получателю, который не может поместить их в буферы, то они могут быть просто отброшены в уверенности, что в конечном итоге будут переданы повторно.

У получателя нет никакого способа притормозить поток таких расточительных передач или ускорить повторную передачу. В-четвертых, данные передаются в виде целого числа 16-битовых слов, принадлежащих безымянным файлам, и потому EFTP либо оказывается очень ограничительным, либо нуждается в каких-то форматах передачи файлов, организованных внутри его слов данных. И в-пятых, утрачена функциональная общность, потому что получатель является также прослушивателем и сервером.

41.8. Заключение

Наш опыт эксплуатации Ethernet позволил сделать вывод, что предпочтение, отданное нами распределенному управлению, вполне оправдано. Сведя количество разделяемых компонентов системы связи к минимуму и сделав их пассивными, мы смогли достичь очень высокого уровня надежности. Установка и сопровождение экспериментальной Ethernet оказались более чем удовлетворительными. Гибкость соединения станций, обеспечиваемая широковещательной коммутацией пакетов, подвигла на разработку многочисленных приложений для компьютерных сетей и многопроцессорной обработки. <...>

43 Новые направления в криптографии (1976)

Уитфилд Диффи и Мартин Хеллман

Многие важные работы не казались таковыми сразу после первой публикации. Даже Алан Тьюринг чувствовал себя непонятым в течение нескольких месяцев после публикации решения проблемы разрешимости Гильберта. В феврале 1937 года он писал матери, что великие игнорируют его и что он получил всего два запроса на оттиски, причем один из них – от коллеги по Кембриджу, который уже знал доказательство (Hodges 1983, стр. 124). И еще меньше статей, в которых так дерзко, как в «Новых направлениях криптографии», заявлено, что мы находимся на «грани революции».

Проблемы, обсуждаемые Уитфилдом Диффи (родился в 1944 году) и Мартином Хеллманом (родился в 1945 году), формулируются просто. Первая из них – распределение ключей. С незапамятных времен люди отправляют друг другу закодированные сообщения. Одна сторона кодирует сообщение с помощью ключа шифрования, а другая сторона, получив сообщение, расшифровывает его с помощью того же ключа. Любая сторона, перехватившая сообщение в процессе передачи, не может расшифровать его без ключа. Проблема в том, что безопасно передать ключ от одной стороны другой принципиально не легче, чем безопасно отправить само незакодированное сообщение. Ключ может быть короче, поэтому его проще спрятать, или же обе стороны могли предварительно встретиться и согласовать ключ, а потом разойтись по своим делам. Но из фундаментальной проблемы разделения ключей шифрования вытекал печальный факт: безопасное шифрование являлось прерогативой наделенных властью и богатством – лишь стороны, обладающие средствами для защиты ключей, могли надежно обмениваться секретными сообщениями.

Вторая проблема – цифровые подписи: как распространить не допускающее подделки сообщение или документ, так чтобы получатель мог быть уверен, что никто не пытается выдать себя за отправителя.

Диффи и Хеллман решают проблему распределения ключей, предложив идею наделить стороны возможностью дистанционно согласовывать ключ по небезопасному каналу связи, не передавая сам ключ – вещь, которую даже Шеннон в своей унифицированной трактовке теории секретности, похоже, отвергал как невозможную, не дав себе труда рассмотреть ее более тщательно: «Ключ должен передаваться с помощью неперехватываемых средств от передающей точки к принимающей» (Shannon 1949, стр. 670). Проблема цифровой подписи оказалась связанной с первой, хотя в данной статье конкретные предложения касательно цифровых подписей математически отличаются от предложений по распределению ключей. Диффи и Хеллман надеялись на появление общего решения обеих проблем, но с этим пришлось подождать (см. главу 45).

Какими бы ошеломляющими ни были «Новые направления» – а в академических кругах очень скоро осознали, что эта статья поднимает ряд важных вопросов, – они имели связи с рядом предшествующих работ, не все из которых были известны авторам в то время. Диффи (Diffie 1988) признает заслуги Ральфа Меркла в выдвижении идеи открытого ключа еще в бытность последнего студентом в Беркли в 1974 году – только он не сумел никого убедить прислушаться к нему. А некоторые идеи криптографии с открытым ключом были открыты в 1970-х годах учеными из GCHQ, Британского разведывательного агентства, но эти работы долго оставались под грифом секретности и не были известны ни в академических, ни в промышленных кругах.

Но детали имеют значение, а Диффи и Хеллман предложили не только концептуальную основу, но и детали, включая применение возведения в степень по модулю в качестве основы своей функции шифрования. Если проблему дискретного логарифма – операции, обратной возведению в степень по модулю, – легко решить, то протокол Диффи–Хеллмана можно взломать. Поэтому начались поиски быстрого алгоритма дискретного логарифмирования. Внезапно исследования по криптографии вышли на передний план, а теория чисел, по поводу которой знаменитый математик Г. Х. Харди гордо заявлял, что она никогда не принесет никому практической пользы, стала самой востребованной отраслью математики. (До сих пор никому не удалось найти быстрого алгоритма дискретного логарифмирования, но в 1980-х годах Ади Шамир [Adi Shamir 1984] обнаружил, что описанное в данной статье предложение использовать задачу об укладке рюкзака в качестве основы для алгоритма односторонней цифровой подписи имеет фатальный дефект.)

Государственные и коммерческие учреждения приняли важность криптографии с открытым ключом далеко не так быстро, как академические круги. На самом деле значение этой статьи для нового общественно-политического устройства мира ничуть не меньше, чем ее техническая ценность. Криптография веками была интеллектуальным омутом, потому что широко признавалось, что все известное в этой области хранится под спудом секретности государственными агентствами. Статья Диффи и Хеллмана ознаменовала начало обмена идеями о криптографии.

И она же положила начало передаче инструментов шифрования в руки обычных граждан – важнейшее событие, коль скоро интернет предполага-

лось использовать для торговых и банковских операций, – и тем самым подорвала попытки правительств контролировать технологии путем установления экспортного контроля над программным обеспечением шифрования и требовать организации закладок в системах связи и сотовых телефонах. Не будь шифрования с открытым ключом, мы сегодня не спорили бы до хрипоты о балансе между свободой общения для обычных граждан и необходимостью защитить граждан от влияния враждебных акторов силами государства.

42.1. Введение

Сегодня мы находимся на грани революции. Разработка дешевого цифрового оборудования освободила нас от ограничений механических вычислительных устройств и уменьшила стоимость высококачественных криптографических устройств до уровня, когда они могут использоваться в таких коммерческих приложениях, как удаленные автоматы по выдаче наличных и компьютерные терминалы. В свою очередь, такие приложения создают потребность в новых типах криптографических устройств, которые минимизируют необходимость в безопасных каналах распределения ключей и предлагают эквивалент рукописной подписи. В то же время теоретические достижения в области теории информации и вычислительной техники обещают создание доказуемо безопасных криптосистем, которые превратят это древнее искусство в науку.

Развитие управляемых компьютерами сетей связи предлагает недорогие и не требующие больших усилий контакты между людьми или компьютерами на разных сторонах земного шара, способные заменить большую часть почты и многие личные перемещения телекоммуникациями. Для многих приложений эти контакты можно сделать защищенными как от подслушивания, так и от внедрения беззаконных сообщений. Однако в настоящее время решение проблем безопасности сильно отстает от прогресса в других технологиях связи. Современная криптография не в силах удовлетворить требованиям, поскольку ее использование привело бы к таким серьезным неудобствам для пользователей системы, что многие преимущества удаленной обработки были бы сведены на нет.

Самая известная криптографическая проблема – конфиденциальность: предотвращение несанкционированного извлечения информации из сообщений, передаваемых по небезопасным каналам. Однако чтобы использовать криптографию для обеспечения конфиденциальности, в настоящее время сообщающиеся стороны должны иметь общий ключ, неизвестный больше никому. Для этого ключ необходимо заранее передать по безопасному каналу, например курьером или заказным почтовым отправлением. Однако конфиденциальный разговор между двумя ранее незнакомыми людьми – типичная ситуация в бизнесе, и было бы нереалистично ожидать, что первый деловой контакт будет отложен на время, достаточное для передачи ключей с помощью каких-то физических средств. Затраты и задержки,

вызванные этой проблемой распределения ключей, – основное препятствие на пути перевода деловых коммуникаций на крупные сети телеобработки.

В § 42.3 предлагается два подхода к передаче ключевой информации по открытым (т. е. небезопасным) каналам, не ставящие под сомнение безопасность системы. В *криптосистеме с открытым ключом* шифрование и дешифрирование производятся различными ключами, E и D, такими, что вычисление D по E технически не осуществимо (например, требует 10^{100} инструкций). Таким образом, ключ шифрования E можно раскрыть, не компрометируя ключ дешифрирования D. Поэтому любой пользователь сети может поместить свой ключ шифрования в открытый каталог. Это позволяет любому пользователю системы отправить сообщение любому другому пользователю, зашифровав его таким образом, что расшифровать сможет только предполагаемый получатель. В этом смысле криптосистема с открытым ключом является шифром с коллективным доступом. Поэтому конфиденциальный разговор между двумя индивидуумами становится возможен вне зависимости от того, общались они прежде или нет. Каждая сторона отправляет другой стороне сообщения, зашифрованные открытым ключом шифрования получателя, и расшифровывает полученные сообщения своим собственным секретным ключом дешифрирования.

Мы предлагаем некоторые методы разработки криптосистем с открытым ключом, но проблема все еще остается в значительной степени нерешенной.

Системы распределения открытых ключей предлагают другой подход к устранению необходимости в безопасном канале распределения ключей. В такой системе два пользователя, желающих обменяться ключом, выполняют повторные раунды коммуникации, пока не выработают общий ключ. Для третьей стороны, подслушивающей этот обмен, определение ключа на основе перехваченной информации должно стать вычислительно неразрешимой задачей. Возможное решение проблемы распределения открытых ключей приведено в § 42.3, а в работе (Merkle 1978) предложено частичное решение другого вида.

Вторая проблема, нуждающаяся в криптографическом решении и в настоящее время мешающая бизнесу перейти к системам телеобработки, – аутентификация. В современном бизнесе достоверность контрактов гарантируется подписями. Подписанный контракт является юридически значимым свидетельством соглашения, которое держатель может предъявить суду в случае необходимости. Однако для использования подписей необходимо передавать и хранить подписанные контракты. Чтобы заменить это бумажное средство чисто цифровым, каждый пользователь должен иметь возможность создать сообщение, подлинность которого может проверить любой, но воспроизвести не может никто, даже получатель. Поскольку только один человек может создавать подписанные сообщения, но получать их могут многие, этот шифр можно рассматривать как широковещательный. Современные электронные средства аутентификации не отвечают этой потребности.

В § 42.4 обсуждается проблема построения настоящей цифровой подписи, зависящей от сообщения. По причинам, которые будут изложены там же, мы называем это проблемой односторонней аутентификации. Приведены неко-

торые частичные решения и показано, как любую криптосистему с открытым ключом можно преобразовать в систему односторонней аутентификации.

В §42.5 будет рассмотрена взаимосвязь различных криптографических проблем и рассказано о еще более трудной проблеме закладок.

В то самое время, когда появление средств связи и вычислений породило новые криптографические проблемы, их отпрыски, теория информации и теория вычислимости, начали поставлять инструменты для решения важных проблем классической криптографии.

Поиск невскрываемых кодов – один из самых старых вопросов криптографических исследований, но до нашего столетия все предложенные системы рано или поздно вскрывались. Однако в 1920-х годах был изобретен «одноразовый блокнот» и показано, что вскрыть его невозможно (Kahn 1967, стр. 398–400). Теоретическая основа этой и родственных ей систем была поставлена на прочный фундамент спустя еще четверть века, благодаря разработке теории информации (Shannon 1949). Для одноразовых блокнотов требуются очень длинные ключи, поэтому в большинстве приложений они оказываются запретительно дорогими.

С другой стороны, безопасность большинства криптографических систем коренится в вычислительной трудности криптоанализа с целью восстановить простой текст без знания ключа. Эта проблема относится к епархии теории вычислительной сложности и анализа алгоритмов, двух недавно сформировавшихся дисциплин, которые изучают трудность решения вычислительных задач. Применяя результаты этих теорий, в обозримом будущем, возможно, удастся обобщить доказательства безопасности на более полезные классы систем. Эта возможность изучается в §42.6.

Прежде чем переходить к новым результатам, мы в следующем разделе введем терминологию и определим угрозы.

42.2. Традиционная криптография

Криптография – это изучение «математических» систем для решения двух видов проблем безопасности: конфиденциальность и аутентификация. Конфиденциальная система предотвращает несанкционированное извлечение информации из сообщений, передаваемых по открытым каналам, и тем самым гарантирует отправителю сообщения, что оно будет прочитано только предполагаемым получателем. Система аутентификации предотвращает несанкционированную вставку сообщений в открытый канал, гарантируя получателю сообщения подлинность его отправителя.

Канал считается открытым, если его безопасность не отвечает потребностям пользователей. Поэтому, скажем, телефонную линию одни пользователи считают закрытым каналом, а другие – открытым. Любому каналу может угрожать прослушивание или внедрение сообщения или то и другое сразу в зависимости от того, как он используется. В телефонной связи угроза внедрения очень велика, потому вызываемая сторона не может определить,

с какого телефона звонят[1]. Прослушивание, требующее физического подключения к линии, технически сложнее и чревато уголовным преследованием. В радиосвязи ситуация прямо противоположная. Прослушивание пассивно и не влечет юридической ответственности, тогда как незаконный передатчик, необходимый для внедрения, можно обнаружить и привлечь оператора к уголовной ответственности.

Разделив проблему на две – конфиденциальность и аутентификация, – мы иногда будем подразделять аутентификацию на аутентификацию сообщения, т. е. ту проблему, которая была описана выше, и аутентификацию пользователя, единственная задача которой – проверить, что индивидуум действительно является тем, за кого себя выдает. Например, личность индивидуума, предъявившего кредитную карту, необходимо верифицировать, но нет никакого сообщения, которое он хотел бы передать. Несмотря на кажущееся отсутствие сообщения при аутентификации пользователя, обе проблемы, по большому счету, эквивалентны. При аутентификации пользователя имеется неявное сообщение «Я ПОЛЬЗОВАТЕЛЬ X», тогда как аутентификация сообщения сводится к проверке подлинности стороны, отправившей сообщение. Однако иногда из-за различий в характере угроз и других аспектов двух этих подпроблем удобно их различать.

На рис. 42.1 показан поток информации в традиционной криптографической системе, используемой для обеспечения конфиденциальности связи. Имеется три стороны: отправитель, получатель и перехватчик. Отправитель генерирует открытый текст, или незашифрованное сообщение P, которое нужно передать по небезопасному каналу законному получателю. Чтобы помешать перехватчику узнать P, отправитель применяет к P обратимое преобразование S_K, порождая шифртекст, или криптограмму $C = S_K(P)$. Ключ K передается только законному получателю по безопасному каналу, изображенному на рис. 42.1 в виде экранированной линии. Поскольку законный получатель знает K, он может дешифрировать C, применив преобразование S_K^{-1} и получив $S_K^{-1}(C) = S_K^{-1}S_K(P) = P$, т. е. исходное открытое сообщение. Безопасный канал нельзя использовать для передачи самого P из-за задержки и недостаточной пропускной способности. Например, в качестве безопасного канала можно было бы использовать еженедельного курьера, а в качестве небезопасного канала – телефонную линию.

Криптографическая система – это однопараметрическое семейство $\{S_K\}_{K\in\{K\}}$ обратимых преобразований $S_K : \{P\} \to \{C\}$ из пространства $\{P\}$ открытых сообщений в пространство $\{C\}$ зашифрованных сообщений. Параметр K называется ключом и выбирается из конечного множества $\{K\}$, называемого пространством ключей. Если пространства сообщений $\{P\}$ и $\{C\}$ совпадают, то будем обозначать их $\{M\}$. При обсуждении конкретных криптографических преобразований S_K мы иногда будем опускать упоминание о системе и называть преобразование просто K.

---

[1] Статья написана в 1976 году, когда услуги «определение номера» в стационарной телефонной связи еще не было. Сейчас, в связи с развитием IP-телефонии, актуальнее прямо противоположная угроза: абонент отлично видит, с какого номера звонят, только сам этот номер подменный. – *Прим. перев.*

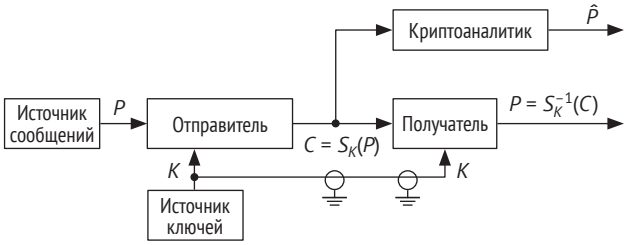


Рис. 42.1. Поток информации
в традиционной криптографической системе

Цель проектирования криптосистемы $\{S_K\}$ заключается в том, чтобы сделать операции шифрования и дешифрирования недорогими, но при этом гарантировать, что любая успешная криптоаналитическая операция будет слишком сложной и экономически неоправданной. К решению этой проблемы есть два подхода. Система, которая безопасна в силу вычислительных затрат на криптоанализ, но уступит атаке с неограниченной вычислительной мощностью, называется *вычислительно стойкой*, а система, способная выдержать любую криптографическую атаку, какие бы вычислительные мощности для нее ни привлекались, – *безусловно стойкой*. Безусловно стойкие системы обсуждаются в работах (Shannon 1949) и (Hellman 1977) и относятся к той части теории информации, называемой теорией Шеннона, которая занимается оптимальными характеристиками, достижимыми при неограниченных вычислительных ресурсах.

Причина безусловной стойкости – существование нескольких осмысленных решений криптограммы. Например, простая подстановочная криптограмма XMD, получающаяся из английского текста, может представлять открытые сообщения now, and, the и т. д. Напротив, вычислительно стойкая криптограмма содержит достаточно информации, чтобы однозначно определить открытый текст и ключ. Ее стойкость связана только со стоимостью их вычисления.

Единственная широко употребительная безусловно стойкая система – одноразовый блокнот, в котором открытый текст объединяется со случайно выбранным ключом той же длины. Хотя такая система доказуемо стойкая, большой размер необходимого ключа делает ее непрактичной для большинства применений. Если явно не оговорено противное, в этой работе рассматриваются только вычислительно стойкие системы, поскольку именно они встречаются чаще всего. Говоря о необходимости разработать доказуемо стойкую криптосистему, мы исключаем те, которые неудобно использовать, в частности одноразовый блокнот, а будем иметь в виду системы, в которых ключ содержит всего несколько сотен бит и допускает реализацию с помощью либо небольшого цифрового устройства, либо нескольких сотен строк программного кода.

Мы будем называть задачу *вычислительно невыполнимой*, если стоимость решения, измеряемая объемом памяти или временем выполнения, конечна, но запретительно велика.

Как коды, исправляющие ошибки, подразделяются на сверточные и блочные, так и криптографические системы можно разбить на два широких клас-

са: *потоковые шифры* и *блочные шифры*. Потоковые шифры обрабатывают открытый текст небольшими порциями (битами или символами) и обычно порождают псевдослучайную последовательность битов, которая складывается по модулю 2 с битами открытого текста. Блочные шифры применяются чисто комбинаторным образом к большим блокам текста так, что даже малое изменение во входном блоке приводит к большому изменению выходного блока. В этой работе рассматриваются в основном блочные шифры, поскольку это свойство *распространения ошибки* ценно во многих приложениях аутентификации.

В системе аутентификации криптография используется, чтобы гарантировать аутентичность сообщения получателю. Злоумышленнику нужно помешать не только внедрять в канал совершенно новые, выглядящие аутентичными сообщения, но и создавать кажущиеся аутентичными сообщения путем комбинирования или просто повторения старых сообщений, которые он скопировал в прошлом. Криптографическая система, предназначенная для гарантии конфиденциальности, в общем случае не предотвращает второй формы злоупотребления.

Чтобы гарантировать аутентичность сообщения, добавляется информация, являющаяся функцией не только самого сообщения и секретного ключа, но также даты и времени; например, путем присоединения даты и времени к каждому сообщению и шифрования всей последовательности. Это дает уверенность в том, что сгенерировать сообщение, которое после дешифрирования будет содержать правильную дату и время, мог лишь тот, кто владеет ключом. Однако следует использовать только такие системы, в которых малое изменение шифртекста приводит к большим изменениям дешифрированного открытого текста. Такое намеренное распространение ошибки гарантирует, что если в результате сознательного внедрения шума в канал сообщение, например, «стереть файл 7» превратится в «стереть файл 8», то будет повреждена также аутентифицирующая информация. Сообщение будет отвергнуто как неаутентичное.

Первым шагом при оценке адекватности криптографических систем является классификация угроз, которым они подвергаются. Криптосистемы, применяемые для обеспечения конфиденциальности или аутентификации, подвержены следующим угрозам.

Атака на основе только шифртекста – криптоаналитическая атака, при которой криптоаналитик располагает только шифртекстом.

Атака на основе известного открытого текста – криптоаналитическая атака, при которой криптоаналитик располагает значительным объемом соответственных открытых и шифртекстов.

Атака на основе подобранного открытого текста – криптоаналитическая атака, при которой криптоаналитик может подавать неограниченное количество открытых сообщений по собственному выбору и изучать получающиеся криптограммы.

Во всех случаях предполагается, что противник знает, какая общая система $\{S_K\}$ используется, потому что эту информацию можно получить путем изучения криптографического устройства. [Примечание редактора: это предположение, известное как принцип Керхгоффа, датируется еще 1883 годом,

хотя его часто и неосмотрительно игнорируют.] Пользователи криптографии часто стараются держать оборудование в секрете, но во многих коммерческих приложениях не только требуется, чтобы общая система была открыта, но это даже оговорено в стандарте.

Атака на основе только шифртекста часто встречается на практике. Криптоаналитик пользуется лишь знанием статистических свойств используемого языка (например, в английском частота встречаемости буквы *e* равна 13 %) и некоторых «вероятных» слов (например, письмо, скорее всего, начинается обращением «Dear Sir:»). Это самая слабая из угроз, и любая система, которая уступает ей, считается абсолютно нестойкой.

Система, стойкая относительно атак на основе известного открытого текста, освобождает пользователей от необходимости хранить в секрете свои прошлые сообщения или перефразировать их перед снятием грифа секретности. Возлагать такую обязанность на пользователей системы было бы неразумно, особенно в коммерческих приложениях, где анонсы изделия или пресс-релизы могут отправляться в зашифрованном виде, но потом публикуются без изъятий. Похожие ситуации в дипломатической переписке приводили ко взлому многих предположительно стойких систем. Хотя атака на основе известного открытого текста возможна не всегда, она встречается достаточно часто, чтобы считать нестойкой систему, не способную ей противиться.

Атаку на основе подобранного открытого текста трудно реализовать на практике, но ее можно аппроксимировать. Например, если отправить предложение конкуренту, то он, возможно, зашифрует его для передачи в штаб-квартиру. Шифр, стойкий относительно атаки на основе подобранного открытого текста, позволяет пользователям не беспокоиться по поводу подбрасывания противником сообщений в их систему.

При сертификации стойкости систем разумно рассматривать более серьезные криптоаналитические угрозы, поскольку они не только дают более реалистичные модели рабочего окружения криптографической системы, но и упрощают оценку стойкости системы. Многие системы, которые трудно проанализировать, ориентируясь на атаку на основе только шифртекста, можно сразу исключить, проанализировав атаки на основе известного или подобранного открытого текста.

Из приведенных определений ясно, что криптоанализ занимается задачей идентификации систем. Атаки на основе известного и подобранного открытого текста соответствуют задачам идентификации пассивных и активных систем. В отличие от многих предметов, в которых рассматривается идентификация систем, например автоматическая диагностика отказов, цель криптографии – построение систем, которые не легко, а трудно идентифицировать.

Атаку на основе подобранного открытого текста часто называют IFF-атакой; эта терминология восходит к разработке криптографических систем «свой–чужой» (identification friend or foe) после Второй мировой войны. Система «свой–чужой» позволяет военным радиолокаторам автоматически отличать свои самолеты от вражеских. Радиолокатор отправляет изменяющийся во времени запрос самолету, который принимает запрос, шифрует

его подходящим ключом и отправляет обратно локатору. Сравнивая ответ с правильно зашифрованной версией запроса, локатор может опознать свой самолет. Когда самолет находится над вражеской территорией, вражеские криптоаналитики могут отправлять запросы и исследовать зашифрованные ответы, пытаясь определить используемый ключ аутентификации и в случае успеха организовать атаку на основе подобранного открытого текста. На практике этой угрозе противостоят, ограничивая форму запросов, которые не обязательно должны быть непредсказуемыми, но обязательно неповторяющимися.

Существуют и другие угрозы системам аутентификации, которые находятся за пределами возможностей традиционной криптографии и требуют новых идей и методов, описываемых в данной работе. *Угроза компрометации аутентификационных данных получателя* возникает в связи с ситуацией, складывающейся в многопользовательских сетях, где получателем часто является сама система. Таблицы паролей получателя и другие данные для аутентификации становятся более уязвимыми для кражи, чем данные отправителя (индивидуального пользователя). Как показано ниже, некоторые способы защиты от этой угрозы защищают также от *угрозы оспаривания*. Это означает, что факт отправки или получения сообщения отрицается отправителем. Или, наоборот, любая сторона может утверждать, что сообщение было отправлено, хотя на самом деле это не так. Необходимы исключающие возможность подделки цифровые подписи и подтверждения. Например, нечестный брокер мог бы попытаться скрыть не санкционированную клиентом продажу и покупку ценных бумаг для извлечения собственной выгоды, подделав поручения клиентов, или клиент мог бы попытаться не признать поручение, которое впоследствии привело к убыткам. Мы опишем методы, которые позволяют получателю проверить подлинность сообщения, но не дают ему возможности генерировать сообщения, которые можно было бы принять за подлинные. Таким образом, они защищают и от угрозы компрометации аутентификационных данных получателя, и от угрозы оспаривания.

42.3. Криптография с открытым ключом

Как показано на рис. 42.1, криптография всегда была производной от обеспечения безопасности. Коль скоро существует безопасный канал, по которому можно передавать ключи, безопасность можно распространить и на другие каналы с более высокой пропускной способностью или меньшей задержкой; для этого нужно лишь шифровать передаваемые по ним сообщения. В результате применение криптографии ограничивалось коммуникацией между людьми, которые приняли предварительные меры для обеспечения безопасности.

Если мы хотим разрабатывать большие и безопасные телекоммуникационные системы, то это положение необходимо изменить. При большом количестве пользователей n количество пар, потенциально желающих конфиденциально общаться между собой, еще больше – $(n^2 - n)/2$. Нереалистично

предполагать, что любая пара ранее незнакомых пользователей будет дожидаться отправки ключа по какому-то безопасному физическому каналу или что все $(n^2 - n)/2$ пар смогут организовать это загодя. В другой работе (Diffie and Hellman 1976b) авторы рассматривали консервативный подход, не требующий никаких новых разработок в самой криптографии, но ему свойственны пониженная безопасность, неудобство и ограничение на сеть в виде требования звездной конфигурации, необходимой для работы начального протокола установления соединения.

Мы предполагаем, что можно разработать системы типа той, что показана на рис. 42.2, в которых обе стороны, взаимодействуя исключительно по открытому каналу и используя только всем известные методы, могут создать безопасное соединение. Мы исследуем два подхода к этой проблеме: криптосистемы с открытым ключом и открытые системы распределения ключей. Первый более мощный и предлагает решение проблем аутентификации, описанное в следующем разделе, а второй гораздо ближе к практическому осуществлению.

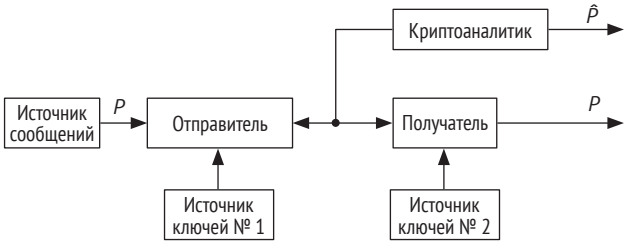


Рис. 42.2. Поток информации в системе с открытым ключом

Криптосистемой с открытым ключом называется пара семейств $\{E_K\}_{K\in\{K\}}$ и $\{D_K\}_{K\in\{K\}}$ алгоритмов, представляющих обратимые преобразования

$$E_K : \{M\} \to \{M\}$$
$$D_K : \{M\} \to \{M\}$$

на конечном пространстве сообщений $\{M\}$ такие, что

1. Для любого $K \in \{K\}$ преобразование E_K является обратным к D_K.
2. Для любого $K \in \{K\}$ и $M \in \{M\}$ алгоритмы E_K и D_K легко вычисляются.
3. Почти для любого $K \in \{K\}$ вывод любого легко вычисляемого алгоритма, эквивалентного D_K, из E_K является вычислительно неразрешимой задачей.
4. Для любого $K \in \{K\}$ нетрудно вычислить обратные пары E_K и D_K, зная K.

В силу третьего свойства пользовательский ключ шифрования E_K можно сделать открытым, не компрометируя его секретный ключ дешифрирования D_K. Таким образом, криптографическая система разбивается на две части – семейство шифрующих преобразований и семейство дешифрирующих пре-

образований – так, что, зная член одного семейства, практически невозможно найти соответствующий ему член другого семейства.

Четвертое свойство гарантирует, что имеется практически осуществимый способ вычисления соответственной пары взаимно обратных преобразований, не налагающий никаких ограничений на то, какое преобразование считать шифрованием, а какое – дешифрированием. На практике криптографическое оборудование должно включать генератор истинно случайных чисел (например, шумовой диод) для генерирования K вкупе с алгоритмом генерирования пары E_K–D_K по его выходам.

При наличии системы такого вида проблема распределения ключей сильно упрощается. Каждый пользователь генерирует пару взаимно обратных преобразований, E и D, на своем терминале. Дешифрирующее преобразование D должно храниться в секрете, но его никогда и не нужно передавать по какому-либо каналу. Шифрующее преобразование E может быть помещено в открытый каталог вместе с именем и адресом пользователя. Любой желающий может зашифровать сообщения и отправить их пользователю, но никто, кроме него самого, не сможет дешифрировать адресованные ему сообщения. Таким образом, криптосистемы с открытым ключом можно рассматривать как шифры с коллективным доступом.

Важно, чтобы открытый файл с ключами шифрования можно было защитить от несанкционированного изменения. Эта задача упрощается ввиду открытой природы файла. Защита от чтения необязательна, а т. к. файл изменяется нечасто, даже изощренные механизмы защиты от записи будут экономически оправданы.

Примером, правда, к сожалению, бесполезным, криптосистемы с открытым ключом может служить шифрование открытого текста, представленного двоичным n-вектором $\mathbf{m}$, путем умножения его на обратимую двоичную матрицу E размера $n \times n$. Криптограммой в этом случае будет $E\mathbf{m}$. Полагая $D = E^{-1}$, имеем $\mathbf{m} = D\mathbf{c}$. Таким образом, и шифрование, и дешифрирование требуют приблизительно n^2 операций. Однако вычисление D по E включает обращение матрицы, а это более трудная задача. И по крайней мере, концептуально проще получить какую-то пару взаимно обратных матриц, чем обратить заданную матрицу. Начнем с единичной матрицы I и выполним элементарные операции над строками и столбцами, чтобы получить какую-то обратимую матрицу E. Затем, начав с I, выполним обращения тех же самых элементарных операций в обратном порядке, чтобы получить $D = E^{-1}$. Последовательность элементарных операций легко построить по заданной случайной строке битов.

К сожалению, для обращения матрицы нужно всего n^3 операций. Следовательно, отношение времени криптоанализа (т. е. вычисления D по E) к времени шифрования или дешифрирования не превышает n, и, чтобы получить отношение 10^6 или больше, понадобятся огромные блоки. Кроме того, не похоже, что знание элементарных операций, использованных для получения E из I, значительно уменьшает время вычисления D. Поскольку в двоичной арифметике не бывает ошибок округления, численная устойчивость обращения матриц в данном случае не важна. Но, несмотря на отсутствие практи-

ческой ценности, этот пример с матрицами полезен для прояснения связей, необходимых в криптосистеме с открытым ключом.

В более практичном подходе к нахождению пары легко вычислимых взаимно обратных алгоритмов E и D – такой, что D сложно вывести из E, – используется трудность анализа программ на языках низкого уровня. Любой, кто пытался понять, что делает программа, написанная кем-то другим на машинном языке, знает, что саму E (т. е. что E делает) трудно вывести из знания алгоритма E. Если программа сознательно запутана путем добавления ненужных переменных и предложений, то найти обратный алгоритм может быть чрезвычайно трудно. Конечно, E должна быть достаточно сложной, чтобы помешать ее идентификации на основе пар вход–выход.

По существу, требуется односторонний компилятор, который принимает вполне понятную программу на языке высокого уровня и транслирует ее в недоступную для понимания программу на каком-нибудь машинном языке. Компилятор односторонний, потому что компиляция должна быть возможна, а обратный процесс – нет. Поскольку для такого приложения несущественны ни размер программы, ни время ее выполнения, подобные компиляторы можно реализовать, если удастся оптимизировать структуру машинного языка, так чтобы запутать код было проще.

Меркл (Merkle 1978) независимо изучал проблему распределения ключей по небезопасному каналу. Его подход отличается от криптосистемы с открытым ключом, описанной выше, и мы будем называть его *открытой системой распределения ключей*. Цель заключается в том, чтобы два пользователя, A и B, могли безопасно обменяться ключом по небезопасному каналу. Затем этот ключ используется обоими пользователями в обычной криптосистеме для шифрования и дешифрирования. Решение Меркла таково, что стоимость криптоанализа возрастает как n^2, где n – затраты законных пользователей. К сожалению, законные пользователи системы несут затрату как на передачу, так и на вычисление, потому что в протоколе Меркла необходимо передать n потенциальных ключей, прежде чем будет выбран один настоящий. Меркл замечает, что высокие накладные расходы на передачу не позволяют эффективно использовать систему на практике. Если накладные расходы на инициализацию протокола ограничены 1 мегабитом, то его метод позволяет достичь соотношения затрат 10 000:1 – слишком мало для большинства приложений. Если станут доступны недорогие широкополосные каналы передачи данных, то можно будет достичь соотношения миллион к одному и выше, тогда практическая ценность системы повысится.

Далее мы предложим новую открытую систему распределения ключей, обладающую несколькими преимуществами. Во-первых, требуется обменяться только одним «ключом». Во-вторых, затраты на криптоанализ растут экспоненциально с ростом затрат законных пользователей. И в-третьих, ее использование можно связать с общедоступным файлом, содержащим информацию о пользователях, который служит для аутентификации пользователя A пользователем B и наоборот. Сделав этот файл доступным только для чтения, мы обеспечим пользователю возможность ввести свои данные однократно и идентифицировать себя для многих пользователей. В методе

Меркла требуется, чтобы A и B проверяли подлинность друг друга иными средствами.

В новом методе используется предполагаемая трудность вычисления логарифмов над конечным полем $GF(q)$, содержащим простое число элементов, q. Обозначим

$$Y = \alpha^X \bmod q \text{ для } 1 \leqslant X \leqslant q - 1,$$

где α – фиксированный примитивный элемент $GF(q)$, тогда X называется логарифмом Y по основанию α по модулю q:

$$X = \log_\alpha Y \bmod q \text{ для } 1 \leqslant Y \leqslant q - 1.$$

Вычислить Y по X просто, требуется не более $2 \log_2 q$ умножений (Knuth 1969, стр. 398–400). Например, для $X = 18$

$$Y = \alpha^{18} = (((\alpha^2)^2)^2)^2 \times \alpha^2.$$

С другой стороны, вычислить X по Y гораздо труднее, и для некоторых тщательно выбранных значений q требуется $q^{1/2}$ операций при использовании лучшего из известных алгоритмов (Pohlig and Hellman 1978; Knuth 1973, стр. 9, 575–576).

Стойкость нашего метода критически зависит от трудности вычисления логарифмов по модулю q, и если бы был найден алгоритм, сложность которого растет как $\log_2 q$, то нашу систему можно было бы вскрыть. Возможно, что простота формулировки проблемы допускает отыскание таких простых алгоритмов, но столь же возможно, что именно по этой причине можно будет доказать трудность проблемы. Пока что будем предполагать, что лучший известный алгоритм вычисления логарифмов по модулю q действительно близок к оптимальному и что $q^{1/2}$ – хорошая мера сложности задачи при правильно выбранном q.

Каждый пользователь независимо и случайным образом выбирает случайное число X_i из равномерного распределения над множеством целых чисел $\{1, 2, \ldots, q - 1\}$. Каждый хранит X_i в секрете, но помещает $Y_i = \alpha^{X_i} \bmod q$ в общедоступный файл вместе со своим именем и адресом. Желая пообщаться конфиденциально, пользователи i и j используют в качестве ключа $K_{ij} = \alpha^{X_i X_j} \bmod q$. Пользователь i получает K_{ij}, взяв Y_j из общедоступного файла и положив

$$\begin{aligned} K_{ij} &= Y_j^{X_i} \bmod q \\ &= (\alpha^{X_j})^{X_i} \bmod q \\ &= \alpha^{X_j X_i} = \alpha^{X_i X_j} \bmod q. \end{aligned}$$

Пользователь j получает K_{ij} аналогично, $K_{ij} = Y_i^{X_j} \bmod q$. Любой другой пользователь должен вычислить K_{ij} по Y_i и Y_j, например вычислив $Kij = Y_i^{(\log_\alpha Y_j)} \bmod q$.

Таким образом, мы видим, что если логарифмы по модулю q легко вычислить, то систему можно вскрыть. В настоящее время мы не располагаем доказательством противного (т. е. что система является стойкой, если логарифмы

по модулю q трудно вычислить), но и не знаем никакого способа вычислить K_{ij} по Y_i и Y_j, не получив сначала либо X_i, либо X_j.

Если q – простое число, немного меньшее 2^b, то все величины можно представить b-битовыми числами. Тогда возведение в степень требует не более $2b$ умножений по модулю q, а, по предположению, взятие логарифмов требует $q^{1/2} = 2^{b/2}$ операций. Поэтому время криптоанализа растет экспоненциально по сравнению с временными затратами законных пользователей. Если $b = 200$, то для вычисления Y_i по X_i или K_{ij} по Y_i и X_i требуется не более 400 умножений, а для вычисления логарифмов по модулю – 2^{100}, или приблизительно 10^{30} операций.

42.4. Односторонняя аутентификация

Проблема аутентификации является, быть может, еще более серьезным препятствием к универсальному принятию телекоммуникаций для деловых сделок, чем распределение ключей. Без аутентификации невозможна никакая система, подразумевающая заключение контрактов и выставление счетов. Без нее бизнес не может функционировать. Современные электронные системы аутентификации не удовлетворяют требованию чисто цифровой, не допускающей подделки, зависящей от сообщения подписи. Они дают защиту от подделок третьей стороной, но не защищают от споров между отправителем и получателем.

Чтобы разработать систему, способную заменить текущий письменный контракт какой-то чисто электронной формой коммуникации, мы должны отыскать цифровой феномен с такими же свойствами, как у рукописной подписи. Любая сторона должна иметь возможность без труда установить аутентичность подписи, но никто, кроме законной подписавшей стороны, не должен иметь возможность создать такую подпись. Любой такой метод мы будем называть *односторонней аутентификацией*. Поскольку любой цифровой сигнал можно точно скопировать, настоящая цифровая подпись должна быть распознаваемой, не будучи известной.

Рассмотрим проблему «входа» в многопользовательскую вычислительную систему. Создавая учетную запись, пользователь выбирает пароль, который вносится в системный каталог паролей. При каждом входе в систему пользователю предлагается ввести свой пароль. Поскольку этот пароль хранится в секрете от всех остальных пользователей, фальсифицированный вход невозможен. Но тогда жизненно важно обеспечить безопасность каталога паролей, потому что содержащаяся в нем информация позволила успешно выступать от лица любого пользователя. Проблема осложняется еще и тем, что у системных операторов могут быть законные причины для доступа к этому каталогу. Разрешить такой законный доступ, но запретить все остальные виды доступа почти невозможно.

Это ставит, на первый взгляд, невозможное требование: новая процедура входа в систему должна судить об аутентичности паролей, не зная их. Несмотря на кажущуюся логическую противоречивость, этому требованию

легко удовлетворить. Когда пользователь впервые вводит свой пароль *PW*, компьютер автоматически и прозрачно для пользователя вычисляет функцию *f*(*PW*) и сохраняет в каталоге паролей ее значение, а не *PW*. При каждом следующем входе компьютер вычисляет *f*(*X*), где *X* – введенный пароль, и сравнивает *f*(*X*) с хранимым значением *f*(*PW*). В том и только в том случае, когда оба значения совпадают, пользователь считается подлинным. Поскольку функцию *f* приходится вычислять при каждом входе в систему, ее вычисление не должно занимать много времени. Миллион инструкций (около 10 центов по ценам 1970 года) представляется разумным ограничением на объем такого вычисления. Но если бы мы могли гарантировать, что для вычисления f^{-1} требуется 10^{30} или более инструкций, то лицо, сумевшее в обход всех правил заполучить каталог паролей, не смогло бы на практике получить *PW* по *f*(*PW*), а значит, не смогло бы осуществить несанкционированный вход в систему. Отметим, что программа входа не примет *f*(*PW*) в качестве пароля, потому что будет автоматически вычислять *f*(*f*(*PW*)), а это значение не совпадет с *f*(*PW*), хранящимся в каталоге паролей.

Мы предполагаем, что функция *f* – не секрет, поэтому трудным вычисление f^{-1} делает отнюдь не незнание *f*. Такие функции называются односторонними, впервые в процедурах входа их использовал Р. М. Нидхэм (Wilkes 1972, стр. 91). Они также обсуждались в двух недавних работах (Evans et al. 1974; Purdy 1974), где предложены интересные подходы к проектированию односторонних функций.

Точнее, функция *f* называется *односторонней функцией*, если для любого аргумента *x* в области определения *f* легко вычислить соответствующее значение *f*(*x*), но почти для всех *y* в области значений *f* вычислительно невозможно решить уравнение *y* = *f*(*x*) относительно *x*.

Важно отметить, что мы определяем функцию, не являющуюся обратимой с вычислительной точки зрения, но эта необратимость не имеет ничего общего с тем, что под этим понимается в математике. Обычно функция *f* называется «необратимой», если обращение точки *y* не единственно (т. е. существуют две разные точки x_1 и x_2 такие, что $f(x_1) = y = f(x_2)$). Подчеркнем, что это не та трудность обращения, которая нам требуется. Нужно, чтобы, зная значение *y* и функцию *f*, было очень трудно вычислить хоть какое-нибудь *x*, обладающее свойством *f*(*x*) = *y*. На самом деле если *f* необратима в обычном смысле слова, то задача нахождения прообраза даже упрощается. В крайнем случае, если $f(x) = y_0$ для всех *x* в области определения, то область значений *f* совпадает с $\{y_0\}$, и мы можем взять любое *x* в качестве $f^{-1}(y_0)$. Поэтому необходимо, чтобы *f* не была слишком вырожденной. Небольшая вырожденность допустима и, как обсуждается выше, по-видимому, присутствует в самом многообещающем классе односторонних функций.

Полиномы дают элементарный пример односторонних функций. Гораздо труднее найти корень x_0 полиномиального уравнения $p(x) = y$, чем вычислить значение полинома $p(x)$ в точке $x = x_0$. В работе (Purdy 1974) предлагается использовать разреженные полиномы очень высокой степени над конечными полями, для которых отношение времен вычисления, по-видимому, очень высокое. Теоретические основы односторонних функций подробно обсужда-

ются в § 42.6. И как показано в § 42.5, односторонние функции очень просто проектировать на практике.

Протокол входа в систему с использованием односторонней функции решает лишь некоторые проблемы, возникающие в многопользовательской системе. Он защищает от компрометации системные аутентификационные данные в состоянии покоя, но по-прежнему требует, чтобы пользователь отправлял системе истинный пароль. Для защиты от подслушивания нужно использовать дополнительное шифрование, а защита от оспаривания отсутствует вовсе.

Криптосистему с открытым ключом можно использовать для создания истинно односторонней системы аутентификации следующим образом. Если пользователь A хочет отправить сообщение M пользователю B, то он «дешифрирует» его своим секретным ключом дешифрирования и отправляет $D_A(M)$. Получив его, пользователь B может удостовериться в подлинности сообщения, «зашифровав» его открытым ключом шифрования E_A пользователя A. B также сохраняется $D_A(M)$ в качестве доказательства того, что сообщение пришло от A. Любой может проверить это утверждение, применив к $D_A(M)$ общеизвестную операцию E_A, чтобы восстановить M. Поскольку только A мог бы создать сообщение с таким свойством, мы сразу же получаем решение проблемы односторонней аутентификации, основанное на разработанной криптосистеме с открытым ключом.

Частичное решение проблемы односторонней аутентификации сообщений было предложено автором Лесли Лампортом из компании Massachusetts Computer Associates. В этом методе используется односторонняя функция f, отображающая k-мерное двоичное пространство в себя для k порядка 100. Желая отправить N-битовое сообщение, отправитель генерирует $2N$ случайно выбранных k-мерных двоичных векторов $x_1, X_1, x_2, X_2, \ldots, x_N, X_N$, которые хранит в секрете. Получателю сообщаются соответствующие образы относительно f: $y_1, Y_1, y_2, Y_2, \ldots, y_N, Y_N$. Впоследствии, когда нужно будет отправить сообщение $\mathbf{m} = (m_1, m_2, \ldots, m_N)$, отправитель посылает x_1 или X_1 в зависимости от того, чему равно m_1: 0 или 1. Он посылает x_2 или X_2 в зависимости от того, чему равно m_2: 0 или 1. И т. д. Получатель применяет f к первому полученному блоку и смотрит, что получилось: y_1 или Y_1, и таким образом узнает, было ли послано x_1 или X_1, а значит, и чему равно m_1: 0 или 1. Точно так же получатель может определить $m_2, m_3, \ldots, m_N$. Но получатель не может подделать ни один бит m.

Это лишь частичное решение, потому что требуется примерно 100-кратное увеличение объема исходных данных. Однако существует модификация, позволяющая устранить эту проблему, когда размер N составляет приблизительно 1 мегабит или более. Пусть g – одностороннее отображение из двоичного N-мерного пространства в двоичное n-мерное пространство, где n приближенно равно 50. Возьмем N-битовое сообщение $\mathbf{m}$ и, применив к нему g, получим n-битовый вектор $\mathbf{m}'$. Затем воспользуемся предыдущей схемой, чтобы отправить m_0. Если $N = 10^6$, $n = 50$ и $k = 100$, то при этом к сообщению будет добавлено $k_n = 5000$ бит аутентификации. Таким образом, в процессе передачи объем данных увеличился только на 5 % (или на 15 %, если учиты-

вать также начальный обмен $y_1, Y_1, \ldots, y_N, Y_N$). Хотя существует много других сообщений (в среднем 2^{N-n}) с такой же последовательностью битов аутентификации, благодаря односторонности g найти их и подделать сообщение вычислительно невозможно. На самом деле g должна быть несколько более стойкой, чем обычная односторонняя функция, поскольку противник знает не только $\mathbf{m}'$, но и один из его прообразов $\mathbf{m}$. Должно быть трудно, даже при известном $\mathbf{m}$, найти другой прообраз $\mathbf{m}'$. По-видимому, найти такую функцию не составит особого труда (см. § 42.5).

Существует еще одно частичное решение проблемы односторонней аутентификации пользователей. Пользователь генерирует пароль X, который хранит в секрете. Он сообщает системе значение $f^T(X)$, где f – односторонняя функция. В момент t правильным кодом аутентификации будет $f^{T-t}(X)$, и система сможет проверить его, применив функцию $f^t(X)$. В силу односторонности f прошлые ответы никак не помогут подделать новый. Недостаток этого решения в том, что может потребоваться значительный объем вычислений для законного входа в систему (хотя и на много порядков меньший, чем для подделывания). Если, например, t увеличивается на 1 каждую секунду и система должна работать с каждым паролем один месяц, то $T = 2{,}6$ млн. И пользователь, и система должны, следовательно, выполнить в среднем 1,3 млн итераций f при каждом входе. Хотя эта проблема преодолима, очевидно, она ограничивает применимость данной техники. Преодолеть ее можно было бы, если бы удалось найти простой метод вычислений $f^{(2^n)}$ для $n = 1, 2, \ldots$. по аналогии с $X^8 = ((X^2)^2)^2$. Ибо тогда, представив $T - t$ и t в двоичном виде, мы могли бы быстро вычислить f^{T-t} и f^t. Но может статься, что быстрое вычисление f^n вступает в противоречие с односторонностью f.

42.5. Взаимосвязи задач и закладки

В этом разделе мы покажем, что некоторые из рассмотренных выше криптографических проблем можно свести к другим и тем самым определить нестрогий порядок в соответствии с трудностью. Мы также опишем более трудную проблему закладок.

В § 42.2 мы показали, что криптографическую систему, предназначенную для обеспечения конфиденциальности, можно использовать также для аутентификации и защиты от поддельной третьей стороны. Такую систему можно использовать и для создания других криптографических объектов.

Криптосистему, стойкую относительно атак на основе известного открытого текста, можно использовать для порождения односторонней функции.

Как показано на рис. 42.3, возьмем криптосистему $\{S_K : \{P\} \to \{C\}\}_{K \in \{K\}}$, стойкую относительно атаки на основе известного открытого текста, зафиксируем $P = P_0$ и рассмотрим отображение $f : \{K\} \to \{C\}$, определенное как $f(X) = S_X(P_0)$.

Эта функция односторонняя, потому что нахождение X по известному $f(X)$ эквивалентно криптоаналитической задаче нахождения ключа по един-

ственной известной паре (открытый текст, криптограмма). Общеизвестность f теперь эквивалентна общеизвестности $\{S_K\}$ и P_0.

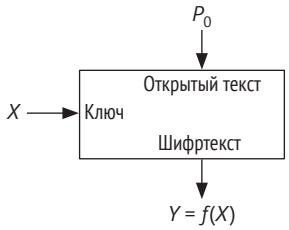


Рис. 42.3. Стойкая криптосистема, используемая в качестве односторонней функции

Хотя обратный результат необязательно верен, функцию, первоначально найденную при поиске односторонних функций, можно использовать для порождения хорошей криптосистемы. Так и случилось с дискретной экспоненциальной функцией, обсуждавшейся в § 42.3 (Pohlig and Hellman 1978).

Односторонние функции являются основой для блочных шифров и генераторов гаммы. Генератором гаммы называется генератор псевдослучайных битов, выход которого, гамма, складывается по модулю 2 с сообщением, представленным в двоичной форме, имитируя тем самым одноразовый блокнот. Ключ используется как начальное значение, определяющее псевдослучайную последовательность гаммы. Таким образом, атака на основе известного открытого текста сводится к проблеме определения ключа по гамме. Чтобы система была стойкой, вычисление ключа по гамме должно быть вычислительно неразрешимой задачей. Но чтобы система была практически полезной, вычисление гаммы по ключу должно быть простым. Таким образом, хороший генератор гаммы, почти по определению, является односторонней функцией.

У использования любого типа криптосистемы в качестве односторонней функции есть небольшая проблема. Как отмечалось выше, если функция f не является однозначно обратимой, то необязательно (а иногда даже невозможно) искать фактически использованное значение X. Подойдет любое X с таким же образом. И хотя любое отображение S_K в криптосистеме должно быть биективным, на определенную выше функцию f, отображающую ключ в криптограмму, такое ограничение не налагается. На самом деле гарантировать, что криптосистема обладает таким свойством, очень трудно. В хорошей криптосистеме можно ожидать, что отображение f обладает характеристиками случайно выбранного отображения (т. е. $f(X_i)$ равномерно выбирается из всех возможных Y и последовательные выборки независимы). В том случае, когда X выбирается равномерно и существует одинаковое количество ключей и сообщений (X и Y), вероятность, что у результирующего Y имеется $k + 1$ прообразов, приближенно равна $e^{-1}/k!$ для $k = 0, 1, 2, 3, \ldots$. Это распределение Пуассона со средним $\lambda = 1$, сдвинутое на единицу. Таким образом, ожидаемое количество прообразов равно всего лишь 2. Хотя степень вырожденности f может быть и больше, хорошая криптосистема не будет слишком вырожден-

ной, потому что это означало бы, что ключ используется плохо. В худшем случае, когда $f(X) \equiv Y_0$ для некоторого Y_0, имеем $S_K(P_0) = C_0$, и результат шифрования P_0 вообще не зависит от ключа!

Хотя обычно нас интересуют функции, для которых размеры области значений и области определения примерно одинаковы, имеются и исключения. В предыдущем разделе нам нужна была односторонняя функция, отображающая длинные строки в гораздо более короткие. Воспользовавшись блочным шифром, в котором длина ключа больше размера блока, такие функции можно получить с помощью описанной выше техники.

В работе (Evans et al. 1974) описан другой подход к построению односторонней функции на основе блочного шифра. Вместо того чтобы выбирать в качестве входа фиксированную точку P_0, они используют функцию $f(X) = S_X(X)$. Этот подход привлекателен тем, что уравнения такого вида в общем случае трудно решить, даже если семейство S сравнительно простое. Однако эта добавочная сложность разрушает эквивалентность между стойкостью системы S к атаке на основе известного открытого текста и односторонностью f.

Другая связь уже была продемонстрирована в § 42.4.

Криптосистему с открытым ключом можно использовать для порождения односторонней системы аутентификации.

Обратное, видимо, неверно, поэтому построение криптосистемы с открытым ключом – задача гораздо более трудная, чем односторонняя аутентификация. Аналогично криптосистему с открытым ключом можно использовать в качестве системы распределения ключей, но не наоборот.

Поскольку в криптосистеме с открытым ключом общая система, в которой используются E и D, должна быть открытой, задание E определяет полный алгоритм преобразования входных сообщений в выходные криптограммы. А раз так, то система с открытым ключом в действительности является набором односторонних функций с закладкой. Это функции, которые на самом деле не являются односторонними, потому что существуют легко вычисляемые обратные к ним. Но, зная алгоритм прямой функции, вычислительно невозможно найти просто вычисляемую обратную. Только полное знание какой-то информации о закладке (например, случайной строки битов, породившей пару E–D) позволит найти легко вычисляемую обратную функцию.

Закладки уже использовались в предыдущем разделе в форме односторонних функций с закладкой, но есть и другие вариации на эту тему. *Шифр с закладкой* – это шифр, стойкий относительно попыток вскрыть его криптоаналитиком, не владеющим информацией о закладке, использованной при проектировании шифра. Это позволяет проектировщику вскрывать систему, проданную клиенту, и при этом сохранять репутацию изготовителя стойких систем. Важно отметить, что вовсе не исключительная гениальность и не знание криптографии дают проектировщику возможность делать то, чего не могут другие. Ситуация в точности аналогична кодовому замку. Всякий, кто знает кодовую комбинацию, может за несколько секунд сделать то, на что у опытного взломщика уйдет несколько часов. Но стоит ему забыть комбинацию, как он теряет все преимущества.

Криптосистему с закладкой можно использовать для создания открытой системы распределения ключей.

Если *A* и *B* хотят выработать общий закрытый ключ, то *A* выбирает случайный ключ и отправляет *B* произвольную пару (открытый текст, криптограмма). *B*, который опубликовал шифр с закладкой, но сохранил информацию о закладке в секрете, использует эту пару, что найти ключ. Теперь у *A* и *B* имеется общий ключ.

В настоящее время мало известно о существовании шифров с закладками. Однако такая возможность, безусловно, существует, и о ней следует помнить, принимая криптосистему от возможного противника (Diffie and Hellman 1977).

По определению, назовем задачей с закладкой такую, в которой придумать закладку вычислительно возможно. Это оставляет место для третьего типа сущностей, при описании которых мы будем использовать префикс «квази». Например, квазиодносторонняя функция не является односторонней, поскольку легко вычисляемая обратная существует. Однако даже для ее проектировщика нахождение этой легко вычисляемой обратной функции – вычислительно неразрешимая задача. Поэтому квазиодностороннюю функцию можно использовать вместо односторонней практически без потери стойкости.

Утрата информации о закладке для односторонней функции с закладкой превращает ее в квазиодностороннюю функцию, но могут также существовать односторонние функции, которые нельзя получить таким образом.

То, что квазиодносторонние функции исключаются из класса односторонних, – вопрос определения и ничего больше. Можно вместо этого говорить об односторонних функциях в широком и строгом смысле.

Аналогично квазистойкий шифр – это шифр, который не поддается криптоанализу, даже его проектировщиком, но при этом существует вычислительно эффективный алгоритм криптоанализа (который, конечно же, вычислительно невозможно найти). Как и выше, с практической точки зрения, не существует разницы между стойким и квазистойким шифром.

Мы уже видели, что из существования криптосистем с открытым ключом вытекает существование односторонних функций с закладками. Обратное, однако, неверно. Чтобы одностороннюю функцию с закладкой можно было использовать в качестве криптосистемы с открытым ключом, она должна быть обратима (т. е. иметь единственную обратную функцию).

42.6. Вычислительная сложность

Криптография отличается от всех прочих областей науки кажущейся легкостью удовлетворения ее требований. Простые преобразования превращают ясный текст в кажущийся бессмысленным набор символов. Критику, заявляющему, что смысл все же можно восстановить посредством криптоанализа, придется провести изнурительную демонстрацию, если он пожелает доказать правильность своей точки зрения. Однако опыт показал, что лишь не-

многие системы способны устоять против слаженной атаки опытных криптоаналитиков, и многие предположительно стойкие системы впоследствии были взломаны.

Поэтому оценка достоинств новых систем всегда была главной заботой криптографов. В XVI и XVII веках для доказательства стойкости криптографических методов часто привлекались математические аргументы, в основе которых обычно лежали методы подсчета, демонстрирующие астрономическое количество возможных ключей. Хотя проблема гораздо труднее, чем можно решить такими простыми методами, даже знаменитый алгебраист Кардано попался в эту ловушку (Kahn 1967, стр. 145). Поскольку системы, чья стойкость доказывалась таким способом, раз за разом вскрывались, идея математического доказательства стойкости системы впала в немилость и была заменена сертификацией по результатам криптоаналитической атаки.

Однако в этом столетии маятник качнулся в обратную сторону. В работе, тесно связанной с рождением теории информации, Шеннон (Shannon 1949) показал, что системы на основе одноразового блокнота, которые использовались начиная с конца двадцатых годов, обладают «идеальной секретностью» (форма безусловной стойкости). Доказуемо стойкие системы, исследованные Шенноном, опираются либо на использование ключа, длина которого растет линейно с ростом длины сообщения, либо на идеальное кодирование источника, и потому слишком громоздки для большинства применений. Отметим, что ни криптосистемы с открытым ключом, ни односторонние системы аутентификации не могут быть безусловно стойкими, потому что открытая информация всегда однозначно определяет секретную, принадлежащую конечному множеству. Поэтому, располагая неограниченными вычислительными ресурсами, проблему можно решить полным перебором.

В прошлом десятилетии мы стали свидетелями расцвета двух тесно связанных дисциплин, изучающих стоимость вычислений: теории вычислительной сложности и анализа алгоритмов. Первая относит известные вычислительные проблемы к нескольким широким классам по их трудности, а предметом второй являются поиск лучших алгоритмов и изучение потребления ими ресурсов. После краткого отвлечения на теорию сложности мы рассмотрим ее применения к криптографии, в особенности к анализу односторонних функций.

Говорят, что функция принадлежит классу сложности $\mathcal{P}$ (полиномиальному), если ее можно вычислить некоторой детерминированной машиной Тьюринга за время, ограниченное сверху полиномиальной функцией от длины входа. Можно было бы подумать, что это класс легко вычислимых функций, но точнее будет сказать, что функцию, не принадлежащую этому классу, должно быть трудно вычислить по крайней мере для некоторых входов. Существуют проблемы, заведомо не принадлежащие классу $\mathcal{P}$ (Aho et al. 1974, стр. 405–425).

В технике имеется много задач, не разрешимых за полиномиальное время никакими известными методами, если только не решать их на компьютере с неограниченной степенью параллелизма. Эти задачи могут принадлежать или не принадлежать классу $\mathcal{P}$, но они принадлежат классу $\mathcal{NP}$ (недетерминированный, полиномиальный) проблем, разрешимых за полиномиальное

время на «недетерминированном» компьютере (т. е. компьютере с неограниченной степенью параллелизма). Очевидно, что класс $\mathcal{NP}$ включает класс $\mathcal{P}$, а одним из величайших открытых вопросов теории сложности является вопрос о том, действительно ли класс $\mathcal{NP}$ строго шире.

Среди проблем, относительно которых известно, что они решаются за время $\mathcal{NP}$, но неизвестно, разрешимы ли они за время $\mathcal{P}$, находится проблема коммивояжера, проблема выполнимости для исчисления высказываний, проблема укладки рюкзака, проблема раскраски графа и многие проблемы планирования и минимизации (Karp 1972; Aho et al. 1974, стр. 363–404). Мы видим, что не отсутствие интереса и не недостаточность усилий мешают найти решения этих проблем за время $\mathcal{P}$. Поэтому имеется твердое убеждение, что по крайней мере одна из этих задач не принадлежит классу $\mathcal{P}$, а потому класс $\mathcal{NP}$ строго шире.

Карп выделил подкласс $\mathcal{NP}$-проблем, который назвал $\mathcal{NP}$-полным, обладающий тем свойством, что если хотя бы одна из них принадлежит $\mathcal{P}$, то и все $\mathcal{NP}$-полные задачи принадлежат $\mathcal{P}$. Карп перечисляет 21 $\mathcal{NP}$-полную проблему, в т. ч. все перечисленные выше (Karp 1972, здесь глава 36).

Хотя у $\mathcal{NP}$-полных проблем есть потенциальные перспективы использования в криптографии, текущее понимание их трудности включает лишь анализ в худшем случае. Для применения в криптографии необходим анализ вычислительных затрат в типичном случае. Если, однако, мы примем в качестве меры сложности не время вычислений в худшем случае, а среднее или типичное время вычислений, то имеющиеся на сегодняшний день доказательства эквивалентности $\mathcal{NP}$-полных проблем перестанут быть верными. Тут открывается несколько интересных тем для исследования. Очевидную роль будут играть понятия ансамбля и типичности, знакомые специалистам в области теории информации.

Теперь мы можем указать место общей проблемы криптоанализа среди всех вычислительных проблем. *Криптоаналитическая трудность системы, в которой шифрование и дешифрирование могут быть выполнены за время $\mathcal{P}$, не может превышать трудность $\mathcal{NP}$.*

Чтобы убедиться в этом, заметим, что любую криптографическую проблему можно решить, найдя ключ, прообраз и т. д., выбираемые из конечного множества. Если всего существует M возможных ключей, то необходимо использовать M-путевой параллелизм. Например, в атаке на основе известного открытого текста открытый текст шифруется одновременно всеми ключами и результаты сравниваются с криптограммой. Поскольку, по предположению, шифрование занимает только время $\mathcal{P}$, криптоанализ займет только время $\mathcal{NP}$.

Заметим также, что общая проблема криптоанализа является $\mathcal{NP}$-полной. Это следует из широты нашего определения криптографических проблем. Далее мы обсудим одностороннюю функцию с $\mathcal{NP}$-полной обратной.

Криптографию можно непосредственно вывести из теории $\mathcal{NP}$-сложности, изучив, как можно адаптировать $\mathcal{NP}$-полные проблемы к использованию в криптографии. В частности, существует $\mathcal{NP}$-полная проблема, известная как проблема укладки рюкзака, которая легко сводится к построению односторонней функции.

Пусть $y = f(x) = \mathbf{a} \cdot \mathbf{x}$, где $\mathbf{a}$ – известный вектор n целых чисел $(a_1, a_2, \ldots, a_n)$, а $\mathbf{x}$ – двоичный вектор длины n. Вычислить y просто, нужно лишь сложить не более n целых чисел. Проблема обращения f известна под названием проблемы укладки рюкзака и заключается в нахождении подмножества $\{a_i\}$ чисел, сумма которых равна y.

Сложность исчерпывающего поиска среди всех 2^n подмножеств растет экспоненциально, поэтому для n, больших 100, задача является вычислительно неразрешимой. Однако необходимо осторожно выбирать параметры задачи, чтобы избежать легких путей. Например, если $n = 100$ и длина каждого a_i равна 32 битам, то длина y не превышает 39 бит, и f сильно вырождена, поэтому для нахождения решения в среднем требуется всего 2^{38} попыток. Еще более тривиальный случай: если $a = 2^{i-1}$, то обращение f эквивалентно нахождению двоичного разложения y....

Еще одна потенциальная односторонняя функция, представляющая интерес для анализа алгоритмов, – возведение в степень по модулю q – была предложена авторам профессором Джоном Гиллом из Стэнфордского университета. Односторонность этой функции уже обсуждалась в § 42.3.

42.7. Историческая перспектива

<...> Последняя характерная черта, которую мы замечаем в истории криптографии, – разделение криптографов на профессионалов и любителей. Навыки криптоанализа всегда были уделом профессионалов, но новации, особенно в изобретении новых типов криптографических систем, исходили преимущественно от любителей. Томас Джефферсон, криптограф-любитель, изобрел систему, которая использовалась даже во время Второй мировой войны (Kahn 1967, стр. 192–195), да и самая знаменитая криптографическая система XX века, роторная машина, была изобретена независимо четырьмя людьми, и все они были любителями (Kahn 1967, стр. 415, 420, 422–424). Мы надеемся, что это вдохновит и других обратить внимание на эту увлекательную деятельность, занятия которой в недавнем прошлом были почти полностью монополизированы государством.

43 Большой омикрон, большая омега и большая тета (1976)

Дональд Э. Кнут

Если бы я стал утверждать, что Дональд Кнут (родился в 1938 году) – величайший специалист по информатике всех времен, то, наверное, мне бы кто-нибудь да возразил, точно так же, как стали бы возражать, если бы я назвал Уилли Мейса более великим бейсболистом, чем бессмертный Бейб Рут. Но никто не назвал бы меня сумасшедшим за то, что я поставил Кнута на первое место. За свою долгую и уникальную карьеру Кнут внес в информатику вклад не меньший, чем Рут в бейсбол.

Он начал, вполне традиционно, с защиты докторской диссертации в Калифорнийском технологическом институте в 1963 году, где впоследствии работал преподавателем. Его заметки по хешированию с линейным опробованием являются одним из первых примеров математического анализа алгоритмов (Knuth 1963), но они никогда не публиковались. Рынок для таких математических раздумий был в то время ограничен; все силы были направлены на развитие систем и языков программирования. Работа 1965 года по классу синтаксических анализаторов с линейным временем работы, или LR-анализаторов (Knuth 1965), была важным вкладом в эту область. (И Рут начинал как неплохой питчер.) Кнут начал работать над книгой по анализаторам и компиляторам. Но, недовольный имеющимися учебниками по основаниям этого раздела науки, он согласился написать однотомный обзор информатики под названием «Искусство программирования» (The Art of Computer Programming, сокращенно *TAOCP*). Поначалу обзор состоял из шести глав, потом из семи, а к моменту, когда была закончена первая глава, он уже вырос до размера книги – в немалой части потому, что многие ба-

зовые алгоритмы, описанные в литературе, не подкреплялись надлежащим математическим анализом, пока его не провел Кнут. Том 1 *TAOCP* вышел в 1968 году, а теперь опубликовано уже третье издание (Knuth 1997a). Кнут перешел в Стэнфорд (как Рут – в Нью-Йорк). После этого были опубликованы и переработаны тома 2 и 3 (Knuth 1998, 1997b), том 4 оказался таким большим, что его пришлось разделить на две книги, и из печати вышел том 4A (Knuth 2011). «Отдельные выпуски» выходят в предвкушении последующих томов и переизданий.

Кнут и его ученики создавали дисциплину анализа алгоритмов, по мере того как Кнут писал свой труд. Вопросы, на которые он не мог ответить в процессе проработки математических основ и алгоритмов информатики, стали сначала задачами для сообщества, затем темами диссертаций и, наконец, целыми подразделами науки. Количество его учеников, ставших докторами наук (PhD), их учеников, ставших докторами, и их влияние на академическую сторону информатики просто ошеломляют.

Издание *TAOCP* задержалось еще и потому, что после появления первого тома издательские технологии изменились, а потому последующие издания и тома выглядели не так, как первый. Но Кнут решил не принимать такое эстетическое поругание, а обратил внимание на проблему верстки, а затем и на проектирование и рендеринг шрифтов. Результатом стали системы T_EX (Knuth, 1986b) и METAFONT (Knuth, 1986a), использованные для верстки этой книги и большей части математической литературы, публикуемой сегодня.

TAOCP не поддается ни краткому конспектированию, ни выжимкам. Вместо того чтобы выбирать для этой книги какие-то журнальные статьи Кнута, я решил остановиться на этой короткой заметке, первоначально напечатанной на машинке и опубликованной в бюллетене теоретического сообщества, сложившегося внутри Ассоциации по вычислительной технике США (аббревиатура SIGACT изначально означала «Special Interest Group on Automata and Computability Theory» – специальная группа по автоматам и теории вычислимости). Как и Лейбниц (стр. 30), Кнут оказал огромное влияние на распространение хорошей математической нотации – в данном случае для только нарождающейся дисциплины анализа алгоритмов, а ранее – для стандартизации употребления $\mathcal{P}$ и $\mathcal{NP}$ (см. стр. 418). Вопреки озорному названию («омикрон» и «омега» буквально означают «малое о» и «большое о»), эта короткая заметка показывает глубину научного стандарта, которую следует удерживать в области информатики, и живописует картину человека за работой, который часто захаживает в библиотеку Стэнфордского университета, бранит тени великих предков предыдущего столетия за неудачные определения и, наконец, в разговорном стиле говорит, что не знает, о чем еще сказать, поэтому пусть все делают, как он. Что, собственно, и произошло; эта нотация стала мировым стандартом.

Большинство из нас привыкло к идее использования нотации $O(f(n))$, применяемой для функций, абсолютная величина которых для всех достаточно больших n ограничена сверху абсолютной величиной $f(n)$, умноженной на постоянную. Иногда нужна также соответственная нота-

ция для функций, ограниченных снизу, т. е. таких, которые для достаточно больших *n не меньше* $f(n)$, умноженной на постоянную. К сожалению, иногда встречаются случаи, когда нотация O используется для обозначения нижней границы, например некоторый метод сортировки отвергается на том основании, что «его время работы составляет $O(n^2)$». Такие примеры я видел довольно часто, и, наконец, это подвигло меня сесть и написать письмо Редактору по поводу сложившейся ситуации.

В классической литературе имеется нотация для функций, ограниченных снизу, а именно $\Omega(f(n))$. Самый известный случай употребления этой нотации – фундаментальный труд Титчмарша по дзета-функции Римана (Titchmarsh 1951), где он определяет $\Omega(f(n))$ на стр. 152 и посвящает целую главу 8 «Ω-теоремам». См. также (Prachar 1957, стр. 245).

Ω-нотация не стала широко распространенной, хотя недавно я заметил ее использование в нескольких местах, в последний раз – в некоторых русских публикациях, к которым я обращался в связи с теорией равнораспределенных последовательностей. Как-то раз я предложил кому-то в письме пользоваться Ω-нотацией, «поскольку она используется в теории чисел уже много лет», но позднее, когда мой корреспондент предложил привести точные ссылки, я провел целый, на удивление, бесплодный час в библиотеке и так и не смог найти ни одной ссылки. Недавно я спросил нескольких выдающихся математиков, знают ли они, что означает $\Omega(n^2)$, и более половины ответили, что в первый раз видят такую нотацию.

Прежде чем писать это письмо, я решил провести более тщательный поиск и изучить историю O-нотации, а заодно и o-нотации. В двухтомнике Кэджори по истории математической нотации они не упомянуты ни разу. В поисках определений я встретил десяток книг, начиная с начала этого столетия, в которых были определены O и o, но не Ω. Я нашел замечание (Landau 1909, стр. 883) о том, что первое известное ему употребление O встречается в работе Бахмана (Bachmann 1894, стр. 401). Там же Ландау говорит, что он сам придумал o-нотацию, когда писал книгу о распределении простых чисел; его оригинальное обсуждение O и o см. в работе (Landau 1909, стр. 59–62).

Я не смог найти никаких упоминаний о Ω-нотации в публикациях Ландау; позже я получил подтверждение этому, когда обсуждал вопрос с Дьёрдем Пойа, который сообщил мне, что был учеником Ландау и хорошо знаком с его трудами. Пойа знал, что означает Ω-нотация, но никогда не использовал ее в собственных работах. (Каков учитель, таков и ученик, – выразился он.)

Поскольку Ω-нотация используется так редко, первые три моих похода в библиотеки не принесли почти ничего, зато во время четвертого посещения я наконец смог точно установить вероятный источник: Харди и Литтлвуд употребили ее в своем классическом сочинении 1914 года (Hardy and Littlewood, 1914, стр. 225), назвав ее «новой» нотацией. Они использовали ее также в своей главной работе по распределению простых чисел (Hardy and Littlewood 1916, стр. 125ff.), но, по-видимому, в более поздних работах такой потребности у них не возникало.

К сожалению, Харди и Литтлвуд определили $\Omega(f(n))$ не так, как я хотел бы; их определение было отрицанием $o(f(n))$, а именно – функция, которая по абсолютной величине превышает $Cf(n)$ для бесконечно многих n, когда C –

достаточно малая положительная постоянная. Во всех применениях, которые мне до сих пор встречались в информатике, гораздо больше подходит более сильное требование (заменить «бесконечно многих n» на «всех достаточно больших n»).

В результате обсуждения этой проблемы с разными людьми на протяжении нескольких лет я пришел к выводу, что следующие определения будут наиболее полезны ученым в области информатики:

- $O(f(n))$ обозначает множество всех $g(n)$ таких, что существуют положительные постоянные C и n_0, для которых $|g(n)| \leqslant Cf(n)$ для всех $n \geqslant n_0$;
- $\Omega(f(n))$ обозначает множество всех $g(n)$ таких, что существуют положительные постоянные C и n_0, для которых $g(n) \geqslant Cf(n)$ для всех $n \geqslant n_0$;
- $\Theta(f(n))$ обозначает множество всех $g(n)$ таких, что существуют положительные постоянные C, C' и n_0, для которых $Cf(n) \leqslant g(n) < C'f(n)$ для всех $n \geqslant n_0$.

Словами $O(f(n))$ можно прочитать как «порядка не больше $f(n)$», $\Omega(f(n))$ – как «порядка не меньше $f(n)$», а $\Theta(f(n))$ – как «порядка в точности $f(n)$». Конечно, эти определения применимы только к поведению при $n \to \infty$; имея дело с поведением $f(x)$ при $x \to 0$, мы должны были бы подставить окрестность нуля вместо окрестности бесконечности, т. е. писать $|x| \leqslant x_0$ вместо $n \geqslant n_0$.

Хотя я изменил определение Ω, данное Харди и Литтлвудом, я чувствую себя вправе сделать это, потому что их определение никак не назовешь широко распространенным и потому что существуют другие способы выразить то, что они хотели выразить, в тех редких случаях, когда применимо их определение. Мне нравится мнемоническое употребление Ω по аналогии с O, и его легко использовать в типографском наборе. Кроме того, оба определенных выше обозначения удачно дополняются Θ-нотацией, которая была независимо предложена Бобом Тарьяном и Майком Патерсоном.

В данных выше определениях употребляется фраза «множество всех $g(n)$ таких, что...», а не «любую функцию $g(n)$, обладающую тем свойством, что...»; я полагаю, что это определение в терминах множеств, которое было предложено мне много лет назад Роном Ривестом как улучшение определения, приведенного в первом издании моего тома 1, – лучший способ определить O-нотацию. При такой интерпретации, используя O-нотацию и родственные ей в формулах, мы фактически говорим о множествах функций, а не об одиночных функциях. Если A и B – множества функций, то $A + B$ обозначает множество $\{a + b : a \in A \text{ и } b \in B\}$ и т. д.; а «$1 + O(n^{-1})$» можно считать множеством всех функций вида $1 + g(n)$, где $|g(n)| \leqslant Cn^{-1}$ для некоторого C и всех достаточно больших n. В этой связи возникает феномен *односторонних равенств*, т. е. мы пишем $1 + O(n^{-1}) = O(1)$, но не $O(1) = 1 + O(n^{-1})$. Здесь знак равенства означает $\subseteq$ (включение множеств), и это беспокоило многих людей, предлагавших запретить использование знака = в этом контексте. Мне кажется, что следует и дальше использовать одностороннее равенство в сочетании с O-нотацией, потому что это давняя практика, привычная многим тысячам математиков, и потому что мы достаточно хорошо понимаем семантику существующей нотации.

Мы могли бы также определить $\omega(f(n))$ как множество всех функций, для которых отношение к $f(n)$ не ограничено, по аналогии с $o(f(n))$. Лично я не

испытываю потребности в такой *o*-нотации; напротив, я пришел к выводу, что очень дисциплинирует именно поиск *O*-оценок, поскольку так я узнал о более мощных математических методах. Однако я готов к тому, что в один прекрасный день может возникнуть необходимость в отходе от этого правила и использовании *o*-нотации, – если я столкнусь с функцией, для которой не смогу доказать более сильную оценку.

Отметим, что в приведенных выше определениях O, Ω и Θ имеет место некоторая асимметрия, потому что абсолютные величины $g(n)$ используются только в случае O. На самом деле это не есть аномалия, т. к. O ссылается на окрестность нуля, тогда как Ω – на окрестность бесконечности. (В книге Харди по расходящимся рядам используются обозначения O_L и O_R, когда требуется односторонний результат в терминах O. Харди и Литтлвуд использовали Ω_L и Ω_R соответственно для функций, которые бесконечно часто принимают значения $< -Cf(n)$ и $> Cf(n)$. Ни одно из этих обозначений не получило широкого распространения.)

Предполагается, что приведенная выше нотация будет полезна в подавляющем большинстве применений, но не во всех мыслимых случаях. Например, при работе с функцией типа $(\log\log n)^{\cos n}$ может возникнуть желание определить нотацию для «всех функций, которые колеблются между $\log\log n$ и $1/\log\log n$, где эти пределы наилучшие из возможных». В таком случае будет достаточно ввести локальную нотацию для этой цели, ограниченную только страницами той работы, которую вы пишете; нет нужды заботиться о стандартной нотации для понятия, если только это понятие не возникает достаточно часто.

Я хотел бы закончить это письмо обсуждением альтернативного способа обозначить порядок роста функции. В своих библиотечных разысканиях я наткнулся на удивительный факт – этот альтернативный подход в действительности даже предшествует самой O-нотации. Поль Дюбуа-Реймон (Paul du Bois-Reymond 1870) использовал реляционную нотацию

$$g(n) \prec f(n)\,,\, f(n) \succ g(n)$$

еще в 1871 году для обозначения положительных функций $f(n)$ и $g(n)$, имея в виду ту же семантику, которую сегодня мы выражаем записью $g(n) = o(f(n))$ (или $f(n) = \omega(g(n))$). В интересном трактате «О порядках бесконечности» (Hardy 1924) Харди обобщает эту нотацию, используя символы сравнения

$$g(n) \preccurlyeq f(n)\,,\, f(n) \succcurlyeq g(n),$$

означающие, что $g(n) = O(f(n))$ (или, эквивалентно, что $f(n) = \Omega(g(n))$, т. к. мы предполагаем, что f и g положительны). Харди также писал

$$f(n) \asymp \mathrm{g(n)},$$

когда $g(n) = \Theta(f(n))$, и

$$f(n) \eqsim g(n),$$

когда $\lim_{n\to\infty} f(n)/g(n)$ существует и не равен ни 0, ни 1. Он же писал

$$f(n) \sim g(n),$$

когда $\lim_{n\to\infty} f(n)/g(n) = 1$. (Нотация Харди $\asymp$ может показаться странной, если не знать, что он с ней делал; например, он доказал следующую изящную теорему: «Если $f(n)$ и $g(n)$ – произвольные функции, рекурсивно построенные из обыкновенных арифметических операций и функций exp и log, то имеет место ровно одно из трех соотношений: $f(n) \prec g(n)$, $f(n) \asymp g(n)$ или $f(n) \succ g(n)$».)

С годами великолепная нотация Харди подверглась искажениям. Например, Виноградов (Vinogradov 1954) пишет $f(n) \ll g(n)$ вместо $f(n) \preccurlyeq g(n)$; таким образом, Виноградову вполне комфортно работать с формулой

$$200^2 \ll \binom{n}{2}$$

– в отличие от меня. Как бы то ни было, такая реляционная нотация обладает интуитивно понятными свойствами транзитивности, и в ней не возникают односторонние равенства, так беспокоящие некоторых. Но тогда почему бы им не отказаться от O и новых символов Ω и Θ?

Основная причина, по которой O так удобна, заключается в том, что ее можно использовать прямо в середине формул (и в середине английских предложений, и в таблицах, содержащих время работы семейства родственных алгоритмов, и т. д.). Реляционная нотация заставляет нас переносить все, кроме оцениваемой функции, в одну часть равенства. (См. Prachar [1957 стр. 191].) Простые выкладки типа

$$\begin{aligned}\left(1+\frac{H_n}{n}\right)^{H_n} &= \exp(H_n \ln(1+H_n/n)) \\ &= \exp\left(H_n(H_n/n + O(\log n/n)^2)\right) \\ &= \exp\left(H_n^2/n + O((\log n)^3/n^2)\right) \\ &= \exp\left((\ln n+\gamma)^2/n + O((\log n)^3/n^2)\right) \\ &= \left(1+O\left((\log n)^3/n^2\right)\right)e^{(\ln n+\gamma)^2/n}\end{aligned}$$

оказались бы крайне громоздкими в реляционной нотации.

Когда я работаю над задачей, в моих заметках часто появляются ситуативные обозначения, и я использую выражения типа «$(\sim 5n^2)$», чтобы обозначить множество всех функций $\preccurlyeq 5n^2$. Аналогично я могу написать «$(\sim 5n^2)$», имея в виду функции, асимптотически равные $5n^2$, и т. д.; а «$(\preccurlyeq n^2)$» было бы тогда эквивалентно $O(n^2)$, если бы я подходящим образом обобщил отношение $\preccurlyeq$ на функции, которые могут быть отрицательными. Это было бы единообразное соглашение об обозначениях для вещей любого рода, пригодное для использования в середине выражений и дающее больше, чем предложенные выше O и Ω.

Но вопреки сказанному я предпочитаю в публикациях статей использовать нотацию O, Ω и Θ; другую нотацию, например «$(\sim 5n^2)$», я использовал бы, только столкнувшись с ситуацией, где она необходима. Почему? Основ-

ная причина заключается в том, что O-нотация настолько универсальна и общепринята, что я не чувствую себя вправе заменять ее нотацией «($\preccurlyeq f(n)$)» собственного изобретения, пусть даже логически оправданной; O-нотация ныне приобрела важное мнемоническое содержание, и нам с ней удобно. По схожим причинам я не отказываюсь от десятичной нотации, хотя считаю восьмеричную (к примеру) более логичной. И мне нравятся нотации Ω и Θ, потому что у них есть мнемонические коннотации, унаследованные от O.

Ну что же, думается, что я забил этот вопрос до смерти и больше не могу придумать аргументов ни за, ни против введения Ω и Θ. На основе приведенного здесь обсуждения я предлагаю членам SIGACT, а также редакторам журналов по информатике и математике принять нотацию O, Ω и Θ в том виде, в каком она описана выше, если в достаточно короткие сроки не будет найдена лучшая альтернатива. И еще я предлагаю принять реляционную нотацию Харди в тех ситуациях, когда она больше подходит.

44 Социальные процессы и доказательства теорем и правильности программ (1976)

Ричард ДеМилло, Ричард Липтон и Алан Перлис

Один мой весьма уважаемый коллега считает, что в своем полемическом накале эта статья заходит настолько далеко, что ее не следовало включать в этот сборник. После первоначальной публикации в трудах одной конференции она вызвала столь ожесточенные споры, что поток ассигнований на работы, связанные с верификацией программ, значительно обмелел. Хотя методы верификации ныне широко применяются при проектировании оборудования, формальная верификация крупных программных систем по-прежнему является редкостью по причинам, которые даже Хоар, похоже, признал основательными (см. стр. 376).

Статья задела за живое, потому что потребовала разобраться, в чем же все-таки сходства и различия между информатикой и математикой – да и вообще во всей формалистической программе математики и в том, как математика работает на практике. Тот факт, что некоторые ученые, занимающиеся информатикой, до сих пор свирепеют при упоминании этой статьи, – свидетельство ее полемической силы и, кстати, того, что эта область науки стала достаточно зрелой, чтобы разделяться на лагери из-за чувства глубокой обиды. Споры продолжались довольно долго; спустя десять лет Джеймс Фетцер (Fetzer 1988) оскорбил обе стороны, объявив, что ДеМилло, Липтон и Перлис

предложили «неубедительные аргументы при отстаивании позиций, которые нуждаются в дальнейшем исследовании и заслуживают лучшей поддержки». Сегодня мало кто сомневается в том, что хотя цели, намеченные Хоаром, чрезмерно амбициозны, формальная верификация проектов оборудования и низкоуровневого кода не только полезна, но и необходима. Премия Тьюринга за 2007 год была вручена Эдмунду Кларку, Аллену Эмерсону и Джозефу Сифакису за их роль в развитии формальной проверки моделей – инструмента верификации как программного, так и аппаратного обеспечения.

Алан Перлис (1922–1990) был пионером в области языков программирования. В 1957 году он возглавил совместный американо-европейский комитет по проектированию алгоритмического языка, впоследствии получившего название Algol 60, – предшественника всех императивных алгоритмических языков с блочной структурой. Ему первому была присуждена премия Тьюринга в 1966 году. Перлис также запомнился своей ролью в утверждении информатики как академической дисциплины.

Перлис основал факультет информатики в учебном заведении, которое затем стало университетом Карнеги–Меллона; сегодня это один из ведущих национальных факультетов. В 1971 году он пришел на новый факультет информатики в Йельском университете и в качестве декана возглавил его развитие в конце 1970-х годов. Ведущая роль, которую он сыграл в становлении информатики как интеллектуально независимой дисциплины в этих двух учебных заведениях, заложила основы весьма влиятельных моделей для других колледжей и университетов.

Ричард ДеМилло (родился в 1947 году) и Ричард Липтон (родился в 1948 году) долгое время сотрудничают в науке. На момент написания этой статьи Липтон работал в Йеле вместе с Перлисом, сегодня ДеМилло и Липтон трудятся в Технологическом институте Джорджии, где ДеМилло возглавляет центр будущего развития высшего образования.

> Я хотел бы задать тот же вопрос, что задавал Декарт. Вы предлагаете дать точное определение логической корректности, которое должно совпадать с моим смутным интуитивным ощущением того, что такое логическая корректность. Как вы собираетесь показать, что это одно и то же? … Средний математик не должен забывать, что интуиция – это окончательный авторитет.
>
> *– Дж. Баркли Россер*

Многие люди отстаивали мнение, что компьютерное программирование должно стремиться стать более похожим на математику. Возможно – только не так, как им кажется. Цель верификации программ, попытки уподобить программирование математике, состоит в том, чтобы резко повысить уверенность в правильности функционирования некоторого элемента программного обеспечения, а в роли механизма, используемого верификаторами для достижения этой цели, выступает длинная цепочка формальных дедуктивных логических рассуждений. В математике цель состоит в том, чтобы повысить уверенность в правильности теоремы, и действительно, средством, которое математики теоретически *могли бы* использовать для достижения этой цели, является длинная цепочка формальных логических

рассуждений. Но не используют. А то, что они используют, – доказательство – это совершенно другой зверь. Доказательство вовсе не ставит в вопросе точку; в противоречие со своим названием, доказательство – лишь один шаг в направлении выработки уверенности. Мы полагаем, что в конечном итоге основанием для уверенности математиков в правильности теоремы является социальный процесс. И еще мы полагаем, что, поскольку в среде верификаторов никакого сравнимого социального процесса не может быть, верификация программ обречена на неудачу. Мы не видим, что она сможет хоть как-то повлиять на уверенность в правильности программ.

Стороннему наблюдателю математика кажется холодным, формальным, логическим, механическим, монолитным порождением чистого интеллекта; мы же считаем, что своими успехами математика обязана тому, что это социальный, неформальный, интуитивный, органический, глубоко человечный процесс, плод работы сообщества. Внутри математического сообщества взгляд на математику как на логическую и формальную науку развивали Бертран Рассел и Давид Гильберт в начале этого века. Им математика виделась как принципиальное шествие от аксиом к гипотезам и далее к теоремам шагами, каждый из который легко обоснуется предыдущими шагами путем применения строгих правил преобразования – их немного и все они фиксированы. Труд «Основания математики» (Principia Mathematica) стал венцом творения для формалистов. Но он также нанес смертельный удар формалистической точке зрения. Здесь нет никакого противоречия: Расселу действительно удалось показать, что обыкновенные рабочие доказательства можно свести к формальной символической дедукции. Но, написав три гигантских тома, требующих огромного напряжения от читателя, он не смог продвинуться дальше элементарных фактов арифметики. Он показал, чего можно добиться в принципе и чего нельзя сделать практически. Если бы математический процесс и вправду сводился к строгому логичному переходу от одного шага к другому, мы до сих пор считали бы на пальцах.

44.1. Вера в теоремы и доказательства

> На самом деле каждый математик знает, что доказательство не было «понято», если человек ограничился лишь пошаговой проверкой правильности дедукций, из которых оно состоит, и не сделал попытки до конца разобраться в идеях, приведших к построению именно этой конкретной цепочки дедукций, а не какой-то другой.
>
> *– Н. Бурбаки*

> Согласись со мной, если кажется, что я говорю правду.
>
> *– Сократ*

Согласно оценке Станислава Улама, математики ежегодно публикуют 200 тыс. теорем. Какая-то их часть впоследствии опровергается или не принимается по другим причинам, еще сколько-то подвергаются сомнению, а большая часть игнорируется. Лишь крохотной доле суждено быть понятыми и завоевать доверие значительной группы математиков.

Теоремы, которые оказались проигнорированными или дискредитированными, редко являются плодами трудов мошенников или несведущих. Кемпе (Kempe 1879) опубликовал доказательство гипотезы о четырех красках, которое продержалось одиннадцать лет, прежде чем Хивуд (Heawood 1890) обнаружил фатальный изъян в рассуждении. Первое сотрудничество между Харди и Литтлвудом вылилось в работу, которую они в июне 1911 года представили на заседании Лондонского математического общества; работа так и не была опубликована, потому что впоследствии они обнаружили ошибку в доказательстве (Bateman and Diamond 1978). Коши, Ламр и Куммер в разное время думали, что доказали Великую теорему Ферма (Davis 1972). В 1945 году Радемахер думал, что доказал гипотезу Римана; его результаты не только циркулировали в математическом мире, но даже были анонсированы в журнале «Time» (Davis 1972).

Недавно нам встретилась следующая группа сносок, включенных в краткий исторический очерк некоторых результатов о независимости в теории множеств (Jech, 1973):

1. Результат Проблемы 11 противоречит результатам, анонсированным Леви. К сожалению, представленное там построение невозможно довести до конца.
2. О переходе к ZF было также заявлено Мареком, но описанный набросок метода оказался неудовлетворительным и не был опубликован.
3. Противоречащий результат был анонсирован, а позже дезавуирован Трассом.
4. Пример в Проблеме 22 является конрпримером к другому условию Мостовского, который предположил его достаточность и выделил пример в качестве контрольного.
5. Результат о независимости противоречит заявлению Фейгнера о том, что из принципа конфинальности следует аксиома выбора. Ошибку обнаружил Моррис (см. исправления Фейгнера к работе [1969]).

У автора нет никакого своекорыстного интереса; он, вероятно, даже никогда не слышал о нынешних противоречиях в программировании; и уж точно, он не собирался выставлять своих друзей и коллег на посмешище. Просто невозможно описать историю математических идей, не описывая сопутствующие социальные процессы при работе над доказательствами. Суть дела не в том, что математики допускают ошибки; это и так понятно. Суть дела в том, что ошибки математиков исправляются, но не с помощью формальной символической логики, а другими математиками.

Простое увеличение числа математиков, работающих над данной проблемой, необязательно приведет к получению достойного доверия доказательства. Недавно две группы топологов, одна из США, другая из Японии, независимо друг от друга анонсировали результаты, касающиеся некоторого топологического объекта – гомотопической группы. Результаты оказались противоречивыми, а поскольку оба доказательства включали сложные символические и численные вычисления, было совершенно не очевидно, кто дал маху. Но ставки были достаточно высоки, чтобы стоять на своем, поэтому японцы и американцы обменялись доказательствами. Очевидно,

у каждой группы были веские причины любой ценой найти ошибку в чужом доказательстве, и очевидно, что либо то, либо другое доказательство ошибку содержало. Но ни японское, ни американское доказательство не удалось дискредитировать. Впоследствии третья группа ученых получила еще одно доказательство, на этот раз подкрепляющее результат американцев. Факты, свидетельствующие против доказательства японцев, были настолько весомыми, что они отказались от дальнейших споров.

Из этой истории можно извлечь мораль двоякого рода. Во-первых, само по себе доказательство не сильно поднимает нашу уверенность в истинности доказываемой теоремы. На самом деле в случае теоремы о гомотопической группе чудовищность всех предложенных доказательств наводит на мысль, что неплохо бы переосмыслить саму теорему. А во-вторых, нужно заметить, что доказательства, целиком состоящие из вычислений, необязательно правильны.

Даже простота, ясность и легкость не гарантируют правильности доказательства. История попыток доказательства постулата о параллельных особенно богата изящными, элегантными доказательствами, на поверку оказавшимися ошибочными. От Птолемея до Лежандра (который пытался снова и снова) величайшие геометры всех времен ломали головы над пятым постулатом Евклида. Хуже того, даже теперь, когда мы знаем, что этот постулат недоказуем, многие неверные доказательства настолько притягательны, что в авторитетном комментарии Хита к книгам Евклида (Euclid 1956) им не отведено почетного места; Хит помечает их курсивом, в сносках или в примечаниях на полях, чтобы не вводить в заблуждение молодых математиков, пролистывающих книгу.

Идея о том, что доказательство может в лучшем случае лишь с некоторой вероятностью отражать истину, имеет интересную связь с недавним математическим спором. В одном из последних выпусков журнала «Science» Колата (Kolata 1976) предположил, что кажущееся незыблемым понятие математического доказательства может быть подвергнуто пересмотру. Главный вопрос – не «Как верят в теоремы?», а «Во что именно мы верим, когда верим в теорему?». Существует две точки зрения на этот вопрос, которые можно грубо назвать классической и вероятностной.

Сторонники классической точки зрения говорят, что когда человек верит математическому утверждению *A*, он верит, что *в принципе* существует корректный, формальный, достоверный, пошаговый, допускающий синтаксическую проверку вывод, приводящий к *A* в подходящем логическом исчислении, например в теории множеств Цермело–Френкеля или в арифметике Пеано, вывод *A* в духе «Оснований математики», вывод, который окончательно формализует истинность *A* в двоичной аристотелевой логике: «Предложение истинно, если оно говорит, что есть, о том, что есть, и говорит, что нет, о том, чего нет». [Примечание редактора: «Говорить о существующем, что его нет, или о несуществующем, что оно есть, – [значит говорить] ложь, а говорить, что существующее есть, а несуществующего нет, – значит говорить истинное»[1].] Эта формальная цепочка рассуждений ни в коей мере не

[1] *Аристотель*. Метафизика. Кн. IV, ч. VII / пер. с греч. П. Д. Розанова и В. В. Розанова. М.: Институт философии, теологии и истории св. Фомы, 2006.

то же самое, что типичное, обыкновенное математическое доказательство. Классическая точка зрения не требует, чтобы обыкновенное доказательство сопровождалось формальным эквивалентом; напротив, существуют математически основательные причины оставить богам формализацию большинства наших аргументов. Например, один теоретик оценил, что формальное доказательство гипотез Рамануджана, апеллирующее к теории множеств и элементарному анализу, заняло бы две тысячи страниц, а длину вывода из первых принципов почти невозможно представить (Manin 1977). Но классицист верит, что формализация в принципе возможна и что выражаемая ей истина двоична – либо верно, либо неверно.

Сторонники вероятностной точки зрения утверждают, что поскольку любое очень длинное доказательство можно в лучшем случае рассматривать как вероятно правильное, то почему бы не формулировать теоремы вероятностно и не давать им вероятностные доказательства? У вероятностного доказательства может быть двойное преимущество: оно технически проще классического, двухзначного и может дать математикам возможность выделить те критические идеи, которые ведут к неуверенности в традиционных двухзначных доказательствах. Этот процесс может даже привести к получению более правдоподобного классического доказательства. Иллюстрацией вероятностного подхода является алгоритм Майкла Рабина вероятностной проверки на простоту (Rabin 1976). Для очень больших целых N все классические способы определения, является ли N простым или составным, перестают работать. Даже с помощью самого хитроумного программирования вычисления, которые необходимо проделать, чтобы узнать, является ли простым число, превышающее 10^{10^4}, займут неимоверное количество машинного времени. Идея Рабина заключалась в том, что если мы готовы удовлетвориться очень высокой вероятностью того, что N простое (или не простое), то можем уложиться в разумное время – и при этом вероятность ошибки будет исчезающе мала.

Ввиду неуверенности в том, что же считать приемлемым доказательством, – а ведь это основной элемент математического процесса – спрашивается, как математика сумела выжить и добиться такого успеха? Если доказательства мало напоминают формальное дедуктивное рассуждение, если они могут не подвергаться сомнению на протяжении поколений, а затем быть опровергнуты, если они могут содержать ошибки, которые никак не удается обнаружить, если они могут выражать всего лишь вероятность истины с определенной ошибкой – если, в конце концов, они не способны доказывать теоремы, т. е. гарантировать их истинность, а не просто вероятие истинности, даже не требуя при этом глубокого понимания, то как тогда работает математика? Как ей удается устанавливать теоремы, одновременно значительные и вызывающие доверие?

Прежде всего доказательство теоремы – это сообщение. Доказательство – не красивый абстрактный объект, обладающий независимым существованием. Нет такого математика, который бы набрел на доказательство, откинулся на спинку стула и удовлетворенно вздохнул, сознавая, что теперь-то он может быть уверен в истинности теоремы. Нет, он бежит в коридор и ищет, кто бы его выслушал. Он врывается в кабинет коллеги и бросается к доске. Он

плюет на запланированную тему и потчует собравшихся на семинар своей новой идеей. Он отвлекает своих аспирантов от работы над диссертацией, заставляя их выслушать себя. Он рвется к телефону, чтобы поведать о своем открытии коллегам в Техасе и в Торонто. В своем первом воплощении доказательство – это устное сообщение или, самое большее, набросок на доске или на салфетке.

Эта устная стадия – первый фильтр для доказательства. Если оно не вызывает волнения или веры у друзей, то мудрый математик пересмотрит его. Но если они находят его умеренно интересным или достойным доверия, то он его записывает. После того как черновик доказательства некоторое время обращался среди коллег и не утратил правдоподобия, он готовит отшлифованный вариант и предлагает его для публикации. Если рецензенты также сочтут его привлекательным и убедительным, то оно будет опубликовано и прочитано более широкой аудиторией. Если достаточное число членов этой широкой аудитории поверят ему и оно им понравится, то после некоторого периода охлаждения критический анализ публикации приобретает более спокойный характер – а действительно ли доказательство такое замечательное, как казалось поначалу, и можно ли ему верить по зрелом размышлении.

А что происходит с доказательством, когда ему поверили? Пожалуй, самый очевидный процесс – усвоение результата. То есть математик, который прочитал доказательство и поверил ему, будет пытаться перефразировать его, изложить в привычных ему терминах, встроить в собственную картину математических знаний. Вряд ли найдется два математика, усвоивших математическую концепцию совершенно одинаково, поэтому такой процесс обычно ведет к появлению нескольких версий одной и той же теоремы, каждая из которых подкрепляет веру, каждая усиливает в математическом сообществе ощущение, что исходное утверждение, скорее всего, верно. Гаусс, к примеру, получил по крайней мере полдюжины независимых доказательств своего «закона квадратичной взаимности», а сегодня известно более пятидесяти таких доказательств. Имре Лакатос в своей книге «Доказательства и опровержения» (Lakatos 1976) приводит несколько исторически точных обсуждений того, какие превращения претерпели несколько знаменитых теорем на пути от начального замысла к всеобщему принятию. Лакатос демонстрирует, что формула Эйлера $V - E + F = 2$ переформулировалась снова и снова на протяжении почти двух веков после ее первой формулировки, пока наконец не достигла нынешней устойчивой формы. Самое убедительное преобразование, которое только может иметь место, – это обобщение. Если в результате того же социального процесса, которому подверглась оригинальная теорема, удается заставить поверить в обобщенную теорему, то это сильно прибавляет правдоподобия оригинальному утверждению.

Теорема, в которую поверили, начинает использоваться. Она может играть роль леммы в объемлющих доказательствах; если это не приводит к противоречиям, то ей верят все больше и больше. Или инженеры могут воспользоваться ей, подставляя в нее физические величины. Мы доверяем классическим уравнениям теории упругости, потому что видим мосты, возведенные

на их основе; мы доверяем основным теоремам механики жидкостей и газа, потому что видим, как самолеты летают.

Внушающие доверие результаты иногда вступают в контакт с другими областями математики – а уж важные результаты непременно. Успешный перенос теоремы или метода доказательства из одного раздела математики в другой увеличивает степень нашего доверия к нему. Например, в 1964 году Пол Коэн использовал технику, названную форсингом, для доказательства одной теоремы из теории множеств (Cohen 1963); в то время его идеи казались такими радикальными, что доказательство почти никто не понял. Но затем другие исследователи интерпретировали идею форсинга в алгебраическом контексте, связали ее с более знакомыми идеями логики, обобщили понятия и нашли эти обобщения полезными. Все эти связи (наряду с другими обычными социальными процессами, приведшими к принятию доказательства) сделали идею форсинга куда более привлекательной, и сегодня она изучается всеми аспирантами, специализирующимися в теории множеств.

После достаточного усвоения, достаточного преобразования, достаточного обобщения, достаточного использования и достаточного формирования связей математическое сообщество наконец решает, что основные идеи оригинальной теоремы, к тому времени, возможно, изрядно преображенные, достигли высшей степени устойчивости. Если различные доказательства не вызывают сомнений и результаты подвергались изучению под разными углами зрения, то истинность теоремы считается окончательно установленной. Теорема считается истинной в классическом смысле слова, т. е. в том смысле, что ее можно было бы доказать с помощью формальной дедуктивной логики, хотя почти для всех теорем такой дедуктивный вывод никогда не имел и вряд ли будет иметь место.

44.2. Роль простоты

> Ибо все, что ясно и легко понимается, привлекает; сложное же отталкивает.
>
> *– Давид Гильберт*

> Иногда приходится говорить трудные вещи, но говорить их нужно так просто, как умеешь.
>
> *– Г. Х. Харди*

Как правило, самые важные математические проблемы формулируются ясно и просто. Важная теорема с гораздо большей вероятностью принимает форму *A*, чем форму *B*.

A Каждое – – есть – –.

B Если – – и – – и – – и – – и – – за исключением особых случаев

a – –
b – –
c – –,

то, если не

i – – или
ii – – или
iii – –,

каждое – –, удовлетворяющее условию – –, есть ––.

Проблемы, которые сильнее всего очаровывали, мучили и восхищали математиков на протяжении столетий, всегда формулировались очень просто. Эйнштейн считал, что о зрелости научной теории можно судить по тому, насколько хорошо ее можно объяснить случайному прохожему. Теорема о четырех красках покоится на таком непритязательном фундаменте, что ее можно во всей полноте объяснить ребенку. Если ребенок выучил таблицу умножения, то он может понять проблему распределения простых чисел. А глубокое обаяние проблемы определения того, что такое «число», может сделать из него математика.

Корреляция между важностью и простотой – не случайность. Простые и привлекательные теоремы имеют больше шансов быть услышанными, прочитанными, усвоенными и используемыми. Для математиков простота – первый оселок, на котором проверяется доказательство. Только если оно выглядит интересным с первого взгляда, математик станет рассматривать его подробно. Математики – не мазохисты-альтруисты. Напротив, история математики – это история долгого поиска простоты, удовольствия и элегантности – в области символов, конечно.

Даже не желая того, математики были бы вынуждены использовать критерий простоты; просто психологически невозможно выбрать из 200 000 претендентов на наше внимание что-то отличное от самого простого и самого привлекательного. Если в математике и существуют важные, фундаментальные, но не простые концепции, то математики, скорее всего, никогда их не откроют. Неуклюжие, уродливые математические утверждения, относящиеся только к пустяшным классам структур; идиосинкратические утверждения; утверждения, опирающиеся на несуразно громоздкий математический аппарат; утверждения, требующие для приблизительной формулировки пяти классных досок или рулона бумажных полотенец, – все они никогда не сольются с телом математики. А только такое слияние внушает доверие к доказательству. Само по себе доказательство – ничто; лишь будучи подвергнуто социальным процессам математического сообщества, оно становится достойным доверия.

В этой статье мы стремились подчеркнуть, что простота превыше всего, потому что это первый фильтр для любого доказательства. Но мы не хотим изображать себя и наших друзей-математиков филистерами или дикарями. Коль скоро идея отвечает критерию простоты, прочие стандарты помогут определить ее место среди идей, заставляющих математиков отстраненно смотреть вдаль. Юрий Манин нашел для этого прекрасные слова: хорошее доказательство делает нас мудрее.

44.3. Недоверие к верификации

> Напротив, в формальной логике я не нахожу для исследователя ничего, кроме оков. Она ничуть не помогает в достижении краткости, отнюдь; если требуется двадцать семь уравнений, чтобы установить, что 1 – число, то сколько же их понадобится для доказательства настоящей теоремы?
>
> *– Анри Пуанкаре*

> Одна из основных обязанностей математика, приглашенного учеными в качестве консультанта ... – предупреждать их, чтобы они не ожидали слишком многого от математики.
>
> *– Норберт Винер*

Математические доказательства повышают нашу уверенность в истинности математических утверждений только после того, как были подвергнуты действию социальных механизмов математического сообщества. Те же самые механизмы неумолимо действуют и для так называемых доказательств правильности программ, длинных формальных верификаций, которые соответствуют не рабочему математическому доказательству, а воображаемой логической конструкции, которую математик творит, чтобы описать свое ощущение веры. Верификации – не сообщения; человек, который выбежит в коридор, чтобы поделиться своей последней верификацией, быстро станет для общества парией. Верификации невозможно по-настоящему прочитать; читатель, возможно, и решится героическими усилиями продраться сквозь колючки тех, что покороче, но это не чтение. Будучи нечитаемыми и – в буквальном смысле – непроизносимыми, верификации, в отличие от теорем, не поддаются усвоению, трансформированию, обобщению, использованию, связыванию с другими дисциплинами; в них нужно либо верить слепо, либо не верить вовсе.

На этом месте некоторые поборники верификации признают, что аналогия с математикой хромает. Утверждая, что *A*, программирование, напоминает *B*, математику, а затем выяснив, что *B* – вовсе не то, что они себе представляли, они готовы сменить точку зрения и утверждать, что *A* похоже на *B'*, придуманную ими мифическую версию *B*. Теперь мы оказываемся в нелепом положении, выдвигая аргумент, который они изначально считали своим, и утверждая, что да, действительно, *A* напоминает *B*; однако в нашем понимании эти термины сопоставляются совсем не так, как в их (см. рис. 44.1 и 44.2).

Математика		*Программирование*
теорема	...	программа
доказательство	...	верификация

Рис. 44.1. Оригинальная аналогия верификаторов

Математика		*Программирование*
теорема	...	спецификация
доказательство	...	программа
воображаемая формальная демонстрация	...	верификация

Рис. 44.2. Наша аналогия

Верификаторы, которые хотели бы отказаться от метафоры и подставить *B'*, должны бы, дабы способствовать лучшему пониманию, отказаться и от терминологии *B* – в частности, хорошо было бы не называть свои верификации «доказательствами». Что до нас самих, то мы и дальше будем утверждать, что программирование похоже на математику и что те же самые социальные процессы, которые работают в математических доказательствах, свойственны и верификациям.

Существует фундаментальное логическое возражение против верификации, возражение, взросшее именно на почве формалистской строгости. Поскольку требования к программе неформальны, а сама программа формальна, должен присутствовать некий переход, и сам этот переход по необходимости должен быть неформальным. Мы были огорчены, узнав, что утверждение, которое кажется нам самоочевидным, является спорным. Поэтому мы считаем нужным подчеркнуть, что, будучи антиформалистами, мы не стали бы возражать против верификации на этом основании; нам только интересно, как этот принципиально неформальный шаг вписывается в формалистскую точку зрения. Неужто поборники верификации упустили из виду неформальное происхождение формальных объектов, с которыми имеют дело? Не они ли утверждают, что их формализации неопровержимы? Должны признаться, что испытываем замешательство и тревогу.

Есть еще одно логическое затруднение, почти такое же фундаментальное, но ни в коем случае не такое мелочное, как приведенное выше. Формальная демонстрация того, что программа отвечает своим спецификациям, имеет ценность, только если спецификации и программы были выведены независимо. В атмосфере модельных программ, окружающей экспериментальную верификацию, этому критерию легко удовлетворить. Но в реальной жизни, если процесс проектирования программы терпит неудачу, программу изменяют, и изменения основаны на знании спецификаций; либо же изменяют спецификации, и эти изменения основаны на знаниях о программе, приобретенных в результате неудачи проектирования. В любом случае требование независимых, перекрестно проверяемых критериев более не удовлетворяется. Надеемся все же, что никто не осмелится предложить, что программы и спецификации не должны многократно модифицироваться в процессе проектирования. Это стало бы позицией немыслимой нищеты – того вида нищеты, который, как мы опасаемся, является результатом безрассудного увлечения формальной логикой.

Но вернемся в реальный мир. Спецификации ввода-вывода, сопровождающие производственное программное обеспечение, редко бывают простыми.

Обычно они длинные, сложные и специфические. Вот вам крайний случай: французские железные дороги требуют более 3000 тарифов (при движении вверх, при движении вниз и т. д.). Спецификации любого сколько-нибудь разумного компилятора или операционной системы занимают тома – и никто не считает, что они полны. Существует даже код, устроенный по принципу черного ящика: численные алгоритмы, которые работают в том смысле, что используются при конструировании летательных аппаратов или нефтяных скважин, но вот почему, никто не знает; исходные предположения для таких алгоритмов невозможно даже сформулировать, не то что формализовать. Приведем только один пример – про важный алгоритм с юмористическим названием «обратный алгоритм Катхилла–Макки» много лет было известно, что он гораздо лучше простого алгоритма Катхилла–Макки, описанного эмпирически на основании лабораторных и полевых испытаний и по результатам эксплуатации. Но лишь недавно его превосходство было доказано теоретически (George 1971), и даже тогда только с помощью обычного неформального математического доказательства, а не формального вывода. А все те годы, когда обратный алгоритм Катхилла–Макки оставался недоказанным, программисты упрямо использовали его, хотя он автоматически делал любую программу, в которой применялся, неверифицируемой.

Можно возразить, что, будучи длинными и сложными, реальные спецификации не глубоки. Их верификация сводится, по сути дела, к очень длинной цепочке подстановок, которые можно проверить с помощью простых алгебраических тождеств. В ответ мы можем сказать лишь одно: именно так. Верификации длинные, запутанные, но поверхностные; в том-то и беда. Верификация даже самой жалкой программы может растянуться на десятки страниц, и ни на одной из них не будет даже проблеска остроумия. Никто не ворвется в кабинет коллеги, чтобы продемонстрировать ему верификацию программы. Никто не станет набрасывать черновик верификации на бумажной салфетке. Никто не возьмет коллегу за пуговицу, чтобы заставить его выслушать верификацию. Никто никогда не прочтет ее. Стоит только подумать об этом, как взор стекленеет. Звучали предложения, что языки сверхвысокого уровня, способные работать напрямую с широким кругом математических объектов, или функциональные языки, которые, как говорят, можно лаконично аксиоматизировать, можно было бы использовать, чтобы сделать верификацию интересной, а потому имеющей потенциал стать предметом социального процесса, такого же, как в математике. В теории эта идея звучит обнадеживающе, но на практике результатов пока не видно. <…>

Некоторые верификаторы допускают, что верификация попросту неприменима к подавляющему большинству программ, но говорят, что для некоторых особо ответственных приложений на эти мучения следует пойти. И упоминают при этом управление воздушным движением, ракетные системы и исследование космоса как области, в которых риски настолько высоки, что любые затраты времени и усилий оправданы.

Но даже если бы это было так, мы все равно настаивали бы, что верификация должна отказаться от претензий на все прочие области программирования; например, учить верификации студентов на начальных курсах программирования – все равно, что на начальном курсе биологии учить студентов

делать операции на открытом сердце; именно так это и следует воспринимать. Но высокие ставки не способны поколебать нашу уверенность в принципиальной невозможности верифицировать любую систему, достаточно большую и гибкую, чтобы справиться с решением какой-то реальной задачи. Ни за какое вознаграждение никто не заставит себя прочитать невыносимо длинную и скучную верификацию реальной системы, а если ее нельзя прочитать, понять и улучшить, то верификация бесполезна.

Можно, конечно, возразить, что все эти разговоры об удобочитаемости и усвоении к делу не относятся, что конечная цель верификации – построить автоматическую верифицирующую систему. К сожалению, имеется множество свидетельств в пользу того, что о полностью автоматических системах верификации даже речи быть не может. Нижние границы длины формальных доказательств математических теорем огромны (Stockmeyer 1974), и нет никаких оснований полагать, что доказательства правильности программ будут короче или чище – скорее, наоборот. На самом деле даже самые ярые сторонники верификации программ не относятся к возможности создания полностью автоматических верификаторов серьезно. Ральф Лондон, сторонник верификации, говорит об автономной системе, которую можно оставить без присмотра, пока она трудится над верификацией, но даже он сомневается, что такая система может быть достаточно надежной. Одна группа, отчаявшись увидеть автоматизацию в обозримом будущем, предложила проводить верификацию группами «ворчащих математиков», т. е. силами математических коллективов низкого уровня, которые будут проверять условия верификации. Менталитет людей, способных выдвинуть такое предложение, кажется странным, но это показывает, насколько отдаленной видится перспектива автоматической верификации.

Предположим, однако, что автоматический верификатор все-таки можно каким-то образом построить. Предположим еще, что программисты каким-то образом уверовали в надежность его верификаций. В отсутствие оснований для таких упований в реальном мире, это был бы акт слепой веры, но не важно. Предположим, что философский камень найден, что свинец можно превратить в золото и что программистов удалось убедить в том, что забрасывать их программы в разверстую пасть верификатора – хорошо и правильно. Нам кажется, что сценарий, представляющийся взору поборников верификации, выглядит примерно так: программист подает свой пакет ввода-вывода, насчитывающий 300 строк, верификатору. Через несколько часов он возвращается. И тут его поджидает результат верификации на 20 000 строк и сообщение «ВЕРИФИЦИРОВАНО».

Существует одна тенденция: обретая уверенность, что конструкция логически, доказуемо правильная, мы начинаем удалять избыточные подпорки, которые первоначально встроили из-за недостатка понимания. В своем крайнем проявлении эта тенденция ведет к эффекту Титаника: когда беда все же случается, она оказывается громадной и неконтролируемой. Можно сказать и по-другому: серьезность отказа системы прямо пропорциональна силе веры проектировщика в ее безотказность. Программы, спроектированные как можно более чистыми и аккуратными просто для того, чтобы их было проще верифицировать, будут особенно уязвимы перед эффектом Титаника.

И мы уже видим признаки этого явления. В своих заметках о языке Euclid (Popek et al. 1977), спроектированном с учетом простоты верификации, некоторые из первейших сторонников верификации говорят: «Поскольку мы ожидаем, что все программы на Euclid будут верифицироваться, мы не стали принимать специальных мер для обработки исключений. ... Программные ошибки времени выполнения не должны встречаться в верифицированных программах». Ошибки не должны встречаться? Ну, точь-в-точь корабль, который не должен потонуть.

Но давайте, отбросив на минутку рациональное недоверие, предположим, что программист получил сообщение «ВЕРИФИЦИРОВАНО». И еще предположим, что это сообщение не является результатом ошибки в верифицирующей системе. Что тогда известно программисту? Ему известно, что его программа формально, логически, доказуемо, подтверждаемо правильна. Однако ему неизвестно, в какой степени она надежна, безопасна, заслуживает доверия; ему неизвестно, при каких ограничениях она будет работать; ему неизвестно, что произойдет при выходе за границы этих ограничений. И тем не менее он заполучил мистическую печать «ВЕРИФИЦИРОВАНО». Мы уже почти видим этот айсберг, нависший над непотопляемым кораблем. По счастью, причин бояться такого будущего мало. Представьте себе того самого программиста, вернувшегося, чтобы найти те самые 20 000 строк. Какое сообщение он в действительности получит в предположении, что автоматический верификатор все-таки можно построить? Конечно же, «НЕ ВЕРИФИЦИРОВАНО». Программист вносит изменение, снова загружает программу и снова возвращается. «НЕ ВЕРИФИЦИРОВАНО». Еще одно изменение, еще одна загрузка программы в верификатор. И снова «НЕ ВЕРИФИЦИРОВАНО». Программа – плод труда человека; реальная программа – сложный плод труда человека. А любой плод труда человека достаточного размера и сложности несовершенен. Сообщения «ВЕРИФИЦИРОВАНО» он не увидит никогда.

44.4. Роль непрерывности

> Огрубляя, мы можем сказать, что математическая идея «значительна», если ее можно связать естественным и поучительным образом с большим комплексом других математических идей.
>
> *– Г. Х. Харди*

Единственный действительно соблазнительный аргумент из когда-либо предлагавшихся в защиту верификации – масштабирование. Насколько мы в состоянии его воспроизвести, звучит он следующим образом:

1. Верификация пока находится на стадии младенчества. В настоящий момент она способна верифицировать разве что алгоритмы типа FIND и модельные программы типа GCD[1]. Со временем ей станут подвластны все более сложные алгоритмы и все более изощренные модельные программы. Эти верификации сопоставимы с математическими до-

[1] Наибольший общий делитель. – *Прим. перев.*

казательствами. Их читают. Они возбуждают такое же волнение и интерес, как и теоремы. Они являются предметом обычных социальных процессов, которые работают для математических рассуждений или, если на то пошло, для рассуждений в любой другой дисциплине.

2. Большие производственные системы составлены из тех же алгоритмов и модельных программ. Будучи один раз верифицированы, алгоритмы и модельные программы могут объединяться в крупные обыденные производственные системы, а (согласны, нечитаемая) верификация большой системы будет являться суммой большого числа маленьких привлекательных интересных верификаций ее компонентов.

Против пункта (1) мы не возражаем. Действительно, правильность алгоритмов доказывалась, и эти доказательства читались и ассимилировались задолго до изобретения компьютеров – и при очевидном отсутствии формальных механизмов. Осмелимся предположить, что изучение алгоритмов и модельных программ будет развиваться, как любая другая математическая деятельность, в основном благодаря неформальным социальным механизмам, при совсем небольшом или вовсе никаком участии формальных.

А вот с пунктом (2) мы не согласны фундаментально. Мы утверждаем, что нет никакой преемственности между миром FIND и GCD и миром производственного ПО, биллинговыми системами, которые выставляют реальные счета, диспетчерскими системами, которые планируют реальные события, билетными системами, которые выписывают реальные билеты. И мы утверждаем, что мир производственного ПО сам по себе не является непрерывным.

Ни один программист не согласится с тем, что крупные производственные системы состоят всего лишь из алгоритмов и мелких программ. Заплаты, ситуативные конструкции, пластыри и жгуты, свистелки и дуделки, клей, наведение марафета, сигнатурный код, кровь, пот и слезы и, конечно, «кухонная мойка»[1] – не правда ли, красочный жаргон программиста-практика кое-что говорит о природе структур, с которыми он работает? Пожалуй, теоретикам стоило бы прислушаться к нему. Согласно некоторым оценкам, более половины кода любой реальной производственной системы состоит из пользовательских интерфейсов и сообщений об ошибках – ситуативных неформальных структур, которые, по определению, не верифицируемы. Даже сами верификаторы иногда, похоже, осознают не допускающую верификации природу большей части реального ПО. Говорят, что Ч. Э. Р. Хоару принадлежат слова: «Во многих приложениях алгоритм не играет почти никакой роли и, конечно, не представляет почти никаких проблем». (Как бы мы хотели сообщить, что после этого он поднял руки и отказался от верификации, но не тут-то было.)

Или взглянем на разницу между миром GCD и миром производственного ПО по-другому: спецификации алгоритмов краткие и точные, тогда как спецификации реальных систем огромны, часто того же порядка величины, что сами системы. Спецификации алгоритмов крайне устойчивы, зачастую

---

[1] Отсылка к идиоме «everything but the kitchen sink», которую можно приблизительно перевести как «все, кроме птичьего молока». – *Прим. перев.*

не изменяются на протяжении десятилетий или даже столетий, тогда как спецификации реальных систем изменяются ежедневно или ежечасно (что может засвидетельствовать любой программист). Спецификации алгоритмов общие и допускают экспорт; спецификации же реальных систем индивидуальные и ситуативные. Это не различия в степени. Это видовые различия. Часочек посидеть со спящим ребенком – совсем не то же, что поднять на ноги семью из десяти человек; проблемы существенно, фундаментально различны.

И в самом мире реального производственного ПО нет непрерывности. Аргумент о масштабировании, похоже, основан на расплывчатом представлении о мире программирования как о некотором аналоге ньютоновской физики – составленном из гладких, непрерывных функций. Но на самом деле программы усеяны шипами и полны дыр и каверн. Каждый программист знает, что изменение одной строки, а иногда даже одного бита может полностью разрушить программу или покалечить ее до такой степени, что мы не сможем ни понять, ни предсказать ее поведение. А иногда даже довольно сильное изменение, похоже, не меняет ничего; фольклор изобилует байками о шалостях и актах вандализма, которые привели преступников в уныние тем, что остались совершенно незамеченными.

Есть классический научно-фантастический рассказ о путешественнике во времени[1], который отправился в доисторические джунгли понаблюдать динозавров, а вернувшись в свое время, нашел его изменившимся до неузнаваемости. Политика, архитектура, язык, даже растения и животные – все казалось неправильным, искаженным. И только сбросив одежду, он понял, что случилось. К каблуку башмака прилипло крылышко раздавленной бабочки – изъятой из прошлого и потому неспособной выполнить свою функцию в эволюции мира. Любому программисту знакомо это ощущение: тривиальное, минутное изменение вносит хаос в огромную систему. До тех пор, пока мы не будем знать больше о программировании, для всех практических целей лучше считать, что системы состоят не из прочных структур типа алгоритмов и меньших программ, а из крылышек бабочек.

Разрывная природа программирования возвещает гибель верификации. Достаточно фанатичный исследователь, возможно, и захотел бы посвятить два-три года верификации важной части программного обеспечения, если бы мог быть уверен в том, что программа останется стабильной. Но реальные программы необходимо сопровождать и модифицировать. Нет никаких причин верить в то, что верификация модифицированной программы окажется проще, чем верификация оригинальной. Нет никаких причин верить в то, что большая верификация может стать суммой более мелких. Нет никаких причин верить в то, что верификацию можно перенести на какую-то другую программу – пусть даже эта другая программа отличается от оригинальной всего одной строчкой.

И именно эта разрывность лишает возможности улучшать верификации с помощью социальных процессов, которые улучшают математические доказательства. Одинокий фанатик мог бы сконструировать свою собствен-

---

[1] Р. Брэдбери «И грянул гром». – *Прим. перев.*

ную верификацию, но у него никогда не было бы причин читать чужую, да и никто другой не захотел бы читать его верификацию. Не смогло бы сформироваться никакое сообщество. Даже самого ревностного верификатора можно было бы подвигнуть на чтение верификации, только если бы он думал, что сможет что-то использовать, заимствовать или слямзить из нее. Ничто не заставило бы его прочитать чужую верификацию, коль скоро он усвоил мысль, что никакая верификация не имеет необходимой связи ни с какой другой.

44.5. Доверие программному обеспечению

> Сама программа является единственным полным описанием того, что программа должна делать.
>
> *– П. Дж. Дэвис*

Поскольку компьютеры могут писать символы и перемещать их с пренебрежимо малыми затратами энергии, возникает соблазн прийти к заключению, что в области символов возможно все. Но реальность так просто не сдается; физику не сокрушить одним ударом. Конструировать символические структуры, не потребляя ресурсов, ничуть не более возможно, чем конструировать материальные структуры на таких же условиях. Даже в самых тривиальных математических теориях существуют простые утверждения, формальное доказательство которых было бы невыносимо длинным. Выдающаяся лекция Альберта Мейера об истории таких исследований заканчивается поразительной демонстрацией того, каким трудным может оказаться вывод даже сравнительно простых математических утверждений. Предположим, что мы кодируем логические формулы двоичными строками и задались целью построить компьютер, который будет устанавливать истинность простого набора формул, длина которых не превышает, скажем, тысячи бит. Предположим даже, что мы можем позволить себе технологию производства электронных компонентов размером с протон, соединенных бесконечно тонкими проводами. И даже при таких условиях проектируемый нами компьютер будет плотно заполнять всю наблюдаемую Вселенную. Это точное наблюдение, касающееся длины формальных выводов, согласуется с нашим интуитивным представлением о количестве деталей в самых обыденных, ничем не примечательных математических доказательствах. Мы часто употребляем обороты типа «будем предполагать без ограничения общности...» или «поэтому после перенумерации, если это необходимо...», которые заменяют огромный объем формальных деталей. Настаивать на формальной детализации было бы глупым и расточительным расходованием ресурсов. Как символические, так и материальные структуры следует проектировать крайне осмотрительно. Ресурсы ограничены, время ограничено, энергия ограничена. И даже компьютер не в силах изменить конечную природу Вселенной.

Мы предполагаем, что эти ограничения помешали приверженцам верификации предложить то, что могло бы показаться довольно убедительным сви-

детельством в поддержку их методов. Отсутствие на данный момент хотя бы одного примера верификации работающей системы иногда относят на счет молодости самой дисциплины. Верификаторы говорят, например, что они только-только начинают понимать инварианты циклов. На первый взгляд, это звучит как еще одна вариация на тему масштабирования. Но на самом деле существуют большие классы реальных систем практически вообще без циклов – они весьма редко встречаются в коммерческих приложениях. И тем не менее никогда не предпринималось попытки верифицировать, скажем, написанную на Cobol систему, которая печатает чеки; отсутствие хотя бы одной ставит под сомнение возможность верифицировать много систем хоть когда-нибудь в будущем. Ресурсы, время и энергия для верификаторов ограничены так же, как для всех остальных.

Поэтому мы должны осознать и принять две проблемы, занимающие инженеров уже много поколений: во-первых, людям приходится заниматься деятельностью, которой они не понимают; во-вторых, люди не умеют создавать совершенных механизмов.

Как же тогда инженерам удается создавать надежные конструкции? Во-первых, они применяют социальные процессы, очень похожие на те, что математики применяют для последовательного приближения к пониманию. Во-вторых, у них имеется зрелое и реалистичное представление о том, что значит «надежный»; в частности, это никогда не означает «совершенный». Не существует способа логически вывести, что мост стоит, что самолет летает или что электростанция вырабатывает электричество. Да, правда, ни один мост не упал бы, ни один самолет не потерпел бы крушения, ни одна электростанция не прекратила бы поставлять в систему ток, если бы инженеры, перед тем как приступать к строительству, доказали бы их совершенство. Правда, потому что тогда ничего и не было бы построено.

Аналогом программированию является любая функционирующая полезная реальная система. Возьмем, к примеру, синтезатор органических соединений SYNCHEM (Gelernter et al. 1973). Для этой программы критерий надежности особенно прост: если она синтезирует химическое соединение – значит, работает, если не синтезирует – значит, не работает. Сколько ни добавляй корректности, нет никакой надежды улучшить этот стандарт; не ясно даже, как можно было бы хотя бы приступить к формализации такого стандарта способом, пригодным для верификации. Но полезная и непрекращающаяся деятельность – пытаться увеличить количество химических веществ, которые программа может синтезировать.

Не что иное, как символьный шовинизм, заставляет специалистов по информатике думать, будто наши конструкции настолько важнее материальных, что (a) они должны быть совершенными и (b) на то, чтобы сделать их совершенными, можно тратить энергию в неограниченных количествах. Мы утверждаем, что (a) совершенными они быть не могут и (b) не следует расточать энергию в тщетных попытках сделать их совершенными. Вовсе не случайно вероятностный взгляд на математическую истинность тесно связан с инженерным понятием надежности. Быть может, стоит провести четкое различие между надежностью и совершенством программ – и сосредоточить усилия именно на надежности.

Стремление сделать программы корректными ценно и конструктивно. Но тоталитарный взгляд на верификацию слеп к тем преимуществам, которые могли бы проистечь из принятия стандарта корректности, похожего на стандарт корректности для реальных математических доказательств или на стандарт надежности для реальных инженерных конструкций. Погоня за работоспособностью в рамках экономических ограничений, готовность тиражировать новшества путем повторного использования успешных конструкций, доверие к функционированию сообщества равных – все эти механизмы, благодаря которым работает математика и техника, заслоняет бесплодный поиск совершенной верифицируемости.

Какие элементы могли бы помочь сделать программирование больше похожим на математику и технику? Один из возможных механизмов – создание общих конструкций, конкретные экземпляры которых становятся более надежными по мере повышения надежности общих конструкций. Эта идея встречалась в нескольких воплощениях, из которых настойчивая рекомендация Кнута создавать и понимать общеполезные алгоритмы – одна из самых важных и ободряющих. Методология командного программирования Бейкера (Baker 1972) – явная попытка подвергнуть программное обеспечение воздействию социальных процессов. Если возможность повторного использования станет критерием эффективного дизайна, то все более и более широкое сообщество сможет изучать наиболее употребительные инструменты программирования.

Концепция верифицируемого программного обеспечения живет с нами так долго, что ее трудно вытеснить. Но в практике программирования не следует позволять верифицируемости оттеснять надежность. Ученые не должны путать математические модели с реальностью, а верификация – не что иное, как модель правдоподобия. Верифицируемость не является и не может являться доминирующим фактором при проектировании программного обеспечения. Экономика, сроки, рентабельность, личный и групповой стиль, пределы допустимой ошибки – вес всего этого для проектирования неизмеримо выше, чем верифицируемость или неверифицируемость.

До сих пор мало внимания уделялось философскому обсуждению того, что важнее: сделать программу верифицируемой или надежной. Если бы поборники верификации смогли переориентировать свои усилия на достижение этой цели или если бы могла возникнуть другая точка зрения на программное обеспечение, которая опиралась бы на социальные процессы математики или на скромные ожидания техники, то это лучше послужило бы интересам программирования для реальной жизни и теоретической информатики.

Но даже если по какой-то причине, которую мы в данный момент не в состоянии понять, окажется, что мы во всем неправы, а верификаторы во всем правы, сейчас не время ограничивать исследования в области программирования. Пока мы знаем слишком мало, чтобы чувствовать, какие направления окажутся наиболее плодотворными. Если наши доводы никого не убедили, если именно верификация представляется той широкой дорогой, которую только и нужно исследовать, так тому и быть; мы втроем можем только возражать против верификации, но не стереть ее с лица земли. Но мы закли-

наем своих друзей и коллег не ограничивать свое видение только одной точкой зрения, какой бы многообещающей она ни была. Пусть это будет не единственный взгляд, не единственная дорога. Джейкоб Броновски высказывал важное соображение о моменте в истории другой дициплины, который походит на переживаемый нами момент в развитии компьютерной науки: «Наука, которая слишком рано располагает свои мысли по строгому ранжиру, задыхается. ... Надежда средневековых алхимиков на то, что элементы изменяемы, не такая химеричная, как мы когда-то думали. Но она была воистину разрушительной для химии, которая еще не понимала даже состава воды и обычной соли».

45 Метод получения цифровых подписей и криптосистемы с открытым ключом (1978)

Рональд Ривест, Ади Шамир и Лен Адлеман

Экземпляр статьи Диффи и Хеллмана (Diffie and Hellman 1976a, здесь глава 42) быстро дошел до МТИ, где Рональд Ривест (родился в 1947 году), Леонард Адлеман (родился в 1945 году) и Ади Шамир (родился в 1952 году) занялись проблемой поиска криптосистемы с открытым ключом, подходящей как для распределения ключей, так и для цифровой подписи. В конечном итоге они втроем придумали, как положить в основание такой системы задачу о разложении чисел на простые множители. Алгоритм RSA широко распространился в программном обеспечении для безопасности в интернете в конце 1990-х годов, когда началось коммерческое использование Всемирной паутины. Сегодня в любой банковской транзакции, в любой покупке через интернет, при любом вызове такси через агрегатора незримо присутствует обмен ключами между браузером или приложением пользователя и компьютером, предлагающим соответствующий сервис.

Эта статья – замечательное доказательство красоты алгоритмов и полезности даже самой что ни на есть чистой математики. Никому из студентов,

готовящих себя к карьере в области информатики в те времена, когда авторы были молоды, не пришло бы в голову изучать теорию чисел в расчете на то, что она пригодится при построении компьютерных систем, но эти трое знали предмет достаточно хорошо, чтобы провести мозговой штурм, выдвинуть идеи и исправлять заблуждения друг друга по части их использования. Алгоритм элегантен и легко описывается – если немного знать математику XVIII века, относящуюся к свойствам целых чисел. И тем не менее он опирается на недоказанное предположение: что не существует быстрого алгоритма разложения больших чисел на множители (факторизации).

О трудности факторизации уже тогда было хорошо известно. В 1801 году Карл Фридрих Гаусс заметил, что лучшие методы, придуманные «античными и современными» математиками, «испытывают терпение даже опытного вычислителя», и призывал «исследовать все возможные средства для решения этой проблемы, столь элегантной и столь прославленной» (Gauss, 1986, стр. 396f). Тем самым он предвосхитил программу, сформулированную в предпоследнем абзаце этой статьи. Уильям Джевонс, изобретатель логического пианино (стр. 54 этой книги), заметил, что факторизация, по-видимому, является одним из примеров того, что сегодня мы назвали бы односторонней функцией. «Существует много примеров, когда мы легко и безошибочно можем сделать какую-то вещь, но противоположная ей дается с большим трудом. … Если даны два числа, то с помощью простого и безошибочного процесса можно получить их произведение; но вот определить множители большого числа – совсем другое дело». (Jevons 1874, стр. 122). О том, что факторизация стоит экспоненциально дорого, говорилось часто (в т. ч. Джевонсом), но ни разу не было доказано, несмотря на рьяные усилия, последовавшие за широким распространением алгоритма RSA.

Быстрая факторизация позволила бы вскрыть криптосистему RSA, а пока система не вскрыта – насколько нам известно; правда, всегда существует возможность, что какое-то правительственное агентство или преступник разработали и держат в секрете метод факторизации больших чисел. Но по мере развития технологий длины ключей, предложенные в статье, на практике были увеличены. Хотя были разработаны системы с открытым ключом, основанные на других математических объектах (например, эллиптических кривых), их безопасность на сегодняшний день тоже не доказана.

Хотя авторы предлагают читателям, знакомым с работой Диффи и Хеллмана (Diffie and Hellman 1976a), пропустить первые несколько разделов статьи, мы включили их, потому что в них определен контекст и, главное, введены персонажи «Алиса» и «Боб», которые в последовавшей затем литературе играют роли сообщающихся сторон.

Ривест по сей день работает в МТИ, Адлеман – в университете Южной Калифорнии, а Шамир – в израильском институте Вейцмана. Все трое внесли значительный вклад и в другие разделы информатики (см., в частности, главу 46). В 1983 году они основали компанию для коммерциализации открытия, представленного в этой статье (RSA Security, позже приобретенную Security Dynamics, которая, в свою очередь, была приобретена EMC, а та приобретена Dell). В 2002 году они все вместе были удостоены премии Тьюринга.

45.0. Краткое описание

Представлен метод шифрования, отличающийся от прежних тем, что публичное раскрытие ключа шифрования не приводит к раскрытию ключа расшифрирования. Отсюда вытекают два важных следствия.

1. Ни курьеры, ни другие средства обеспечения безопасности не нужны для передачи ключей, потому что сообщение можно зашифровать ключом, который предполагаемый получатель может публично раскрыть. Только он может расшифрировать сообщение, потому что только ему известен соответствующий ключ расшифрирования.
2. Сообщение может быть «подписано» с помощью хранящего в секрете ключа расшифрирования. Любой желающий может проверить эту подпись, воспользовавшись соответствующим открытым ключом шифрования. Подписи невозможно подделать, а подписавший не сможет впоследствии отрицать подлинность своей подписи. У этого факта имеются очевидные применения в системах «электронной почты» и «электронного перевода средств». Чтобы зашифровать сообщение, оно представляется числом M, это число возводится в несекретную степень e, а затем берется остаток от деления результата на несекретное произведение n двух секретных больших простых чисел p и q. Расшифрирование производится аналогично, только используется другая, уже секретная, степень d, где $e \cdot d \equiv 1 \pmod{(p-1)\cdot(q-1)}$. Безопасность системы основана частично на трудности разложения опубликованного числа n на множители.

45.1. Введение

На нас надвигается эра «электронной почты» (Potter 1977); мы должны быть уверены, что будут сохранены два важных свойства современной «бумажной почты»: (a) сообщения конфиденциальны и (b) сообщения можно подписать. В этой работе мы продемонстрируем, как можно встроить эти свойства в систему электронной почты.

В основе нашего предложения лежит новый метод шифрования. Этот метод обеспечивает реализацию «криптосистемы с открытым ключом», элегантной концепции, придуманной Диффи и Хеллманом (1976a, здесь глава 42). Их статья подвигла нас на это исследование, поскольку они предложили лишь концепцию, но не предложили никакой практической реализации такой системы. Читатели, знакомые с работой (Diffie and Hellman 1976a), могут перейти прямо к § 45.5, где описывается наш метод.

45.2. Криптосистемы с открытым ключом

В «криптосистеме с открытым ключом» каждый пользователь помещает в открытый общедоступный файл процедуру шифрования E. То есть открытый файл – это каталог, который предоставляет процедуры шифрования всех пользователей. Пользователь хранит в секрете детали соответствующей процедуры расшифрирования D. Эти процедуры обладают следующими четырьмя свойствами:

(a) расшифрирование зашифрованной формы сообщения M дает M. Формально

$$D(E(M)) = M; \tag{45.1}$$

(b) E и D легко вычислить;
(c) раскрывая E, пользователь не раскрывает простого способа вычислить D. На практике это значит, что только он может эффективно расшифрировать сообщения, зашифрованные E, или вычислить D;
(d) если сообщение M сначала расшифрировать, а затем зашифровать, то в результате получится M. Формально

$$E(D(M)) = M. \tag{45.2}$$

Процедура шифрования (и расшифрирования) обычно состоит из *общего метода* и *ключа шифрования*. Общий метод, под контролем ключа, применяется для шифрования сообщения M с целью получить его зашифрованную форму, называемую *шифртекстом* C. Любой может воспользоваться тем же самым общим методом; безопасность данной процедуры зависит от безопасности ключа. То есть раскрытие алгоритма шифрования означает раскрытие ключа.

Зная только E, пользователь получает очень *неэффективный* метод вычисления $D(C)$: проверять все возможные сообщения M, пока не встретится такое, для которого $E(M) = C$. Если свойство (c) выполнено, то количество возможных сообщений столь велико, что этот подход практически бесполезен.

Функция E, обладающая свойствами (a)–(c), называется «односторонней функцией с закладкой»[1]; если она также обладает свойством (d), то называется «односторонней перестановкой с закладкой». В работе (Diffie and Hellman 1976a, здесь глава 42) введено понятие односторонних функций с закладкой, но не представлено никаких примеров. Эти функции называются «односторонними», потому что их легко вычислить в одном направлении, но (на первый взгляд) очень трудно – в противоположном. Слово «закладка» означает, что на самом деле обратную функцию вычислить легко, если известна

[1] В литературе (см., например, «Словарь криптографических терминов» под ред. Б. А. Погорелова и В. Н. Сачкова, М.: МЦНМО, 2006) рекомендуется перевод «с секретом», а не «с закладкой». Но, на наш взгляд, слово «секрет» в этом контексте и так перегружено, что чревато недоразумениями. – *Прим. перев.*

секретная информация о «закладке». Односторонняя функция с закладкой, удовлетворяющая условию (d), обязана быть перестановкой: любое сообщение является шифртекстом какого-то другого сообщения, и любой шифртекст сам является допустимым сообщением (т. е. отображение «взаимно однозначно» и «на»). Свойство (d) необходимо для реализации «подписей».

Читателю рекомендуется прочитать великолепную статью Диффи и Хеллмана, где имеются предварительные сведения, уточнение концепции криптосистемы с открытым ключом и обсуждение других проблем в области криптографии. Способы, с помощью которых криптосистема с открытым ключом может обеспечить конфиденциальность и «подписи» (описаны в § 45.3 и 45.4 ниже), также предложены Диффи и Хеллманом.

В наших сценариях мы будем предполагать, что A и B (они же Алиса и Боб) – два пользователя криптосистемы с открытым ключом. Будем различать их процедуры шифрования и расшифрирования по индексам: E_A, D_A, E_B, D_B.

45.3. Конфиденциальность

Шифрование – стандартный способ сделать переписку конфиденциальной. Отправитель шифрует каждое сообщение перед отправкой его получателю. Получатель (но не какое-то другое, не авторизованное лицо) знает подходящую функцию расшифрирования, которую нужно применить к полученному сообщению, чтобы получить исходное. Пассивный противник, который прослушивает передаваемое сообщение, слышит только «мусор» (шифртекст), не имеющий никакого смысла, потому что он не знает, как его расшифровать.

Из-за большого объема персональной и секретной информации, хранящейся ныне в компьютеризованных банках данных и передаваемой по телефонным линиям, шифрование становится все более важным. Признавая тот факт, что эффективные высококачественные методы шифрования крайне необходимы, но их предложение явно недостаточно, Национальное бюро стандартов (НБС) недавно приняло «Стандарт шифрования данных», разработанный компанией IBM (федеральный реестр: том 40, № 42, 17 марта 1975; том 40, № 149, 1 августа 1975). Новый стандарт не обладает свойством (c), необходимым для реализации криптосистемы с открытым ключом.

Все классические методы шифрования (включая стандарт НБС) испытывают «проблему распределения ключей». Проблема состоит в том, что до начала взаимодействия должна состояться другая закрытая операция по вручению ключей шифрования и расшифрирования отправителю и получателю соответственно. Обычно снаряжается курьер, который доставляет ключ от отправителя получателю. Такая практика не годится, если мы хотим, чтобы система электронной почты была быстрой и недорогой. В криптосистеме с открытым ключом частные курьеры не нужны; ключи можно распределять по небезопасным каналам связи.

Как Боб может отправить конфиденциальное сообщение M Алисе в криптосистеме с открытым ключом? Сначала он извлекает E_A из открытого файла. Затем он отправляет ей зашифрованное сообщение $E_A(M)$. Алиса расшифри-

рует сообщение, вычисляя $D_A(E_A(M)) = M$. В силу свойства (c) криптосистемы с открытым ключом, только она может расшифрировать $E_A(M)$. Она может зашифровать конфиденциальный ответ процедурой E_B, также имеющейся в открытом файле.

Заметим, что для установления конфиденциальной связи между Алисой и Бобом не нужны никакие конфиденциальные операции. Единственная необходимая «подготовка» заключается в том, что каждый пользователь, желающий получать конфиденциальные сообщения, должен поместить свой алгоритм шифрования в открытый файл.

Два пользователя могут также организовать конфиденциальную переписку по небезопасному каналу связи, не обращаясь к открытому файлу. Каждый пользователь отправляет другому свой ключ шифрования. После этого все сообщения шифруются ключом шифрования получателя, как в системе с открытым ключом. Злоумышленник, прослушивающий канал, не сможет расшифрировать сообщения, потому что не способен вывести ключи расшифрирования из ключей шифрования. (Мы предполагаем, что злоумышленник не может ни модифицировать сообщения, ни вставлять новые сообщения в канал.) Ральф Меркл (Merkle 1978) разработал другое решение этой проблемы.

Криптосистему с открытым ключом можно использовать в качестве «начальной ступени» входа в стандартную схему шифрования, например с помощью метода НБС. После того как безопасная связь установлена, первое передаваемое сообщение может быть ключом в схеме НБС, которым будут шифроваться все последующие сообщения. Это может оказаться желательным, если шифрование нашим методом медленнее, чем по стандартной схеме. (Схема НБС, вероятно, будет несколько быстрее при использовании специального шифровального оборудования; наша схема, возможно, будет работать быстрее на универсальном компьютере, потому что арифметические операции с многократной точностью проще реализовать, чем сложные поразрядные операции.)

45.4. Подписи

Если системам электронной почты суждено заменить существующую систему бумажной почты в деловой корреспонденции, то должно быть возможно «подписание» электронного сообщения. У получателя подписанного сообщения есть доказательство того, что это сообщение исходит от отправителя. Это качество сильнее простой аутентификации (когда получатель может проверить, что сообщение поступило от отправителя); получатель может убедить «судью», что подписавший действительно отправлял сообщение. Для этого он должен убедить судью, что не подделал подписанное сообщение сам! В проблеме аутентификации получателя не волнует эта возможность, потому что он лишь хочет убедиться, что сообщение пришло от отправителя.

Электронная подпись должна зависеть от сообщения и от подписавшего. В противном случае получатель мог бы модифицировать сообщение до предъявления пары сообщение–подпись судье. Или он мог бы присоединить

подпись вообще к любому сообщению, поскольку нет никакой возможности обнаружить электронное «вырезание и вклеивание».

Чтобы можно было реализовать подписи, криптосистема с открытым ключом должна быть реализована с помощью односторонней перестановки с закладкой (т. е. обладать свойством (d)), поскольку алгоритм расшифрирования будет применяться к незашифрованным сообщениям.

Как Боб может отправить Алисе «подписанное» сообщение M в криптосистеме с открытым ключом? Сначала он вычисляет свою «подпись» S для сообщения M, пользуясь процедурой D_B: $S = D_B(M)$. (Расшифрирование незашифрованного сообщения «имеет смысл» в силу свойства (d) криптосистемы с открытым ключом: каждое сообщение является шифртекстом какого-то другого сообщения.) Затем он шифрует S процедурой E_A (ради конфиденциальности) и отправляет результат $E_A(S)$ Алисе. Ему не нужно отправлять также M; его можно вычислить по S.

Алиса сначала расшифрирует шифртекст процедурой D_A для получения S. Она знает, кто предполагаемый отправитель подписи (в данном случае Боб); при необходимости эту информацию можно присоединить к S в открытом виде. Затем она извлекает сообщение, пользуясь процедурой шифрования отправителя, в данном случае E_B (она имеется в открытом файле): $M = E_B(S)$. Теперь у нее есть пара сообщение–подпись (M, S) со свойствами, аналогичными свойствам подписанного бумажного документа.

Боб не сможет впоследствии отрицать, что отправил Алисе это сообщение, потому что никто, кроме него, не смог бы создать $S = D_B(M)$. Алиса может убедить «судью», что $E_B(S) = M$, поэтому у нее есть доказательство того, что документ подписал именно Боб.

Очевидно, Алиса не может изменить M, создав другую версию M', потому что тогда ей пришлось бы создать также и соответствующую подпись $S' = D_B(M')$.

Таким образом, Алиса получила сообщение, «подписанное» Бобом, и может доказать, что именно он его отправил, но модифицировать это сообщение она не может. (Как не может и подделать его подпись под любым другим сообщением.)

Электронная чековая система могла бы быть основана на подобной системе подписи. Легко представить себе шифровальное устройство в домашнем терминале, позволяющее вам подписывать чеки, отправляемые по электронной почте получателю денег. Нужно было бы только включать уникальный номер чека в каждый чек, тогда даже в том случае, когда получатель делает копию чека, банк примет только первую увиденную версию.

Еще одна возможность возникает, если шифровальные устройства удастся сделать достаточно быстрыми: можно будет организовать телефонный разговор, в котором каждое слово еще до передачи подписывается шифровальным устройством.

Когда шифрование используется для подписания, как описано выше, важно, чтобы шифровальное устройство не встраивалось «намертво» между терминалом (или компьютером) и каналом связи, потому что не исключено, что сообщение нужно будет последовательно зашифровать несколькими ключами. Быть может, более естественно рассматривать шифровальное устройство как «аппаратную подпрограмму», которую можно выполнять по мере

необходимости. Выше мы предполагали, что каждый пользователь в любой момент может надежно обратиться к открытому файлу. В «компьютерной сети» это может оказаться трудно; «злоумышленник» может подделывать сообщения, якобы взятые из открытого файла. Пользователь хотел бы иметь уверенность в том, что он действительно получает процедуру шифрования своего предполагаемого корреспондента, а не, скажем, процедуру шифрования злоумышленника. Эта опасность исчезает, если открытый файл «подписывает» каждое сообщение перед отправкой пользователю. Пользователь может проверить подпись, воспользовавшись алгоритмом шифрования E_{PF} открытого файла. Проблемы «поиска» самого E_{PF} в открытом файле можно избежать, передавая каждому пользователю описание E_{PF}, когда он в первый раз приходит (лично), чтобы присоединиться к криптосистеме с открытым ключом и разместить свою открытую процедуру шифрования. Затем он сохраняет это описание и больше никогда его не ищет. Таким образом, необходимость отправлять курьера между каждой парой пользователей заменена требованием однократной безопасной встречи между каждым пользователем и администратором открытого файла в момент, когда пользователь присоединяется к системе. Еще одно решение – выдавать каждому пользователю в момент подписания соглашения книгу (аналог телефонного справочника), в которой содержатся ключи шифрования всех зарегистрированных в системе пользователей.

45.5. Наши методы шифрования и расшифрирования

Чтобы зашифровать сообщение M нашим методом с использованием открытого ключа шифрования (e, n), нужно действовать следующим образом (здесь e и n – положительные целые числа).

Сначала представить сообщение целым числом от 0 до $n - 1$. (Длинное сообщение разбить на блоки и представить таким числом каждый блок.) Можно использовать любое стандартное представление. Цель здесь – не зашифровать сообщение, а лишь представить его в числовой форме, пригодной для шифрования.

Затем зашифровать сообщение, возведя его в степень e по модулю n. То есть результатом (шифртекстом C) является остаток от деления M^e на n.

Чтобы расшифрировать сообщение, возвести его в другую степень d, снова по модулю n. Таким образом, алгоритмы шифрования и расшифрирования E и D имеют вид:

$$C \equiv E(M) \equiv M^e \pmod{n} \text{ для сообщения } M;$$
$$D(C) \equiv C^d \pmod{n} \text{ для шифртекста } C.$$

Отметим, что шифрование не увеличивает размера сообщения; как сообщение, так и шифртекст – целые числа от 0 до $n - 1$.

Итак, *ключ шифрования* – пара положительных целых чисел (e, n). Аналогично ключ расшифрирования – пара положительных целых чисел (d, n). Каж-

дый пользователь делает свой ключ шифрования открытым, а соответствующий ключ расшифрирования хранит в секрете. (Эти целые числа следовало бы снабжать индексами – n_A, e_A и d_A, т. к. у каждого пользователя имеется свой собственный набор. Но поскольку мы рассматриваем типичный набор, эти индексы будем опускать.)

Как выбирать ключи шифрования и расшифрирования, чтобы воспользоваться нашим методом? Сначала нужно вычислить n, равное произведению двух простых чисел p и q: $n = p \cdot q$. Это очень большие «случайные» простые числа. Хотя n будет открыто, множители p и q, по сути дела, скрыты от всех, потому что факторизация n представляет огромные трудности. Тем самым скрывается также способ вывода d из e.

После этого мы случайным образом выбираем большое целое d, взаимно простое с $(p-1)\cdot(q-1)$, т. е. такое, что $\gcd(d, (p-1)\cdot(q-1)) = 1$ («gcd» означает «greatest common divisor» – наибольший общий делитель).

Наконец, по p, q и d вычисляется целое число e, являющееся «мультипликативным обращением» d по модулю $(p-1)\cdot(q-1)$. Таким образом, мы имеем $e \cdot d \equiv 1 \pmod{(p-1)\cdot(q-1)}$.

В следующем разделе мы докажем, что это гарантирует выполнение равенств (45.1) и (45.2), т. е. что E и D – взаимно обратные преобразования. В § 45.7 показано, как каждую из перечисленных выше операций можно выполнить эффективно.

Описанный метод не следует путать с техникой «возведения в степень» из работы (Diffie and Hellman 1976a, здесь глава 42) для решения проблемы распределения ключей. Предложенная ими техника позволяет двум пользователям выработать общий ключ для использования в обычной криптографической системе. Она не основана на односторонней перестановке с закладкой. В работе (Pohlig and Hellman 1978) изучается схема, родственная нашей, где возведение в степень производится по модулю простого числа.

45.6. Математические основы

Мы докажем правильность алгоритма расшифрирования, воспользовавшись тождеством, открытым Эйлером и Ферма (Niven 1972): для любого целого числа (сообщения) M, взаимно простого с n:

$$M^{\phi(n)} \equiv 1 \pmod{n}. \tag{45.3}$$

Здесь $\phi(n)$ – функция Эйлера, равная количеству положительных целых чисел, меньших n и взаимно простых с n. Для простых чисел p $\phi(p) = p - 1$. Для нашего случая справедливы следующие элементарные свойства функции Эйлера:

$$\begin{aligned}\phi(n) &= \phi(p)\cdot\phi(q)\\ &= (p-1)\cdot(q-1)\\ &= n - (p+q) + 1.\end{aligned}$$

Поскольку d взаимно просто с $\phi(n)$, у него есть мультипликативно обрат-

ный элемент e в кольце целых чисел по модулю $\phi(n)$:

$$e \cdot d \equiv 1 (\mathrm{mod}\ \phi(n)). \tag{45.4}$$

Теперь докажем, что имеют место равенства (45.1) и (45.2) (т. е. что расшифрирование работает правильно, если e и d выбраны, как описано выше). Имеем

$$D(E(M)) \equiv (E(M))^d \equiv (M^e)^d \equiv M^{e \cdot d} (\mathrm{mod}\ n);$$
$$E(D(M)) \equiv (D(M))^e \equiv (M^d)^e \equiv M^{e \cdot d} (\mathrm{mod}\ n)$$

и

$$M^{e \cdot d} \equiv M^{k \cdot \phi(n)+1} (\mathrm{mod}\ n) \text{ (для некоторого целого } k).$$

Из (45.3) видно, что для всех M таких, что p не делит M, $M^{p-1} \equiv 1 \pmod{p}$ и, т. к. $p - 1$ делит $\phi(n)$,

$$M^{k \cdot \phi(n)+1} \equiv M (\mathrm{mod}\ p).$$

Это очевидно, если $M \equiv 0 \pmod{p}$, поэтому равенство имеет место для всех M. Аналогичное рассуждение для q дает

$$M^{k \cdot \phi(n)+1} \equiv M (\mathrm{mod}\ q).$$

Их двух последних сравнений следует, что для всех M

$$M^{e \cdot d} \equiv M^{k \cdot \phi(n)+1} \equiv M (\mathrm{mod}\ n).$$

Отсюда следует справедливость (45.1) и (45.2) для всех M, $0 \leqslant M < n$. Таким образом, E и D – взаимно обратные перестановки. (Мы благодарны Ричу Шреппелю за этот улучшенный вариант первого доказательства авторов.)

45.7. Алгоритмы

Чтобы продемонстрировать практическую применимость нашего метода, мы опишем эффективные алгоритмы всех требуемых операций.

45.7.1. Как эффективно выполнять шифрование и расшифрирование. Вычисление $M^e (\mathrm{mod}\ n)$ требует не более $2\log_2(e)$ умножений и $2\log_2(e)$ делений, если использовать следующую процедуру (расшифрирование можно выполнить аналогично, используя d вместо e):

1. Пусть $e_k e_{k-1} \ldots e_1 e_0$ – двоичное представление e.
2. Положить переменную C равной 1.
3. Повторять шаги 3a и 3b для $i = k, k - 1, \ldots, 0$.

3a. Положить C равной остатку от деления C^2 на n.

3b. Если $e_i = 1$, то положить C равной остатку от деления $C \cdot M$ на n.

4. Остановиться. Теперь C – зашифрованная форма M.

Эта процедура называется «возведением в степень путем повторного возведения в квадрат и умножения». Она в два раза хуже наилучшей из известных процедур. В работе Knuth (1969) эта проблема рассматривается во всех подробностях.

Поскольку шифрование и расшифрирование производятся одинаково, реализация оказывается очень простой. (Всю операцию можно реализовать с помощью нескольких специализированных интегральных схем.)

Высокоскоростной компьютер может зашифровать 200-значное сообщение M за несколько секунд, специализированное оборудование – намного быстрее. Время шифрования блока растет не быстрее, чем куб числа цифр в n.

45.7.2. Как найти большие простые числа. Каждый пользователь должен (по секрету от других) выбрать два больших случайных простых числа p и q для создания своих ключей шифрования и расшифрирования. Эти числа должны быть настолько большими, чтобы нахождение разложения на множители $n = p \cdot q$ было вычислительно неразрешимой задачей. (Напомним, что именно n, а не p и не q будет находиться в открытом файле.) Мы рекомендуем использовать простые p и q со 100 десятичными знаками, чтобы n было 200-значным.

Чтобы найти 100-значное «случайное» простое число, будем генерировать (нечетные) 100-значные числа, пока не встретится простое число. По теореме о простых числах (Niven 1972), придется проверить приблизительно $(\ln 10^{100})/2 = 115$ чисел, прежде чем будет найдено простое.

Для проверки большого числа b на простоту мы рекомендуем элегантный «вероятностный» алгоритм из работы Solovay and Strassen (1977). Он выбирает случайное число из равномерного распределения на отрезке $[1, \ldots b - 1]$ и проверяет, верно ли, что

$$\gcd(a, b) = 1 \text{ и } J(a, b) \equiv a^{(b-1)/2} (\mathrm{mod}\ b), \tag{45.5}$$

где $J(a, b)$ – символ Якоби (Niven 1972). Если b простое, то (45.5) заведомо истинно. Если b составное, то (45.5) ложно с вероятностью не менее ½. Если (45.5) справедливо для 100 случайно выбранных a, то b почти наверняка простое; существует (пренебрежимо малый) шанс 1 из 2^{100}, что b будет составным. Даже если в нашей системе по случайности было использовано составное число, получатель, скорее всего, заметит это, поскольку расшифрирование будет работать неверно. Если b нечетно, $a \leqslant b$ и $\gcd(a, b) = 1$, то символ Якоби $J(a, b)$ принимает значение из отрезка $[-1, 1]$ и может быть эффективно вычислен следующей программой:

$$\begin{aligned} J(a, b) = {} & \textbf{if } a = 1 \textbf{ then } 1 \textbf{ else} \\ & \textbf{if } a \text{ четно } \textbf{then } J(a/2, b) \cdot (-1)^{(b^2-1)/8} \\ & \textbf{else } J(b \bmod a, a) \cdot (-1)^{(a-1)\cdot(b-1)/4} \end{aligned}$$

(Кстати, вычисления $J(a, b)$ и $\gcd(a, b)$ можно элегантно совместить.) Заметим, что в этом алгоритме для проверки простоты числа не делается попытка разложить его на множители. Другие эффективные процедуры проверки больших чисел на простоту приведены в работах Miller (1976), Pollard (1974), Rabin (1976).

Для дополнительной защиты от изощренных алгоритмов факторизации длины чисел p и q должны отличаться на несколько цифр; как $p-1$, так и $q-1$ должны содержать большие простые множители и $\gcd(p-1, q-1)$ должно быть мало. Последнее условие легко проверяется.

Чтобы найти простое число p такое, что $p-1$ имеет большой простой множитель, сгенерируем большое случайное простое число u, и пусть p – первое простое число в последовательности $i \cdot u + 1$ для $i = 2, 4, 6, \ldots$. (Нахождение p не должно занять много времени.) Чтобы еще усилить защиту, потребуем, чтобы $(u-1)$ также имело большой простой множитель.

Высокоскоростной компьютер может за несколько секунд определить, является ли 100-значное число простым, а первое простое число после заданного найдет за минуту или две.

Другой подход к отысканию больших простых чисел заключается в том, чтобы взять число с известной факторизацией, прибавить к нему 1 и проверить результат на простоту. Если предположительно простое число p найдено, то доказать, что оно действительно простое, можно, воспользовавшись факторизацией $p-1$. Мы опускаем обсуждение этого вопроса, потому что вероятностного метода вполне достаточно.

45.7.3. Как выбрать *d*. Очень легко выбрать число d, взаимно простое с $\phi(n)$. Например, подойдет любое простое число, большее, чем $\max(p, q)$. Важно, чтобы d выбиралось из достаточно большого множества, так чтобы криптоаналитик не мог найти его прямым перебором.

45.7.4. Как вычислить *e* по *d* и $\phi(n)$. Для вычисления e используется следующий вариант алгоритма Евклида для вычисления наибольшего общего делителя $\phi(n)$ и d (см. упражнение 4.5.2.15 в книге Knuth [1969]). Найти $\gcd(\phi(n), d)$ путем вычисления последовательности $x_0, x_1, x_2, \ldots$, где $x_0 = \phi(n)$, $x_1 = d$ и $x_{i+1} = x_{i-1} \pmod{x_i}$, пока не встретится x_k, равное 0. Тогда $\gcd(x_0, x_1) = x_{k-1}$. Для каждого x_i вычислить числа a_i и b_i такие, что $x_i = a_i \cdot x_0 + b_i \cdot x_1$. Если $x_{k-1} = 1$, то b_{k-1} – мультипликативное обращение $x_1 \pmod{x_0}$. Так как k будет меньше $2 \cdot \log_2(n)$, то это вычисление завершается очень быстро.

Если e оказалось меньше $\log_2(n)$, повторить с другим значением d. Тем самым гарантируется, что любое зашифрованное сообщение (кроме $M = 0$ и $M = 1$) хотя бы раз «обернулось» (т. е. выполнялось деление по модулю n). <...>

45.8. Небольшой пример

<...>

45.9. Стойкость метода: подходы к криптоанализу

Поскольку не существует никакого метода, позволяющего доказать, что некая схема шифрования безопасна, единственный доступный нам критерий –

посмотреть, не придумает ли кто-нибудь способы вскрыть ее. Стандарт НБС был «сертифицирован» именно таким образом; IBM потратила 17 человеколет, тщетно пытаясь вскрыть эту схему. Коль скоро метод успешно устоял против сосредоточенной атаки, он может считаться стойким для практических целей. (На самом деле по поводу стойкости метода НБС есть некоторые сомнения, см. [Diffie and Hellman, 1977].)

В следующем разделе мы покажем, что все очевидные подходы к вскрытию нашей системы по меньшей мере так же трудны, как факторизация n. Хотя трудность факторизации больших чисел не доказана, эта задача хорошо известна, и над ней работало много знаменитых математиков в течение последних трехсот лет. Ферма (1601?–1665) и Лежандр (1752–1833) разработали алгоритмы факторизации; некоторые из современных, более эффективных алгоритмов основаны на идеях Лежандра. Но, как мы увидим в следующем разделе, никому еще не удалось найти алгоритм, который мог бы факторизовать 200-значное число за разумное время. Отсюда мы делаем вывод, что наша система уже частично «сертифицирована» благодаря предшествующим усилиям найти эффективные алгоритмы факторизации.

В следующих разделах мы рассмотрим, какими способами криптоаналитик мог бы попытаться найти секретный ключ расшифрования по открытому ключу шифрования. Мы не рассматриваем способы защиты ключа от кражи, должно хватить обычных методов обеспечения физической безопасности. (Например, шифровальное устройство могло бы быть автономным устройством, которое используется также для генерирования ключей шифрования и расшифрования таким образом, что ключ расшифрования никогда не печатается (даже для своего владельца), а только используется для расшифрования сообщений. Устройство могло бы также стирать ключ расшифрования при попытке вскрытия.)

45.9.1. Факторизация *n*. Успешная факторизация n позволила бы противнику «вскрыть» наш метод. Зная множители n, он смог бы вычислить $\phi(n)$, а значит, и d. По счастью, факторизация числа представляется гораздо более трудной задачей, чем определение того, является оно простым или составным. <...> [Примечание редактора: обсуждение алгоритмов факторизации опущено.]

45.9.2. Вычисление $\phi(n)$ без факторизации *n*. Если бы криптоаналитик мог вычислить $\phi(n)$, то он мог бы вскрыть систему, вычислив d как мультипликативное обращение e по модулю $\phi(n)$ (применив процедуру из § 45.7.4).

Мы утверждаем, что этот подход ничуть не проще, чем факторизация n, потому что он позволяет криптоаналитику легко факторизовать n, воспользовавшись знанием $\phi(n)$. Такой подход к факторизации n оказался практически непригодным.

Как можно факторизовать n, зная $\phi(n)$? Во-первых, $p + q$ получается из n и $\phi(n) = n - (p + q) + 1$. Затем $p - q$ является квадратным корнем из $(p + q)^2 - 4n$. Наконец, q равно полуразности $p + q$ и $p - q$.

Таким образом, вскрыть нашу систему путем вычисления $\phi(n)$ не проще, чем вскрыть ее путем факторизации n. (Именно поэтому n должно быть составным; $\phi(n)$ вычисляется тривиально, если n простое.)

45.9.3. Определить d без факторизации n или вычисления $\phi(n)$. Конечно, d следует выбирать из достаточно большого множества, чтобы прямой перебор был бесперспективен.

Мы утверждаем, что вычислить d криптоаналитику не проще, чем факторизовать n, поскольку, коль скоро d известно, выполнить факторизацию n нетрудно. Этот подход к факторизации также оказался бесплодным.

Знание d позволяет факторизовать n следующим образом. Зная d, криптоаналитик может вычислить число $e \cdot d - 1$, которое является кратным $\phi(n)$. Миллер показал, что n можно факторизовать, зная любое кратное $\phi(n)$. Поэтому если n велико, то у криптоаналитика не должно быть возможности определить d с большей легкостью, чем факторизовать n.

Криптоаналитик может надеяться найти d', которое было бы эквивалентно d, хранящемуся в секрете пользователем криптосистемы с открытым ключом. Если бы такие значения d' были широко распространены, то систему можно было бы вскрыть полным перебором. Однако все такие d' отличаются на наименьшее общее кратное $p - 1$ и $q - 1$, а нахождение хотя бы одного позволит нам факторизовать n. (В сравнениях (45.3) и (45.4) $\phi(n)$ можно заменить на $\mathrm{lcm}(p - 1, q - 1)$[1].) Поэтому нахождение любого такого d' так же трудно, как факторизация n.

45.9.4. Вычисление d каким-то другим способом. Хотя эта проблема «вычисления корней степени e по модулю n без факторизации n» не является столь же хорошо известной трудной задачей, как факторизация, у нас довольно стойкое ощущение, что она вычислительно неразрешима. Возможно, кто-то сумеет доказать, что любой общий метод вскрытия нашей схемы дает эффективный алгоритм факторизации. Это позволило бы утверждать, что любой способ вскрытия нашей схемы должен быть таким же трудным, как факторизация. Нам, однако, эту гипотезу доказать не удалось.

Наш метод должен быть сертифицирован, для чего нужно подвергнуть эту гипотезу о неразрешимости сосредоточенной атаке и убедиться, что она не принесла успеха. Читателю предлагается найти способ «вскрыть» наш метод.

45.10. Избегание «переразбиения на блоки» при шифровании подписанного сообщения

Может понадобиться «переразбить» сообщение на блоки для шифрования, потому что n подписи может быть больше, чем n шифрования (у каждого пользователя свое собственное n). Этого можно избежать следующим образом. Выберем пороговую величину h (скажем, $h = 10^{199}$) для криптосистемы с открытым ключом. Каждый пользователь хранит две открытые пары (e, n), одну для шифрования, другую для проверки подписи, причем каждое n подписи меньше h, а каждое n шифрования больше h. Тогда переразбивать

[1] lcm – наименьшее общее кратное (НОК). – *Прим. перев.*

подписанное сообщение на блоки для шифрования необязательно; сообщение разбито на блоки в соответствии с n подписи отправителя.

Еще в одном решении используется техника, описанная в работе (Levine and Brawley 1977). У каждого пользователя имеется единственная пара (e, n), где n – число от h до $2h$, а h – описанный выше порог. Сообщение кодируется числом, меньшим h, как и раньше, с тем отличием, что если шифртекст больше h, то оно перешифровывается, пока он не станет меньше h. Точно так же расшифрирование шифртекста производится повторно, пока не получится значение, меньшее h. Если n близко к h, то перешифрование будет нечастым. (Зацикливание невозможно, потому что в худшем случае результатом шифрования сообщения является оно само.)

45.11. Выводы

Мы предложили метод реализации криптосистемы с открытым ключом, стойкость которого опирается частично на трудность факторизации больших чисел. Если будет доказано, что стойкость нашего метода приемлема, то он позволит организовывать безопасную связь без использования курьеров для транспортировки ключей, а также «подписывать» оцифрованные документы.

Стойкость этой системы необходимо исследовать более детально. В частности, нужно очень тщательно изучить трудность факторизации больших чисел. Читателю настоятельно рекомендуется попытаться «вскрыть» систему. Если метод устоит против всех атак в течение достаточно длительного времени, то его можно будет использовать с разумно обоснованной степенью уверенности.

Наша функция шифрования – единственный известный авторам кандидат на роль «односторонней перестановки с закладкой». Было бы желательно найти другие примеры, чтобы можно было предложить альтернативные реализации на случай, если в один прекрасный момент окажется, что стойкость нашей системы неадекватна. И наверняка найдется много новых приложений для таких функций.

46 Как разделить секрет (1979)

Ади Шамир

По мере того как компьютеры объединяются в сети, а информация передается и распространяется, у вопросов секретности и конфиденциальности стали проявляться такие аспекты, которые не имели значения в мире бумажных документов и централизованных банков данных. Шутливые метафоры – эта работа основана на задаче из книги по математике, изданной в 1968 году, – приобрела серьезное значение в мире, опутанном информационными сетями. Успехи криптографии (главы 42 и 45) вселили надежду (математически еще не доказанную), что обыкновенные люди смогут конфиденциально общаться на расстоянии, будучи уверены, что их секреты не могут быть скомпрометированы без приложения нереалистично больших вычислительных усилий. В этой статье, напротив, описывается протокол разделения секрета между сторонами, которые должны, в точно оговоренной степени, сотрудничать, чтобы восстановить его, – но так, что ни для какой меньшей группы злоумышленников будет *доказуемо невозможно* раскрыть секрет, даже если они обладают неограниченными ресурсами. Новая дисциплина – разделение секретов, – которой положила начало эта коротенькая статья, теперь разрослась до многих тысяч работ.

В главе 45 мы встречались с Ади Шамиром (родился в 1952 году) как с одним из авторов криптосистемы с открытым ключом RSA. Эта работа начинается постановкой задачи, которая на первый взгляд неразрешима, а затем приводится красивое и короткое решение, в котором не используется ничего, кроме школьной математики. По существу, это проблема из области распределенных вычислительных систем, но о компьютерных технологиях в ней нет ни слова. На 1980-е годы пришлось взрывное развитие информатики, и в этой области по сей день остается немало просто формулируемых проблем, с нетерпением ожидающих элегантного решения.

46.1. Введение

В работе Liu (1968) рассматривается следующая задача: одиннадцать ученых работают над секретным проектом. Они хотят запереть документы в шкафу, так чтобы шкаф можно было открыть тогда и только тогда, когда присутствует шесть или более ученых. Какое наименьшее число замков необходимо для этого? Какое наименьшее число ключей должен носить с собой каждый ученый?

Нетрудно показать, что в минимальном решении используется 462 замка и 252 ключа у каждого ученого. Очевидно, что такие числа практически неприемлемы, а при увеличении количества ученых они еще и растут экспоненциально. В этой работе мы обобщим задачу: будем считать, что секрет – это некоторые данные D (например, комбинация цифр на сейфовом замке), и разрешены также немеханические решения (т. е. манипулирование данными). Наша цель – разбить D на n частей $D_1, \ldots, D_n$ таким образом, что:

1. Знание любых k или более из частей D_i позволяет легко вычислить D.
2. Знание любых $k - 1$ или менее из частей D_i не дает никакой информации о D (в том смысле, что все возможные значения равновероятны).

Такая схема называется (k, n)-*пороговой*. Эффективные пороговые схемы могут сильно помочь в управлении криптографическими ключами. Чтобы защитить данные, мы можем зашифровать их, но для защиты самого ключа шифрования необходим какой-то другой метод (последующие шифрования лишь изменяют задачу, но не решают ее). Самые безопасные схемы управления ключами предусматривают хранение ключа в одном хорошо охраняемом месте (в компьютере, в голове человека или в сейфе). Эта схема в высшей степени ненадежна, потому что достаточно единственного несчастливого стечения обстоятельств (взлом компьютера, внезапная смерть или диверсия), чтобы информация стала недоступной. Очевидное решение – хранить несколько копий ключа в разных местах, но это увеличивает риск нарушения системы безопасности (проникновение в компьютер, предательство или человеческие ошибки). (k, n)-пороговая схема с $n = 2k - 1$ дает очень надежную схему управления ключами: мы можем восстановить оригинальный ключ, даже если $\lfloor n/2 \rfloor = k - 1$ из n частей уничтожены, но наши противники не смогут реконструировать ключ, даже если в результате нарушения безопасности будут раскрыты $\lfloor n/2 \rfloor = k - 1$ из оставшихся k частей.

В других приложениях имеет место компромисс не между секретностью и надежностью, а между безопасностью и удобством использования. Рассмотрим, к примеру, компанию, которая подписывает все свои чеки цифровой подписью (Rivest et al. 1978). Если каждому должностному лицу выдать секретный ключ компании для подписания, то система будет удобной, но зато и использовать ее неправильно будет просто. Если для подписания каждого чека необходимы совместные действия всех должностных лиц компании,

то система будет безопасной, но неудобной. Стандартное решение требует подписывать каждый чек по меньшей мере тремя подписями, и его легко реализовать с помощью пороговой (3, n)-схемы. Каждому должностному лицу выдается небольшая магнитная карта, содержащая одну из частей D_i, а принадлежащее компании устройство генерирования подписей принимает любые три из них, чтобы сгенерировать (и впоследствии уничтожить) временную копию ключа подписи D. Устройство не содержит никакой секретной информации, и потому его не нужно защищать от изучения посторонними. У предателя должно быть по меньшей мере два сообщника, чтобы при такой схеме подделать подпись компании.

Пороговые схемы прекрасно подходят для приложений, в которых должны сотрудничать отдельные люди, не доверяющие друг другу и имеющие противоречивые интересы. В идеале мы хотели, чтобы сотрудничество было основано на взаимном согласии, но право вето, которым этот механизм наделяет каждого члена, способно парализовать деятельность всей группы. Путем подходящего подбора параметров k и n мы можем предоставить значительному большинству власть предпринимать действия, но при этом оставить за достаточно большим меньшинством право заблокировать это действие.

46.2. Простая пороговая (k, n)-схема

Наша схема основана на полиномиальной интерполяции: если дано k точек на двумерной плоскости $(x_1, y_1), \ldots, (x_k, y_k)$ с различными x_i, то существует один и только один полином $q(x)$ степени $k - 1$ такой, что $q(x_i) = y_i$ для всех i. Без ограничения общности можно предполагать, что данные D являются (или могут быть представлены) числом. Чтобы разбить его на части D_i, случайным образом выберем полином степени $k - 1$ $q(x) = a_0 + a_1 x + \ldots + a_{k-1} x^{k-1}$, в котором $a_0 = D$, и вычислим:

$$D_1 = q(1), \ldots, D_i = q(i), \ldots, D_n = q(n).$$

Зная любое подмножество, содержащее k из этих значений D_i (вместе с индексами), мы можем найти коэффициенты $q(x)$ с помощью интерполяции, а затем вычислить $D = q(0)$. С другой стороны, знания всего лишь $k - 1$ из этих значений недостаточно для вычисления D.

Чтобы сделать это утверждение более точным, будем использовать не обычную, а модульную арифметику. Множество целых чисел по модулю простого числа p образует поле, в котором возможна интерполяция. Зная целочисленные данные D, выберем простое число p, большее D и n. Коэффициенты $a_1, \ldots, a_{k-1}$ полинома $q(x)$ случайным образом выбираются из равномерного распределения целых чисел на полуинтервале $[0, p)$, а значения $D_1, \ldots, D_n$ вычисляются по модулю p.

Предположим теперь, что $k - 1$ из этих n частей стали известны противнику. Для каждого значения-кандидата D', принадлежащего $[0, p)$, он может построить один и только один полином $q'(x)$ степени $k - 1$ такой, что $q'(0) = D'$ и $q'(i) = D_i$ для $k - 1$ заданных аргументов. По построению, эти p возможных

полиномов равновероятны, и потому противник не может узнать абсолютно ничего об истинном значении D.

Эффективные алгоритмы сложности $O(n \log_2 n)$ вычисления и интерполяции полиномов обсуждались в работах Aho et al. (1974) и Knuth (1997b), но даже быстродействия прямолинейных квадратичных алгоритмов достаточно для практических схем управления ключами. Если число D длинное, то рекомендуется разбить его на более короткие битовые блоки (которые обрабатываются по отдельности), чтобы избежать арифметических операций с многократной точностью. Блоки не могут быть сколь угодно короткими, поскольку наименьшее пригодное значение p равно $n + 1$ (должно быть по меньшей мере $n + 1$ различных значений, принадлежащих $[0, p)$, в которых вычисляется $q(x)$). Однако это ограничение не слишком обременительное, потому что 16-битового модуля (который можно обработать с помощью дешевого 16-разрядного арифметического устройства) достаточно для приложений, в которых количество частей D_i не превышает 64 000.

Ниже перечислено несколько полезных свойств этой пороговой (k, n)-схемы (по сравнению с решениями на основе механических замков и ключей).

1. Размер каждой части не превосходит размера исходных данных.
2. Если зафиксировать k, то части D_i можно динамически добавлять и удалять (например, когда должностные лица приходят на работу и увольняются), не оказывая влияния на прочие части D_i. (Часть удаляется, только когда увольняющийся сотрудник делает ее полностью недоступной, в т. ч. себе самому.)
3. Легко изменить части D_i, не изменяя исходных данных D, – нужно только выбрать новый полином $q(x)$ с таким же свободным членом. Частое изменение такого рода может значительно повысить безопасность, потому что части, ставшие известными в результате нарушений системы безопасности, невозможно накапливать, если только не все они связаны с одной и той же редакцией полинома $q(x)$.
4. Благодаря использованию кортежей значений полинома в качестве частей D_i мы можем получить иерархическую схему, в которой количество необходимых частей зависит от их важности. Например, если мы выдадим президенту компании три значения $q(x)$, каждому вице-президенту – по два значения $q(x)$, а каждому рядовому управленцу – по одному значению $q(x)$, то пороговая $(3, n)$-схема позволит подписывать чеки либо любым трем управленцам, либо любым двум управленцам, при условии что один из них – вице-президент, либо одному президенту.

Полиномы можно заменить любым другим набором функций, лишь бы они допускали простое вычисление и интерполяцию. Другая (несколько менее эффективная) пороговая схема была недавно разработана Дж. Р. Блэкли (Blakley 1979).

Литература

Abelson, R., and J. Carroll. 1965. Computer simulation of individual belief systems. Amer. Behav. Scientist 8 (9): 24–30.

Abramson, Norman. 1970. The Aloha system. In AFIPS conf. prof., Vol. 37, 281–285. Montvale, NJ: AFIPS Press.

Abramson, Norman. 1973. Computer-communication networks. Computer applications in electrical engineering series. Englewood Cliffs, NJ: Prentice-Hall.

ACM. 1968. Richard W. Hamming (ACMTuring award citation). https://amturing.acm.org/info/hamming_1000652.cfm.

ACM. 2004. Vinton Cerf and Robert Kahn (ACM Turing award citation). https://amturing.acm.org/award_winners/cerf_1083211.cfm.

Adams, C. W., S. Gill, and D. Combalic, eds. 1954. Notes from a special summer program in digital computers: Advanced coding techniques. Cambridge, MA: MassachusettsInstitute of Technology, Department of Electrical Engineering, Digital Computer Laboratory.

Agrawal, Manindra, Neeraj Kayal, and Nitin Saxena. 2004. PRIMES is in P. Annals of Mathematics 160 (2): 781–793.

Aho, Alfred V., J. E. Hopcroft, and J. D. Ullman. 1974. The design and analysis of computer algorithms. Boston: Addison-Wesley.

Aiken, Howard, A. G. Oettinger, and T. C. Bartee. 1964. Proposed automatic calculating machine. IEEE Spectrum 1 (8): 62–69.

Aitchison, T. M., et al. 1970. Comparative evaluation of index languages, Technical report, Institution of Electrical Engineers, London.

al Khwārizmī, Muh.ammad Ibn Mūsā. 1915. Robert of Chester's Latin translation of the algebra of al Khowarizmi. University of Michigan studies. Humanistic series; 11. New York; London: Macmillan.

Aleph-Null. 1971. Computer recreations. Software Practice and Experience 1 (2): 201–204.

Alt, Franz L. 1948a. A Bell Telephone Laboratories' computing machine. I. Mathematics of Computation 3 (21): 1–13.

Alt, Franz L. 1948b. A Bell Telephone Laboratories' computing machine. II. Mathematics of Computation 3 (22): 69–84.

Aristotle. 1933. The metaphysics. Loeb classical library. Cambridge, MA: Harvard University Press.

Aristotle. 1989. Prior analytics. Indianapolis: Hackett Pub. Co.

Artandi, Susan, and Edward H. Wolf. 1969. The effectiveness of automatically generated weights and links in mechanical indexing. American Documentation 20 (3): 198–202.

Ash, R., E. Broadwin, V. Della Valle, M. Greene, A. Jenny, C. Katz, and L. Yu. 1957. Preliminary manual for MATH-MATIC and ARITH-MATIC systems for algebraic translation and compilation for UNIVAC I and II. Automatic program development, Remington Rand UNIVAC. http://archive.computerhistory.org/resources/text/Knuth_Don_X4100/PDF_index/k-7-pdf/k-7-u2310-UNIVAC-MATH-MATIC-ARITH-MATIC.pdf.

Ashby, W. Ross. 1952. Design for a brain. New York: J. Wiley.

Ashby, W. Ross. 1956. Design for an intelligenceamplifier. In Automata studies, eds. Claude E. Shannon and J. McCarthy, 215–234. Princeton, NJ: Princeton University Press.

Ashenhurst, R. L., and R. H. Vonderohe. 1975. A hierarchical network. Datamation 21 (2): 40–44.

Atchison,William F., Samuel D. Conte, JohnW. Hamblen, Thomas E. Hull, Thomas A. Keenan,William B. Kehl, Edward J. McCluskey, Silvio O. Navarro, Werner C. Rheinboldt, Earl J. Schweppe, William Viavant, and David M. Young Jr. 1968. Curriculum 68: Recommendations for academic programs in computer science: A report of the acm curriculum committee on computer science. Commun. ACM 11 (3): 151–197. doi:10.1145/362929.362976. http://doi.acm.org/10.1145/362929.362976.

Babai, László. 2016. Graph isomorphism in quasipolynomial time. arXiv.org. https://arxiv.org/abs/1512.03547.

Babbage, Charles. 1843. Letter from Babbage to Prince Albert. https://www.instagram.com/p/ButJtlMnBrV/.

Babbage, Charles. 1989. Science and reform: Selected works of Charles Babbage. Cambridge; New York: Cambridge University Press.

Bachman, C. W. 1965. Software for random access processing. Datamation.

Bachmann, Paul. 1894. Die analytische zahlentheorie. Zahlentheorie. versuch einer gesammtdarstellung dieser wissenschaft in ihren haupttheilen. 2. th. Leipzig: Teubner.

Baker, F. T. 1972. Chief programmer team management of production programming. IBM Systems Journal 11 (1): 56–73.

Baran, Paul. 1964. On distributed communications networks (all-digital data distributed communications network with connected adjacent stations for increased survivability if nodes and links are destroyed). IEEE Transactions on Communications Systems cs-12: 1–9.

Barber, D. L. A. 1972. The European computer network project. In Computer communications: Impacts and implications, ed. S. Winkler, 192–200. Washington, DC.

Barnes, G. H., R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes. 1968. The ILLIAC IV computer. IEEE Transactions on Computers C-17 (8): 746–757.

Bateman, P., and D. Diamond. 1978. John E. Littlewood (1885–1977): An informal obituary. The Math. Intelligencer 1 (1): 28–33.

BBN. 1973. Specification for the interconnection of a host and an IMP, Technical Report BBN 1822 (revised), Bolt Beranek and Newman Inc., Cambridge, MA.

Bennett, James. 1962. On spectra. Princeton, NJ: Princeton University Press.

Bernstein, Alex, and Michael de V. Roberts. 1958. Computer v. chess-player. Scientific American 198 (6): 96–106.

Blakley, G. R. 1979. Safeguarding cryptographic keys. In 1979 international workshop on managing requirements knowledge (MARK), 313–318.

Bledsoe,W.W., and I. Browning. 1959. Pattern recognition and reading by machine. In IRE-AIEE-ACM '59 (Eastern): Papers presented at the December 1–3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference, ed. F. Heart. New York: ACM.

Bleier, Robert. 1967. Treating hierarchical data structures in the SDC Time-Shared Data Management System (TDMS). In ACM annual conference/annual meeting: Proceedings of the 1967 22nd national conference, 41–49. ACM.

Bobrow, Daniel G. 1964. Natural language input for a computer problem solving system. http://hdl.handle.net/2027/pst.000008813244.

Böhm, Corrado, and Giuseppe Jacopini. 1966. Flow diagrams, Turing machines and languages with only two formation rules. Comm. ACM 9 (5): 366–371.

Boole, George. 1854. An investigation of the laws of thought, on which are founded the mathematical theories of logic and probabilities. London: Walton and Maberly.

Borko, H. 1968. Interactive document storage and retrieval systems–design concepts. In Mechanised information storage, retrieval and dissemination, ed. K. Samuelson, 591–599. Amsterdam: North-Holland Publishing.

Borůvka, Otakar. 1926. On a minimal problem. Práce Moravské Pridovedecké Spolecnosti 3.

Bowles, Nellie. 2019. Overlooked no more: Karen Spärck Jones, who established the basis for search engines. New York Times, 2 January 2019: D6.

Brooks, Frederick P. 1956. The analytic design of automatic data processing systems. PhD diss., Harvard University).

Brooks, Frederick P. 1987. No silver bullet: Essence and accidents of software engineering. Computer 20 (4): 10–19.

Brooks, Frederick P. 1995. The mythical man-month: essays on software engineering, Anniversary edn. Boston: Addison-Wesley[1].

Brown, G., J. C. R. Licklider, J. McCarthy, and A. Perlis. 1962. Management and the computer of the future. MIT Press.

Burks, Arthur W., Herman H. Goldstine, and John von Neumann. 1947. Preliminary discussion of the logical design of an electronic computing instrument. planning and coding of problems for an electronic computing instrument. Princeton, NJ: Institute for Advanced Study.

Bush, Vannevar. 1945a. As we may think. Atlantic Monthly 176: 101–108.

Bush, Vannevar. 1945b. As we may think. Life, 10 September 1945: 112–124.

Bush, Vannevar. 1945c. Science, the endless frontier. U. S. Government Printing Office. https://www.nsf.gov/od/lpa/nsf50/vbush1945.htm.

Butler, Samuel. 1863. Darwin among the machines (letter to the editor). The Press (Christchurch NZ) June 13 1863. http://www.nzetc.org/tm/scholarly/tei-ButFir-t1-g1-t1-g1-t4-body.html.

Butler, Samuel. 1872. Erewhon: or, Over the range. London: Trübner.

Cantor, Georg. 1996. On an elementary question in the theory of manifolds. In From Kant to Hilbert: A source book in the foundations of mathematics, ed. William B. Ewald Jr. Oxford science publications, 920–922. Oxford: Clarendon Press.

Carnap, Rudolf. 1937. The logical syntax of language. International library of psychology, philosophy, and scientific method. London: K. Paul, Trench, Trubner.

Carr, S., S. Crocker, and V. Cerf. 1970. HOST-HOST communication protocol in the ARPA network. In Spring Joint Computer Conferemce, AFIPS Conf. Proc., Vol. 36, 539–597. Montvale, NJ: AFIPS Press.

Cerf, V., and R. Kahn. 1974. A protocol for packet network intercommunication. IEEE Transactions on Communications 22 (5): 637–648.

Chambon, J. F., M. Elie, J. Le Bihan, G. LeLann, and H. Zimmerman. 1973. Functional specification of transmission station in the CYCLADES network. ST-ST protocol, Technical Report SCH502.3, I.R.I.A.

Chandrasekhar, S. 1943. Stochastic problems in physics and astronomy. Reviews of Modern Physics 15 (1): 1–89.

Childs, D. L. 1968. Feasibility of a set-theoretical data structure–a general structure based on a reconstituted definition of relation. In Proceedings IFIP Cong, 162–172. Amsterdam: North Holland Publishing.

Church, Alonzo. 1936a. A note on the Entscheidungsproblem. J. Symbolic Logic 1 (1): 40–41.

Church, Alonzo. 1936b. An unsolvable problem of elementary number theory. American Journal of Mathematics 58 (2): 345–363.

Church, Alonzo. 1941. The calculi of lambda-conversion. Annals of mathematics studies; Number 6. Princeton, NJ: Princeton University Press.

Church, Alonzo. 1956. Introduction to mathematical logic. Princeton mathemati-

[1] *Брукс Ф. П. мл.* Как проектируются и создаются программные комплексы. Мифический человеко-месяц. М.: Наука, глав. ред. физико-матем. литературы, 1979.

cal series; 17. Princeton, NJ: Princeton University Press.

Clark, W. A., J. M. Frankovich, H. P. Peterson, J. W. Forgie, R. L Best, and K. H. Olsen. 1957. The Lincoln TX-2 computer. In Proc. Wstrn. Jt. Computer Conf. Clay Mathematics Institute. 2000. Millennium Prize Problems. https://www.claymath.org/millennium-problems.

Cleverdon, C. W., J. Mills, and E. M. Keen. 1966. Factors determining the performance of indexing systems. Cranfield, Engand: ASLIB-Cranfield Research Project.

Cobham, Alan. 1965. The intrinsic computational difficulty of functions. In Logic, methodology and philosophy of science: Proceedings of the 1964 International Congress (Studies in Logic and the Foundations of Mathematics), ed. Yehoshua Bar-Hillel, 24–30. Amsterdam: North-Holland Publishing.

Codd, E. 1960. Multiprogram scheduling: parts 1 and 2. Introduction and theory. Comm. ACM 3 (6): 347–350.

Codd, E. 1970. A relational model of data for large shared data banks. Comm. ACM 13 (6): 377–387. https://doi.org/10.1145/362384.362685.

Cohen, I. Bernard. 1999. Howard Aiken: portrait of a computer pioneer. History of computing. Cambridge, MA: MIT Press.

Cohen, Paul J. 1963. The independence of the continuum hypothesis. Proceedings of the National Academy of Sciences of the United States of America 50 (6): 1143–1148.

Conway, Flo. 2005. Dark hero of the information age: in search of Norbert Wiener, the father of cybernetics. New York: Basic Books.

Cook, Stephen. 1971a. Characterizations of pushdown machines in terms of time-bounded computers. J. ACM 18 (1): 4–18.

Cook, Stephen. 1971b. The complexity of theorem-proving procedures. Symposium on Theory of Computing (STOC), 151–158. New York: Association for Computing Machinery. https://doi.org/10.1145/800157.805047.

Cooper, Franklin S., Pierre C. Delattre, Alvin M. Liberman, John M. Borst, and Louis J. Gerstman. 1952. Some experiments on the perception of synthetic speech sounds. J. Acoustical Society of America 24 (6): 597–606.

Corbató Fernando J., and Arthur L. Norberg. 1989. OH 162: An interview with Fernando J. Corbat´o on 18 April 1989 and 14 November 1990. The Charles Babbage Institute, University of Minnesota.

Corbató, Fernando J., Marjorie Merwin Daggett, and Robert C. Daley. 1962. An experimental time-sharing system. In Proceedings of Spring Joint Computer Conference sponsored by the American Federation of Information Processing Societies. New York: Association for Computing Machinery. https://doi.org/10.1145/1460833.1460871.

Corneil, D., and C. Gotlieb. 1970. An efficient algorithm for graph isomorphism. J. ACM 17 (1): 51–64.

Crocker, S. D., J. F. Heafner, R. M. Metcalfe, and J. B. Postel. 1972. Function-oriented protocols for the ARPA computer network. In AFIPS conf. proc., Spring Joint Computer Conference, Vol. 40, 271–279. Montvale, NJ: AFIPS Press.

Culbertson, James T. 1950. Consciousness and behavior; a neural analysis of behavior and of consciousness. Dubuque, IA: W. C. Brown.

Culbertson, James T. 1956. Some uneconomical robots. In Automata studies, eds. Claude E. Shannon and J. Mc-Carthy, 99–116. Princeton, NJ: Princeton University Press.

Curtice, P. M., and P. E. Jones. 1968. An operational interactive retrieval system. Paris: Arthur D. Little.

Davis, K. H., R. Biddulph, and S. Balashek. 1952. Automatic recognition of spoken digits. J. Acoustical Society of America 24 (6): 637–642.

Davis, Martin. 1965. The undecidable; Basic papers on undecidable propositions, unsolvable problems and computable functions. Hewlett, NY: Raven Press.

Davis, Martin, and Hilary Putnam. 1960. A computing procedure for quantification theory. J. ACM 7 (3): 201–215.

Davis, P. J. 1972. Fidelity in mathematical discourse: Is one and one really two? The American Mathematical Monthly 79 (3): 252–263.

DEC. 1972. Digital Equipment Corporation PDP-11/40 processor handbook, 1972, and PDP-11/45 processor handbook, 1971. Maynard, MA: Digital Equipment Corporation.

Dell, F. R. E. 1971. Features of a proposed synchronous data network. In Proc. 2nd symp. problems in the optimization of data communications systems, 50–57.

DeMillo, Richard, Richard Lipton, and Alan Perlis. 1977. Social processes and proofs of theorems and programs. In Annual symposium on principles of programming languages: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on principles of programming languages: Los.

Angeles, California; 17–19 Jan. 1977, 206–214. DeMillo, Richard, Richard Lipton, and Alan Perlis. 1979. Social processes and proofs of theorems and programs. Comm. ACM 22 (5): 271–280.

Despres, R. 1972. A packet switching network with graceful saturated operation. In First international conference on computer communication: Computer communications: Impacts and implications, October 24–26, 345–351.Washington, DC.

Dewey, Godfrey. 1923. Relative frequency of English speech sounds. Cambridge, MA: Harvard University Press.

Diffie, W. 1988. The first ten years of public-key cryptography. Proceedings of the IEEE 76 (5): 560–577.

Diffie,W., and M. Hellman. 1976a. New directions in cryptography. IEEE Transactions on Information Theory 22 (6): 644–654.

Diffie, W., and M. E. Hellman. 1976b. Multiuser cryptographic techniques. In Proceedings of the June 7–10, 1976, national computer conference and exposition. New York: ACM.

Diffie,W., and M. E. Hellman. 1977. Exhaustive cryptanalysis of the nbs data encryption standard. Computer 10 (6): 74–84.

Dijkstra, Edsger W. 1959. A note on two problems in connexion with graphs. Numerische Mathematik 1 (1): 269–271.

Dijkstra, Edsger W. 1965. Solution of a problem in concurrent programming control. Comm. ACM 8 (9): 569. https://doi.org/10.1145/365559.365617.

Dijkstra, Edsger W. 1968a. Go to statement considered harmful [letter to the editor]. Comm. ACM 11 (3): 147–148. https://doi.org/10.1145/362929.362947.

Dijkstra, Edsger W. 1968b. The structure of the "THE"-multiprogramming system. Comm. ACM 11 (5): 341–346. https://doi.org/10.1145/363095.363143.

Dijkstra, Edsger W. 1972. Notes on structured programming. In Structured programming, eds. O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, 1–81.

Dijkstra, Edsger W. 1975. Comments at a symposium. http://www.cs.utexas.edu/users/EWD/transcriptions/EWD05xx/EWD512.html.

Dinneen, G. P. 1955. Programming pattern recognition. In Proc. Western Joint Computer Conference, 94–100.

Diophantus. 1910. Diophantus of Alexandria, a study in the history of Greek algebra. Cambridge: Cambridge University Press.

du Bois-Reymond, Paul. 1870. Sur la grandeur relative des infinis des fonctions. Annali di Matematica Pura ed Applicata (1867–1897) 4 (1): 338–353.

Dunn, H. K. 1950. The calculation of vowel resonances, and an electrical vocal tract. J. Acoustical Society of America 22 (6): 740–753.

EAI. 1959. Handbook for Variplotter Models 205S and 205T, PACE. Long Branch, NJ: Electronic Associates Incorporated.

Eccles, John C. 1953. The neurophysiological basis of mind; the principles of neurophysiology. Oxford: Clarendon Press.

Edmonds, Jack. 1965. Paths, trees, and flowers. Canadian Journal of Mathematics 17: 449–467.

Edmonds, Jack, and Richard Karp. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM 19 (2): 248–264.

Engelbart, D. C. 1962. Augmenting human intellect: A conceptual framework. Prepared for: Director of Information Sciences, Air Force Office of Scientific Research.

Washington DC, Contract AF 49(638)–1024, Technical Report SRI 3678, AFOSR 3223, Stanford Research Institute, Menlo Park, CA.

Euclid. 1956. The thirteen books of Euclid's elements, Revised 2nd edn. New York: Dover Publications.

Evans Jr., Arthur, William Kantrowitz, and Edwin Weiss. 1974. A user authentication scheme not requiring secrecy in the computer. Comm. ACM 17 (8): 437–442.

Fant, G. 1959. On the acoustics of speech. In Third international congress on acoustics. Stuttgart, Germany.

Farber, D. J. 1973. The distributed computing system. In Proc. 7th ann. IEEE Computer Soc. International Conf., 31–34.

Farber, D. J. 1975a. A ring network. Datamation 21 (2): 44–46.

Farber, D. J. 1975b. A virtual channel network. Datamation 21 (2): 51–53.

Farley, B. G., and W. Clark. 1954. Simulation of selforganizing systems by digital computer. IEEE Transactions on Information Theory 4 (4): 76–84.

Feldman, J. A., and P. D. Rovner. 1968. An Algol-based associative language, Technical report, Stanford Artificial Intelligence Laboratory, Stanford, CA.

Fetzer, James. 1988. Program verification: the very idea. Comm. ACM 31 (9): 1048–1063.

Floyd, Robert. 1962. Algorithm 97: Shortest path. Comm. ACM 5 (6): 345.

Floyd, Robert W. 1967. Assigning meanings to programs. Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics 19. doi:10.1090/psapm/019/0235771.

Forgie, James W., and Carma D. Forgie. 1959. Results obtained from a vowel recognition computer program. J. Acoustical Society of America 31 (6): 1480–1489.

Frechet, M. 1938. Methode des fonctions arbitraires. Theorie des evenements en chaıne dans le cas d'un nombre fini

d'etats possibles. Paris: Gauthier-Villars.

Fredkin, Edward. 1960. Trie memory. Comm. ACM 3 (9): 490–499.

Frege, Gottlob. 1879. Begriffsschrift, eine der arithmetischen nachgebildete formelsprache des reinen denkens. Halle an der Saale: L. Nebert.

Friedberg, R. M. 1958. A learning machine: Part I. IBM J. Research and Development 2 (1): 2–13.

Garey, Michael R., and David S. Johnson. 1979. Computers and intractability: a guide to the theory of npcompleteness. A series of books in the mathematical sciences. San Francisco: W. H. Freeman.

Gauss, Carl Friedrich. 1986. Disquisitiones arithmeticae, English edn. New York: Springer.

Gefter, Amanda. 2015. The man who tried to redeem the world with logic. Nautilus 21: 95–103. http://nautil.us/issue/21/information/the-man-who-triedto-redeem-the-world-with-logic.

Gelernter, H. 1959. Realization of a geometry theorem proving machine. In International conference on information processing. Paris: UNESCO, NS, ICIP, 1.6.6.

Gelernter, H., N. S. Sridharan, A. J. Hart, S. C. Yen, F. W. Fowler, and H. J. Shue. 1973. Cheminform abstract: The discovery of organic synthetic routes by computer. Chemischer Informationsdienst 4 (52).

George, J. Alan. 1971. Computer implementation of the finite element method. PhD diss, Stanford University.

Gilmore, T., and R. E. Savell. 1959. A program for the production of proofs for theorems derivable within the first order predicate calculus from axioms. In International conference on information processing. Paris: UNESCO, NS, ICIP, 1.6.14.

Gorn, S. 1964. Semiotic relationships in unambiguously stratified language systems. In Int. colloq. algebraic linguistics and automatic theory. Jerusalem: Hebrew University of Jerusalem.

Gödel, Kurt. 1931. Ü ber formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik und Physik 38 (1): 173–198.

Hamming, R.W. 1950. Error detecting and error correcting codes. The Bell System Technical Journal 29 (2): 147–160.

Hardy, G. H. 1924. Orders of infinity, 2d edn. Cambridge tracts in mathematics and mathematical physics. Cambridge: Cambridge University Press.

Hardy, G. H., and J. Littlewood. 1914. Some problems of diophantine approximation. Acta Mathematica 37 (1): 155–191.

Hardy, G. H., and J. Littlewood. 1916. Contributions to the theory of the Riemann zeta-function and the theory of the distribution of primes. Acta Mathematica 41 (1): 119–196.

Hartley, R. V. L. 1928. Transmission of information. The Bell System Technical Journal 7 (3): 535–563.

Hartree, Douglas R. 1949. Calculating instruments and machines. Urbana: University of Illinois Press.

Hayek, Friedrich A. von. 1952. The sensory order; An inquiry into the foundations of theoretical psychology.Chicago: University of Chicago Press.

Heart, F. E., R. E. Kahn, S. Ornstein, W. Crowther, andD. Walden. 1970. The interface message processor for the ARPA computer network. In 1970 Spring Joint Computer Conference, AFIPS Conf. Proceedings, Vol. 36. Montvale, NJ: AFIPS Press.

Heart, F. E., S. M. Ornstein, W. R. Crowther, and W. B. Barker. 1972. A new minicomputer-multiprocessor for the ARPA network. In AFIPS conf. proc., 1972 SJCC, Vol. 42, 529–537. Montvale, NJ: AFIPS Press.

Heawood, P. J. 1890. Map colouring theorems. Quarterly J. Math., Oxford Series 24: 322–339.

Hebb, D. O. 1949. The organization of behavior: a neuropsychological theory. Wiley book in clinical psychology. New York: Wiley.

Heller, J. 1961. Sequencing aspects of multiprogramming. J. ACM 8 (3): 426–439.

Hellman, M. 1977. An extension of the Shannon theory approach to cryptography. IEEE Transactions on Information Theory 23 (3): 289–294.

Hilbert, David. 1902. Mathematical problems. Bulletin of the American Mathematical Society 8 (10): 437–479.

Hilbert, David. 1928. Die grundlagen der mathematik. Abhandlungen aus dem Mathematischen Seminar der Hamburgischen Universität 6.

Hoare, C. A. R. 1962. Quicksort. The Computer Journal 5 (1): 10–16.

Hoare, C. A. R. 1969. An axiomatic basis for computer programming. Comm. ACM 12 (10): 576–580. https://doi.org/10.1145/363235.363259.

Hoare, C. A. R. 1981. Emperor's old clothes. Comm. ACM 24 (2): 75–83.

Hoare, C. A. R. 1996. Unification of theories: A challenge for computing science. Recent Trends In Data Type Specification 1130: 49–57.

Hobbes, Thomas. 1655. Elementorum philosophiæ sectio prima De corpore authore Thoma Hobbes Malmesburiensi.

Early english books online. Londini: excusum sumptibus Andreæ Crook sub signo Draconis viridis in Coemeterio B. Pauli.

Hobson, Ernest William. 1921. The theory of functions of a real variable and the theory of

Fourier's series, 2nd edn. Cambridge: Cambridge University Press.

Hodges, Andrew. 1983. Alan Turing: the enigma. New York: Simon and Schuster.

Homer. 1962. The Iliad. Chicago: University of Chicago Press.

Hopper, Grace. 1952. The education of a computer. In ACM Annual Conference/Annual Meeting: Proceedings of the 1952 ACM national meeting (Pittsburgh): Pittsburgh, Pennsylvania, 243–249.

Huffman, David A. 1952. A method for the construction of minimum-redundancy codes. Proceedings of the IRE 40 (9): 1098–1101.

IBM. 1956. FORTRAN programmer's reference manual. White Plains, NY: IBM.

IBM. 1965a. GIS (Generalized Information System), application description manual H20-0574. White Plains, NY: IBM.

IBM. 1965b. Information Management System/360, application description manual H20-0524-1. White Plains, NY: IBM.

IBM. 1974. IBM synchronous data link control–general information. Research Triangle Park, NC: IBM Systems Development Division.

IBM. 1975. IBM system network architecture–general information. Research Triangle Park, NC: IBM Systems Development Division.

IDS. 1968. IDS reference manual ge 625/635, CPB 1093B. Phoenix, AZ: GE Information Systems Division.

Jarník, V. 1930. O jistém problému minimálním. Práce Moravské Přírodovédecké Společnosti 6 (4): 57–63.

Jech, Thomas J. 1973. The axiom of choice. Amsterdam: North-Holland Publishing. http://hdl.handle.net/2027/coo.31924000209357.

Jefferson, Geoffrey. 1949. The mind of mechanical man. British Medical Journal 1 (4616). http://bmj.com/content/1/4616/1105.full.pdf.

Jevons, William Stanley. 1874. The principles of science: A treatise on logic and scientific method. London: Macmillan.

Jones, Matthew L. 2016. Reckoning with matter: calculating machines, innovation, and thinking about thinking from pascal to babbage. Chicago: University of Chicago Press.

Kahn, David. 1967. The codebreakers: The story of secret writing. New York: Macmillan.

Kahn, R., and W. Crowther. 1972. Flow control in a resource-sharing computer network. IEEE Transactions on Communications 20 (3): 539–546.

Kahn, R. R. 1975. The organization of computer resources into a packet radio network. In Proc. 1975 NCC, Vol. 44, 177–186. Montvale, NJ: AFIPS Press.

Karatsuba, A., and Yuri Ofman. 1962. Multiplication of many-digital numbers by automatic computers. Proc. USSR Academy of Sciences 142: 293–294.

Karp, R. M. 1972. Reducibility among combinatorial problems. In Complexity of computer computations, eds. R. E. Miller, J. W. Thatcher, and J. D. Bohlinger. The IBM research symposia series. Boston: Springer. https://doi.org/10.1007/978-1-4684-2001-2_9.

Keen, E., and J. Digger. 1972. Report of an information science index languages test. Aberystwyth, Department of Information Retrieval Studies, College of Librarianship Wales, June 72, (2v) 173 p.

Kempe, A. B. 1879. On the geographical problem of the four colours. American Journal of Mathematics 2 (3): 193–200.

Kendall, Maurice G. 1939. Tables of random sampling numbers. Tracts of computers. Cambridge: Cambridge University Press.

Khachiyan, L. G. 1979. A polynomial algorithm for linear programming (in Russian). Doklady Akadmiia Nauk USSR 244: 1093–1096.

Kleene, S. C. 1935a. A theory of positive integers in formal logic, Part I. American Journal of Mathematics 57 (1): 153–173.

Kleene, S. C. 1935b. A theory of positive integers in formal logic, Part II. American Journal of Mathematics 57 (2): 219–244.

Kleene, S. C. 1951. Representation of events in nerve nets and finite automata, Technical report, RAND Project Air Force, Santa Monica, CA.

Klyuev, V. V., and H. I. Kokovkin-Shcherbak. 1965. Minimization of the number of arithmetic operations in the solution of linear algebraic systems of equations. USSR Computational Mathematics and Mathematical Physics 5 (1): 25–43.

Knuth, Donald Ervin. 1963. Notes On «Open» Addressing. http://jeffe.cs.illinois.edu/teaching/datastructures/2011/notes/knuth-OALP.pdf.

Knuth, Donald Ervin. 1965. On the translation of languages from left to right. Information and Control 8 (6): 607–639.

Knuth, Donald Ervin. 1969. The art of computer programming, Volume 2: Seminumerical algorithms. Boston: Addison-Wesley.

Knuth, Donald Ervin. 1973. The art of computer programming, Vol. 3: Sorting and searching. Boston: Addison-Wesley.

Knuth, Donald Ervin. 1974a. Structured programming with go to statements. ACM Computing Surveys (CSUR) 6 (4): 261–301.

Knuth, Donald Ervin. 1974b. A terminological proposal. SIGACT News 6 (1): 12–18. doi:10.1145/1811129. 1811130.

Knuth, Donald Ervin. 1976. Big omicron and big omega and big theta. ACM SIGACT News 8 (2): 18–24.

Knuth, Donald Ervin. 1986a. The METAFONTbook. Boston: Addison-Wesley.

Knuth, Donald Ervin. 1986b. The TeXbook. Boston: Addison-Wesley.

Knuth, Donald Ervin. 1997a. The art of computer programming. Volume 1, Fundamental algorithms, 3rd edn. Boston: Addison Wesley.

Knuth, Donald Ervin. 1997b. The art of computer programming, Volume 2: Seminumerical algorithms, 3rd edn. Boston: Addison-Wesley.

Knuth, Donald Ervin. 1998. The art of computer programming. Volume 3, Sorting and searching, 2nd edn. Boston: Addison-Wesley.

Knuth, Donald Ervin. 2011. The art of computer programming. Volume 4A, Combinatorial algorithms, part 1. Boston: Addison Wesley.

Köhler, W. 1951. Relational determination in perception. In Cerebral mechanisms in behavior, ed. Lloyd A. Jeffress, 200–243. New York: Wiley.

Kolata, Gina Bari. 1976. Mathematical proofs: The genesis of reasonable doubt. Science 192 (4243): 989–990.

Kreider, D. L., and R. W. Ritchie. 1964. Predictably computable functionals and definition by recursion. Mathematical Logic Quarterly 10 (5): 65–80.

Kruskal, Joseph B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. Proc. American Mathematical Society 7 (1): 48–50.

Lakatos, Imre. 1976. Proofs and refutations: The logic of mathematical discovery. Cambridge: Cambridge University Press[1].

Lamport, Leslie. 2015. The computer science of concurrency: the early years. https://lamport.azurewebsites.net/pubs/turing.pdf.

Lampson, Butler. 1968. A scheduling philosophy for multiprocessing systems. Comm. ACM 11 (5): 347–360.

Lancaster, F. Wilfrid. 1968. Information retrieval systems; characteristics, testing, and evaluation. Information sciences series. New York: Wiley.

Landau, Edmund. 1909. Handbuch der lehre von der verteilung der primzahlen. Leipzig, Germany: B. G. Teubner.

Laplante, Phillip. 1996. Great papers in computer science. Eagan, MN: West Publishing Company.

[1] *Лакатос И.* Доказательства и опровержения. М.: Наука, 1967.

Lawrence, W. 1956. Methods and purposes of speech synthesis, Technical Report 56/1457, Signals Res. and Dev. Estab., Ministry of Supply, Christchurch, Hants., England.

Leeds, H. D., and G. M. Weinberg. 1961. Multiprogramming. In Computer programming fundamentals, ed. Herbert D. Leeds, 356–359. New York: McGraw-Hill.

Leibniz, G. W. 1666. Dissertatio de arte combinatorica. Leipzig, Germany: Fickium and Seuboldum.

Leibniz, G.W. 2020. The true method. Translated by Lloyd Strickland from the original 1677 manuscript.

Lettvin, J. Y., H. R. Maturana, W. S. McCulloch, and W. H. Pitts. 1959. What the frog's eye tells the frog's brain. Proceedings of the IRE 47 (11): 1940–1951.

Levien, R., and M. Maron. 1967. A computer system for inference execution and data retrieval. Comm. ACM 10 (11): 715–721.

Levin, Leonid A. 1973. Universal search problems (in Russian). Problems of Information Transmission 9 (3).

Levine, Jack, and J. V. Brawley. 1977. Some cryptographic applications of permutation polynomials. Cryptologia 1 (1): 76–92.

Liberman, A. M., Frances Ingemann, Leigh Lisker, and F. S. Cooper. 1959. Minimal rules for synthesizing speech. J. Acoustical Soc. America 31 (1): 1490–1499.

Licklider, J. C. R. 1960. Man-computer symbiosis. IRE Transactions on Human Factors in Electronics HFE-1 (1): 4–11.

Liskov, Barbara, and Stephen Zilles. 1974. Programming with abstract data types. ACM SIGPLAN Notices 9 (4): 50–59. https://doi.org/10.1145/942572.807045.

Ludolph van Ceulen. 1596. Vanden circkel. Delft, Netherlands: Jan Andriesz.

MacHale, Des. 2014. The life and work of George Boole: A prelude to the digital age, New edn. Cork, Ireland: Cork University Press.

Manin, I. 1977. A course in mathematical logic. Graduate texts in mathematics; 53. New York: Springer.

Matiyasevich, Yuri. 1993. Hilbert's tenth problem. Foundations of computing. Cambridge, MA: MIT Press.

McCarthy, John. 1960. Recursive functions of symbolic expressions and their computation by machine, Part I. Comm. ACM 3 (4): 184–195. https://doi.org/10.1145/367177.367199.

McCarthy, John. 1961. Programs with common sense. In Mechanisation of thought processes: proceedings of a symposium held at the National Physical Laboratory on 24th, 25th, 26th and 27th November 1958. London: H. M. Stationery Office. National Physical Laboratory (Great Britain).

McCarthy, John. 1963. Towards a mathematical science of computation. In Information processing 1962: proceedings of IFIP congress 62, convened with the financial assistance of UNESCO, Munich [Germany], 27 August–1 September 1962, ed. Cicely Popplewell, 21–28.

McCulloch, W. S. 1951. Why the mind is in the head. In Cerebral mechanisms in behavior, ed. Lloyd A. Jeffress, 42–111. New York: Wiley.

McCulloch, W. S., and W. Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biophysics 5 (4): 115–133.

McGee, W. C. 1969. Generalized file processing. In Annual review in automatic programming, Vol. 5. New York: Pergamon Press.

McKenzie, A. 1972. HOST-HOST protocol for the ARPA network, Technical Report NIC 8246, Network Information Center, Menlo Park, CA.

McKenzie, A. 1973. HOST-HOST protocol design considerations, Technical Report INWG 16, NIC 13879, Network Information Center, Menlo Park, CA.

Menabrea, Luigi Federico. 1843. Sketch of the analytical engine invented by Charles Babbage, Esq. London: Taylor and Francis.

Merkle, Ralph C. 1978. Secure communications over insecure channels. Comm. ACM 21 (4): 294–299. http://doi.acm.org/10.1145/359460.359473.

Metcalfe, R. M. 1972a. Steady-state analysis of a slotted and controlled alhoa system with blocking. In Proc. 6th Hawaii Conf. on System Sci., 278–281.

Metcalfe, R. M. 1972b. Strategies for interprocess communication in a distributed computing system. In Proc. symp. on computer communication networks and teletraffic. New York: Polytechnic Press.

Metcalfe, R. M. 1972c. Strategies for operating systems in computer networks. In Proc. ACM National Conf. New York: ACM.

Metcalfe, R. M. 1973. Packet communication. PhD diss., Harvard University, Technical Report TR-114, Project MAC.

Metcalfe, Robert, and David Boggs. 1976. Ethernet: Distributed packet switching for local computer networks. Comm. ACM 19 (7): 395–404. https://doi.org/10.1145/360248.360253.

Miller, Gary L. 1976. Riemann's hypothesis and tests for primality. J. Computer and System Sciences 13 (3): 300–317.

Milner, P. 1957. The cell assembly: Mark II. Psychological Review 64.

Minsky, M. L. 1956. Some universal elements for finite automata. In Automata studies, eds. Claude E. Shannon and J. McCarthy, 117–128. Princeton, NJ: Princeton University Press.

Minsky, Marvin, and Seymour Papert. 1969. Perceptrons: an introduction to computational geometry. Cambridge, MA: MIT Press.

Moore, G. E. 1965. Cramming more components onto integrated circuits. Electronics 38: 114–117.

Moore, G. E. 2006. Cramming more components onto integrated circuits. IEEE Solid-State Circuits Society Newsletter 11 (3): 33–35.

Morland, Samuel. 1673. The description and use of two arithmetick instruments. London: Moses Pitt.

Newell, A. 1955. The chess machine: an example of dealing with a complex task by adaptation. In Proc. Western Joint Computer Conference, 101–108.

Newell, A., and J. C. Shaw. 1957. Programming the logic theory machine. In Proc. Western Joint Computer Conference, 230–240.

Newell, Allen, J. C. Shaw, and H. A. Simon. 1958. Chessplaying programs and the problem of complexity. IBM J. Research and Development 2 (4): 320–335.

Niven, Ivan Morton. 1972. An introduction to the theory of numbers. Hoboken, NJ: Wiley. http://hdl.handle.net/2027/umn.319510004761203.

North, J. D. 1954. The rational behavior of mechanically extended man, Technical report, Boulton Paul Aircraft, Wolverhampton, UK.

Nyquist, H. 1924. Certain factors affecting telegraph speed. Bell System Technical Journal 3 (2): 324–346.

Nyquist, H. 1928. Certain topics in telegraph transmission theory. Transactions of the American Institute of Electrical Engineers 47 (2): 617–644.

Ornstein, S. M., W. R. Crowther, M. F. Kraley, R. D.Bressler, A. Michel, and F. E. Heart. 1975. Pluribus–a reliable multiprocessor. In AFIPS Conf. Proc., 1975 NCC, Vol. 44, 551–559. Montvale, NJ.

Parnas, D. L. 1971. Information distribution aspects of design methodology, Technical report, Carnegie-Mellon Univ., Dept. of Computer Science, Pittsburgh.

Peacock, George. 1830. A treatise on algebra. Cambridge: Cambridge University Press.

Perlis, A., and K. Samelson. 1958. Preliminary report: international algebraic language. Comm. ACM 1 (12): 8–22.

Péter, Rózsa. 1951. Rekursive funktionen. Budapest: Akadémiai Kiadó.

Pohlig, S., and M. Hellman. 1978. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance (corresp.). IEEE Transactions on Information Theory 24 (1): 106–110.

Pollard, J. M. 1974. Theorems on factorization and primality testing. Mathematical Proceedings of the Cambridge Philosophical Society 76 (3): 521–528.

Popek, G., J. Horning, B. Lampson, J. Mitchell, and R. London. 1977. Notes on the design of Euclid. ACM SIGPLAN Notices 12 (3): 11–18.

Potter, Robert J. 1977. Electronic mail. Science 195 (4283): 1160.

Pouzin, L. 1973a. Address format in Mitranet, Technical Report INWF 20, NIC 14497, Network Information Center.

Pouzin, L. 1973b. Presentation and major design aspects of the CYCLADES computer network. In Proc. 3rd Data Communications Symposium.

Prachar, Karl. 1957. Primzahlverteilung. Die grundlehren der mathematischen wissenschaften. Berlin: Springer.

Pratt, Fletcher. 1939. Secret and urgent. Garden City, NY: Blue Ribbon Books.

Pratt, Vernon. 1987. Thinking machines: The evolution of artificial intelligence. Oxford, UK: B. Blackwell.

Priestley, Mark. 2011. A science of operations: Machines, logic and the invention of programming. History of computing. London: Springer.

Prim, R. C. 1957. Shortest connection networks and some generalizations. The Bell System Technical Journal 36 (6): 1389–1401.

Purdy, George. 1974. A high security log-in procedure. Comm. ACM 17 (8): 442–445.

Rabin, Michael O. 1976. Probabilistic algorithms. In Algorithms and complexity: Proceedings of a symposium on new directions and recent results in algorithms and complexity held by the Computer Science Department, Carnegie-Mellon University, April 7–9, 1976, ed. Joseph Traub. Cambridge, MA: Academic Press.

Rashevsky, Nicolas. 1938. Mathematical biophysics; physicomathematical foundations of biology. Chicago: Univ. of Chicago Press.

Raymo, Chet. 1996. A giant of logic lost to the irrational. Boston Globe, 25 November 1996: D2.

Ritchie, Dennis, and Ken Thompson. 1974. The UNIX time-sharing system. Comm. ACM 17 (7): 365–375. https://doi.org/10.1145/361011.361061.

Rivest, R., A. Shamir, and L. Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. Comm. ACM 21 (2): 120–126. https://doi.org/10.1145/359340.359342.

Roberts, L. 1973. Capture effects on Aloha channels. In Proc. 6th Hawaii Conf. on System Sci.

Roberts, L., and B. Wessler. 1970a. Computer network development to achieve resource sharing. In 1970 spring joint computer conference, AFIPS conf. proceedings, Vol. 36. Montvale, NJ: AFIPS Press.

Roberts, L., and B. Wessler. 1970b. Computer network development to achieve resource sharing. In AFIPS conf. proc., 1970 Spring Joint Computer Conference, Vol. 36, 543–549. Montvale, NJ: AFIPS Press.

Rogers, Hartley. 1967. Theory of recursive functions and effective computability. McGraw-Hill series in higher mathematics. New York: McGraw-Hill.

Rosenblatt, F. 1958a. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 65 (6): 386–408. https://doi.org/10.1037/h0042519.

Rosenblatt, F. 1958b. The perceptron: A theory of statistical separability in cognitive systems, Technical Report VG-1196-G-1, Cornell Aeronautical Lab., Buffalo, NY.

Rowe, L. A. 1975. The distributed computing operating system, Technical Report 66, Dep. of Information and Computer Sci., University of California, Irvine.

Royce, Winston. 1970. Managing the development of large software systems. In Proceedings, IEEE WESCON, 1–9. Los Angeles, CA: IEEE Computer Society Press.

Royce, Winston. 1987. Managing the development of large software systems. In ICSE '87, proceedings of the 9th international conference on Software Engineering, 328–338. Los Angeles, CA: IEEE Computer Society Press.

Rustin, R., ed. 1970. Computer networks (Proc. Courant Computer Sci. Symp. 3, December, 1970). Englewood Cliffs, NJ: Prentice-Hall.

Salton, G., and M. Lesk. 1968. Computer evaluation of indexing and text processing. J. ACM 15 (1): 8–36.

Salton, Gerard. 1968. Automatic information organization and retrieval. McGraw-Hill computer science series. New York: McGraw-Hill.

Sammet, Jean. 1972. Programming languages: history and future. Comm. ACM 15 (7): 601–610.

Scantlebury, R. A., and P. T. Wilkinson. 1971. The design of a switching system to allow remote access to computer services by other computers and terminal devices. In Proc. 2nd symp. problems in the optimization of data communications systems, 160–167.

Schmitt, W. F., and A. B. Tonik. 1959. Sympathetically programmed computers. In International conference on information processing. Paris: UNESCO, NS, ICIP, 8.2.18.

Searle, John. 1980. Minds, brains and programs. Behavioral and Brain Sciences 3: 417–424. http://cogprints.org/7150/.

Selfridge, O. 1958. Pandemonium, a paradigm for learning. In Proc. symp. mechanisation of thought processes. Teddington, England: Natl. Physical Lab.

Shamir, Adi. 1979. How to share a secret. Comm. ACM 22 (11): 612–613. https://doi.org/10.1145/359168.359176.

Shamir, Adi. 1984. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. IEEE Trans. on Information Theory 30 (5): 699–704.

Shannon, Claude E. 1938. A symbolic analysis of relay and switching circuits. Transactions of the American Institute of Electrical Engineers 57 (12): 713–723.

Shannon, Claude E. 1948. A mathematical theory of communication. Bell System Technical Journal 27 (4): 623–656.

Shannon, Claude E. 1949. Communication theory of secrecy systems. Bell System Technical Journal 28 (4): 656–715.

Shannon, Claude E. 1950. Programming a computer for playing chess. Philosophical Magazine 41 (314).

Shaw, J. C., A. Newell, H. A. Simon, and T. O. Ellis. 1958. A command structure for complex information processing. In Proc. Western Joint Computer Conference, 119–128.

Sherman, H. 1959. A quasi-topological method for recognition of line patterns. In International conference on information processing. Paris: UNESCO, NS, ICIP, H.L.5.

Shieber, Stuart, ed. 2004. The Turing test: Verbal behavior as the hallmark of intelligence. Cambridge, MA: MIT Press.

Shor, PeterW. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Review 41 (2): 303–332.

Simon, Herbert A. 1996. The sciences of the artificial, 3rd edn. Cambridge, MA: MIT Press.

Smith, Jane I., and Yvonne Y. Haddad. 1975. Women in the afterlife. Journal of the American Academy of Religion 43 (1): 39–50.

Solovay, R., and V. Strassen. 1977. A fast Monte-Carlo test for primality. SIAM Journal on Computing 6 (1): 84–85.

Spärck Jones, Karen. 1971. Automatic keyword classification for information retrieval. Hamden, CT: Archon Books. http://hdl.handle.net/2027/uc1.b4333044.

Spärck Jones, Karen. 1972. A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation 28 (1): 11–21.

Sparks, S., and R. G. Kreer. 1947. Tape relay system for radio telegraph operation. R.C.A. Review 8: 393–426.

Stevens, K. N., S. Kasowski, and C. Gunnar M. Fant. 1953. An electrical analog of the vocal tract. J. Acoustical Society of America 25 (4): 734–742.

Stockmeyer, L. 1974. The complexity of decision problems in automata theory and logic. PhD diss, MIT, Cambridge, MA.

Strachey, C. 1959. Time sharing in large fast computers. In International conference on information processing. Paris: UNESCO, NS, ICIP, 8.2.19.

Strassen, Volker. 1969. Gaussian elimination is not optimal. Numerische Mathematik 13 (4): 354–356.

Struik, Dirk J. 1969. A source book in mathematics, 1200–1800. Cambridge, MA: Harvard University Press.

Sutherland, Ivan Edward. 1963. Sketchpad: a manmachine graphical communication system, Technical Report 296, MIT Lincoln Laboratory, Lexington, MA.

Tarjan, Robert. 1975. Efficiency of a good but not linear set union algorithm. J. ACM 22 (2): 215–225.

Teager, Herbert. 1962. Real-time, time-shared computer project. Comm. ACM 5 (1).

Teager, Herbert, and John McCarthy. 1959. Time-shared program testing. In ACM annual conference/annual meeting: Preprints of papers presented at the 14th national meeting of the Association for Computing Machinery: Cambridge, Massachusetts; 01–03 Sept. 1959, 1–2.

Thornton, J. E. 1970. Design of a computer: the Control Data 6600. Glenview, IL: Scott, Foresman.

Titchmarsh, E. C. 1951. The theory of the Riemann zetafunction. Oxford: Clarendon Press.

Turing, Alan M. 1936. On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society s2–42 (1): 230–265. http://doi.org/10.1112/plms/s2-42.1.230.

Turing, Alan M. 1938. On computable numbers, with an application to the Entscheidungsproblem. A correction. Proceedings of the London Mathematical Society 43 (1): 544–546.

Turing, Alan M. 1945. Proposed electronic calculator, Technical report, National Physical Laboratory. http://www.alanturing.net/turing_archive/archive/p/p01/p01.php.

Turing, Alan M. 1950. Computing machinery and intelligence. Mind 59 (236): 433–460. https://doi.org/10.1093/mind/LIX.236.433.

Unknown. 1906. The most remarkable boy in the world. World Magazine, 7 October 1906: 1–3.

Uttley, A. M. 1956. Conditional probability machines and conditional reflexes. In Automata studies, eds. Claude E. Shannon and J. McCarthy, 253–275. Princeton, NJ: Princeton University Press.

Van Heijenoort, Jean. 1967. From Frege to Gödel; a source book in mathematical logic, 1879–1931. Cambridge, MA: Harvard University Press.

van Vlissingen, Rogier F., and Edsger W. Dijkstra. 1985. Interview with Prof. Dr. Edsger W. Dijkstra. https://www.cs.utexas.edu/users/EWD/misc/vanVlissingenInterview.html.

Vinogradov, I. M. 1954. The method of trigonometrical sums in the theory of numbers. Geneva: Interscience Publishers. http://hdl.handle.net/2027/mdp.39015000961329.

von Neumann, J. 1951. The general and logical theory of automata. In Cerebral mechanisms in behavior; the Hixon symposium, ed. Lloyd A. Jeffress, 1–41. New York: Wiley.

von Neumann, J. 1956. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In Automata studies, eds. Claude E. Shannon and John McCarthy, 43–98. Princeton, NJ: Princeton University Press.

von Neumann, J. 1993. First draft of a report on the EDVAC. IEEE Annals of the History of Computing 15 (4): 27–75.

von Neumann, J. 2000. The computer and the brain, 2nd edn. Mrs. Hepsa Ely Silliman memorial lectures. New Haven, CT: Yale University Press.

Walden, David. 1972. A system for interprocess communication in a resource sharing computer network. Comm. ACM 15 (4): 221–230.

Warshall, Stephen. 1962. A theorem on boolean matrices. J. ACM 9 (1): 11–12.

Webster, Noah. 1959. New international dictionary of the English language, 2nd edn. Springfield, MA: G. & C. Merriam.

Weizenbaum, J. 1963. Symmetric list processor. Comm. ACM 6 (9): 524–536.

Weizenbaum, J. 1966. ELIZA–A computer program for the study of natural language communication between man and machine. Comm. ACM 9 (1): 36–45. https://doi.org/10.1145/365153.365168.

Weizenbaum, J. 1976. Computer power and human reason: from judgment to calculation. San Francisco: W. H. Freeman.

Wells, H. G. 1938. World brain. London: Methuen.

Whitehead, Alfred North, and Bertrand Russell. 1910. Principia mathematica,. Cambridge: Cambridge University Press.

Wiener, Norbert. 1947. A scientist rebels. Atlantic 179: 31.

Wiener, Norbert. 1948. Cybernetics: or, control and communication in the animal and the machine. New York: J. Wiley.

Wiener, Norbert. 1950. The human use of human beings: cybernetics and society. Boston: Houghton Mifflin.

Wiener, Norbert. 1960. Some moral and technical consequences of automation. Science 131 (3410): 1355–1358.

Wilkes, Maurice V. 1972. Time-sharing computer systems, 2nd edn. Computer monographs; 5. New York: Elsevier.

Wilkes, Maurice V. 1981. The best way to design an automatic calculating machine. Microprocessing and Microprogramming 8.

Wilkes, Maurice V. 1986. The genesis of microprogramming. Annals of the History of Computing 8 (2): 116–126.

Willard, D. G. 1973. A sophisticated digital cable communications system. In Proc. National Telecommunications Conference.

Winograd, S. 1968. A new algorithm for inner product. IEEE Transactions on Computers C-17 (7): 693–694.

Wirth, N. 1971. The programming language Pascal. Acta Informatica 1 (1): 35–63.

Wirth, Niklaus, and C. Hoare. 1966. A contribution to the development of ALGOL. Comm. ACM 9 (6): 413–432.

Wulf, W., and R. Levin. 1972. C.mmp–a multi-miniprocessor. In AFIPS Conf. Proc., 1972 NCC, Vol. 41, 551–559. Montvale, NJ.

Zunde, Pranas, and Vladimir Slamecka. 1967. Distribution of indexing terms for maximum efficiency of information transmission. American Documentation 18 (2): 104–108.

Предметный указатель

М

Н

О

П

Р

С

Т

У

Ф

Х

Ц

Ч

Ш

Э

Ю

Я

Книги издательства «ДМК ПРЕСС»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).
Адрес: г. Москва, пр. Андропова, 38, оф. 10;
тел.: **(499) 782-38-89**, электронная почта: **books@alians-kniga.ru**.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **http://www.galaktika-dmk.com/**.

Под редакцией Гарри Р. Льюиса

Идеи, определившие облик информатики

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Зам. главного редактора *Сенченкова Е. А.*
Перевод *Слинкин А. А.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 50,05. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**