ВЫСШЕЕ
ОБРАЗОВАНИЕ

В. Б. Кудрявцев, Э. Э. Гасанов,
А. С. Подколзин

КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ ЛОГИЧЕСКИХ ПРОЦЕССОВ

Учебник
2-е издание

Курс с онлайн–оцениванием

УМО ВО
РЕКОМЕНДУЕТ

Юрайт
издательство

В. Б. Кудрявцев, Э. Э. Гасанов, А. С. Подколзин

КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ ЛОГИЧЕСКИХ ПРОЦЕССОВ

УЧЕБНИК ДЛЯ ВУЗОВ

2-е издание, переработанное и дополненное

Рекомендовано Учебно-методическим отделом высшего образования в качестве учебника для студентов высших учебных заведений, обучающихся по математическим и инженерно-техническим направлениям

Книга доступна на образовательной платформе «Юрайт» urait.ru, а также в мобильном приложении «Юрайт.Библиотека»

Москва ▪ Юрайт ▪ 2024

УДК 519.71((075.8)
ББК 22.18я73
К88

***Автор*:**

Кудрявцев Валерий Борисович — доктор физико-математических наук, академик РАЕН, академик АТН РФ;

Гасанов Эльяр Эльдарович — доктор физико-математических наук, профессор кафедры математической теории интеллектуальных систем Московского государственного университета имени М. В. Ломоносова;

Подколзин Александр Сергеевич — доктор физико-математических наук, профессор кафедры математической теории интеллектуальных систем Московского государственного университета имени М. В. Ломоносова, академик АТН РФ.

***Рецензенты*:**

Бабин Д. Н. — доктор физико-математических наук, профессор кафедры математической теории интеллектуальных систем отделения математики механико-математического факультета Московского государственного университета имени М. В. Ломоносова;

Козлов В. Н. — доктор физико-математических наук, профессор Московского государственного университета имени М. В. Ломоносова.

Кудрявцев, В. Б.

К88 Компьютерное моделирование логических процессов : учебник для вузов / В. Б. Кудрявцев, Э. Э. Гасанов, А. С. Подколзин. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2024. — 143 с. — (Высшее образование). — Текст : непосредственный.

ISBN 978-5-534-15336-1

Курс разработан на основе специальных курсов «Компьютерный решатель математических задач», «Интеллектуальные системы», читаемых на кафедре математической теории интеллектуальных систем механико-математического факультета МГУ имени М. В. Ломоносова.

Рассматриваются два подхода к моделированию логических процессов: стандартный подход с использованием формальных логик и подход, который пытается повторить логику рассуждений человека. В рамках первого подхода приводятся понятия и результаты исчисления высказываний и исчисления предикатов. Второй подход иллюстрируется на примере компьютерного решателя математических задач.

Для студентов, аспирантов и специалистов в области математической кибернетики, дискретной математики и информатики.

УДК 519.71((075.8)
ББК 22.18я73

ISBN 978-5-534-15336-1

Оглавление

Введение

Данный курс является третьим в серии курсов авторов по теории интеллектуальных систем и посвящен моделированию самого сложного блока интеллектуальной системы — блоку управления, или блоку принятия решения. Рассматриваются два подхода к моделированию логических процессов: стандартный подход с использованием формальных логик и подход, который пытается повторить логику рассуждений человека.

В рамках первого подхода приводятся понятия и результаты исчисления высказываний и исчисления предикатов.

Второй подход иллюстрируется на примере компьютерного решателя математических задач.

Решатель математических задач имеет в качестве среды класс задач по элементарной алгебре, тригонометрии, математическому анализу и многим другим разделам математики. Процесс его работы составляет поиск решения предложенной задачи, а результатом этой работы являются ход решения задачи и ответ, если таковые достижимы решателем, и отказ от решения, если последнее невозможно для решателя.

Базы данных и знаний решателя задач включают список стандартных приемов тождественных преобразований выражений алгебры и тригонометрии, основных теорем из этих разделов, а также логических операций вывода.

Самым сложным здесь является блок управления, принцип работы которого состоит в эвристической оптимальности извлечения приемов из баз данных и знаний, обеспечивающий в определенном смысле градиентность последовательности примененных приемов, что существенно сокращает перебор вариантов вывода.

В этом блоке реализуется новая идея, позволившая обойти неэффективные попытки использовать для подобных целей логико-аксиоматический подход; упомянем в этой связи General Problem Solver, Mathematika и др.

Эта интеллектуальная система, созданная А. С. Подколзиным, показала высокую эффективность, справляясь за секунды с большинством задач из известных учебников. Решающая способность решателя такова, что он может «поступить» на механико-математический факультет МГУ и решать задачи на уровне студента-отлич-

ника практически по всем основным курсам, читаемым на механико-математическом факультете.

Решатель демонстрировался на международной выставке компьютеров и программных продуктов в г. Ганновере (ФРГ), на российских и международных конференциях и семинарах.

В данном курсе принципы работы решателя описываются на примере таких разделов математики, как алгебра множеств, элементарная алгебра и элементарная геометрия. Полный список разделов математики, охваченных решателем, значительно шире.

Предлагаемое изложение осуществлено авторами на основе курсов лекций, читаемых ими на протяжении ряда лет студентам, специализирующимся по кафедре математической теории интеллектуальных систем (МаТИС) механико-математического факультета МГУ имени М. В. Ломоносова.

В результате изучения материалов курса студент должен:

знать

— основные понятия исчисления высказываний и исчисления предикатов;

— принципы управления выводом следствий;

— процедуру автоматического доказательства теорем;

— принципы компьютерного моделирования логических процессов;

— схему функционирования компьютерного решателя задач;

уметь

— осуществлять проверку общезначимости формулы высказываний различными методами;

— доказывать утверждения логики высказываний;

— доказывать общезначимость формулы логики предикатов методом резолюций;

— решать различные задачи с помощью компьютерного решателя;

владеть

— навыками представления задач в компьютерном решателе;

— навыками проведения вычислений в компьютерном решателе;

— методами доказательств теорем.

Нам приятно поблагодарить весь коллектив кафедры МаТИС, чье постоянное внимание, доброжелательная критика и советы способствовали появлению этой книги. Авторы выражают особую благодарность Г. В. Бокову за редакцию книги на заключительном этапе ее создания.

Тема 1
ИСЧИСЛЕНИЕ ВЫСКАЗЫВАНИЙ

Формализация понятия задачи при разработке систем автоматического решения задач часто приводит к необходимости использования логического языка для представления обрабатываемой информации, а также связанной с этим языком формальной дедуктивной системы, определяющей допустимые процессы логического вывода. Возникающие здесь математические модели мы проиллюстрируем на двух типичных примерах — языке исчисления логики высказываний, а также языке исчисления логики предикатов.

По этой тематике может быть рекомендована книга [3].

1.1. Язык логики высказываний

При задании формального логического языка обычно следуют следующей схеме.

1. Указывается конечный либо бесконечный набор символов, образующих алфавит языка; при описании алфавита могут вводиться те или иные характеристики его символов, в частности определяться разбиения их на подклассы.

2. Вводится индуктивное описание множества правильных выражений языка — конечных последовательностей символов алфавита.

3. Индуктивным образом определяется интерпретация языка (либо множество допустимых интерпретаций), сопоставляющая каждому выражению языка обозначаемую им функцию либо объект из некоторой «области интерпретации».

Пункты 1 и 2 задают синтаксис логического языка, пункт 3 — его семантику.

В случае языка логики высказываний эта общая схема реализуется следующим образом.

1. *Алфавит языка логики высказываний* состоит из счетного списка переменных $x_1, x_2, \ldots$; двух логических констант И, Л; заданного набора логических связок (например, связок $\vee$, &, $\neg$, $\rightarrow$, $\sim$), а также скобок (,).

2. Правильные выражения языка логики высказываний, называемые *формулами логики высказываний*, вводятся при помощи следующего определения:

а) однобуквенное слово, состоящее из переменной или логической константы, есть формула логики высказываний;

б) если слова A и B суть формулы логики высказываний, то слова $(\neg A)$, $(A \vee B)$, $(A \,\&\, B)$, $(A \to B)$, $(A \sim B)$ суть формулы логики высказываний.

3. При задании *интерпретации формул логики высказываний* возможны два различных подхода — эти формулы могут рассматриваться либо как обозначения функции алгебры логики, либо как обозначения единичных высказываний, имеющих определенное истинностное значение. В обоих случаях определяемый формулой объект (функция либо истинностное значение) вводится индуктивным образом, по шагам построения формулы, с использованием хорошо известных таблиц истинности для основных логических связок. В первом случае базис индукции сопоставляет каждой однобуквенной формуле x_i тождественную функцию; во втором случае — некоторое конкретное значение из множества {И, Л}.

Для большей определенности будем ниже рассматривать второй случай. Заметим, что при этом возникает множество различных интерпретаций языка логики высказываний, отличающихся друг от друга лишь сопоставлением переменным алфавита x_1, x_2, … различных истинностных констант.

Таким образом, интерпретацией языка логики высказываний назовем отображение I, определенное на множестве переменных x_1, x_2, … и сопоставляющее каждой переменной x_i некоторую истинностную константу a_i; $a_i \in$ {И, Л}. Расширим это отображение на множество формул. Значение $I(F)$ формулы F в интерпретации I определяем индукцией по построению формулы F:

1) если F есть переменная x_i, то $I(F)$ есть $I(x_i)$;

2) I(И) = И, I(Л) = Л;

3) если F имеет вид $(\neg G)$ и уже определено $I(G) = b$, то $I(F)$ есть истинностная константа, отличная от b;

4) если F имеет вид $(G_1 \vee G_2)$ и уже определены $I(G_1) = b_1$, $I(G_2) = b_2$, то $I(F)$ есть константа И, если хотя бы одно из b_1, b_2 равно И, иначе $I(F)$ есть Л;

5) если F имеет вид $(G_1 \,\&\, G_2)$ и уже определены $I(G_1) = b_1$, $I(G_2) = b_2$, то $I(F)$ есть константа И, если b_1, b_2 равны И, иначе $I(F)$ есть Л;

6) если F имеет вид $(G_1 \to G_2)$ и уже определены $I(G_1) = b_1$, $I(G_2) = b_2$, то $I(F)$ есть константа Л, если b_1 равно И, а b_2 равно Л; иначе $I(F)$ есть И;

7) если F имеет вид $(G_1 \sim G_2)$ и уже определены $I(G_1) = b_1$, $I(G_2) = b_2$, то $I(F)$ есть константа И, если $b_1 = b_2$, иначе $I(F)$ есть Л.

Если F — формула, x_1, …, x_n — все переменные формулы F, I — интерпретация, то для вычисления значения $I(F)$ нам достаточно знать только интерпретацию переменных x_1, …, x_n, а интерпрета-

ция остальных переменных нам не нужна. Если I — интерпретация, при которой $I(x_1) = a_1, \ldots, I(x_n) = a_n$, то будем писать $I = (a_1, \ldots, a_n)$.

Введем для логики высказываний несколько общих понятий, связанных с интерпретацией формул и являющихся типичными для логических языков.

Моделью формулы f будем называть произвольную интерпретацию языка логики высказываний, в которой f имеет значение И.

Формулу, имеющую хотя бы одну модель, назовем *выполнимой*.

Формулу, не имеющую ни одной модели, назовем *невыполнимой*.

Формулу, для которой каждая интерпретация языка логики высказываний является моделью, назовем *общезначимой*.

В качестве примеров отметим, что формула $((x_1 \vee x_2) \rightarrow x_2)$ выполнима, но не общезначима, формула $((x_1 \,\&\, x_2) \rightarrow x_2)$ общезначима, а формула $(\neg(x_1 \rightarrow (x_2 \rightarrow x_1)))$ невыполнима.

В дальнейшем часто мы будем опускать скобки, подразумевая, что $\neg$ — самая приоритетная операция, а также в случае, когда порядок расстановки скобок несущественен.

Описание класса всех общезначимых формул является одной из наиболее важных задач, возникающих при изучении логических языков, и для него обычно используются следующие подходы.

1. Нахождение индуктивного описания класса всех общезначимых формул. Такое описание определяет в базисе индукции некоторое подмножество общезначимых формул (конечное либо бесконечное), называемых аксиомами, и указывает в индуктивном шаге перечень допустимых правил вывода, позволяющих получать новые общезначимые формулы исходя из ранее найденных. Логический язык вместе с индуктивным описанием такого рода образует *дедуктивную систему*.

2. Нахождение алгоритма, перечисляющего все общезначимые формулы: фактически обеспечивается при построении дедуктивной системы.

3. Нахождение алгоритма, распознающего общезначимые формулы в классе всех правильных выражений языка. В отличие от пунктов 1 и 2, описание такого типа удается найти лишь для весьма немногих логических языков, к числу которых относится и язык логики высказываний.

1.2 Полнота исчисления высказываний

Существует много различных дедуктивных систем для описания общезначимых формул логики высказываний, различающихся множеством используемых логических связок и выбором аксиом; приведем здесь одну из простейших таких систем, в которой рассматриваются лишь две логические связки: $\neg$, $\rightarrow$. Остальные логи-

ческие связки легко могут быть выражены через них и трактоваться как своего рода «сокращенные обозначения» для соответствующих формул со связками $\neg$ и $\to$. Аксиомы данной дедуктивной системы задаются при помощи так называемых *схем аксиом*.

Еще раз напомним, что в дальнейшем для упрощения записи формул мы будем опускать внешние скобки и скобки при отрицаниях, считая, что отрицание имеет наивысший приоритет.

A1. Если f, g — формулы, то формула

$$f \to (g \to f)$$

есть аксиома.

A2. Если f, g, h — формулы, то формула

$$(f \to (g \to h)) \to ((f \to g) \to (f \to h))$$

есть аксиома.

A3. Если f, g — формулы, то формула

$$(\neg f \to \neg g) \to ((\neg f \to g) \to f)$$

есть аксиома.

В действительности каждая из схем аксиом A1, A2, A3 определяет бесконечное множество аксиом, различающихся выбором формул f, g, h.

Правило вывода в этой дедуктивной системе одно — если уже получены формулы f и $f \to g$, то из них выводится формула g (согласно классификации, предложенной еще Аристотелем, это правило называется *modus ponens*).

Чтобы проиллюстрировать технику, применяемую при изучении дедуктивных систем, приведем схематические наброски доказательств некоторых важных свойств только что определенной дедуктивной системы.

Выводом формулы f из списка формул $\Gamma = g_1, \ldots, g_n$ будем называть произвольную последовательность формул $f_1, f_2, \ldots, f_m = f$, такую, что каждое f_i, $i = 1, \ldots, m$, есть либо аксиома, либо элемент списка Γ, либо получено по правилу *modus ponens* из некоторых f_j, f_k при $j, k \in \{1, \ldots, i-1\}$. Если существует вывод f из Γ, то обозначаем это обстоятельство посредством $\Gamma \vdash f$ (в частности, список Γ может быть пуст; те формулы, которые выводимы из пустого Γ, образуют *класс всех выводимых формул* нашей дедуктивной системы). Следующая лемма есть пример выводимой формулы.

Лемма 1.1. *Если f — формула, то $\vdash f \to f$.*

Доказательство. Следующая последовательность формул, представляющая собой вывод из пустого Γ, доказывает утверждение:

$$(f \to ((g \to f) \to f)) \to ((f \to (g \to f)) \to (f \to f)),$$

$$f \to ((g \to f) \to f),\ (f \to (g \to f)) \to (f \to f),$$

$$f \to (g \to f), f \to f.$$

Первый ее элемент — аксиома, возникающая по схеме аксиом A2; второй и четвертый элементы — аксиомы по A1; третий и пятый элементы получены применением правила вывода. □

Имеет место следующее важное утверждение, известное как теорема дедукции Эрбрана.

Теорема 1.1 (дедукции Эрбрана [12]). *Если* Γ *— список формул и* f, g *— формулы, то из* $\Gamma, f \vdash g$ *вытекает* $\Gamma \vdash f \to g$.

Доказательство. Данное утверждение доказывается индукцией по длине n вывода $g_1, \ldots, g_n = g$ формулы g из Γ, f.

Базис индукции. Если $n = 1$, то g — либо аксиома, либо элемент списка Γ, f. Если g — аксиома или элемент списка Γ, то вывод $f \to g$ из Γ имеет вид $g \to (f \to g), g, f \to g$. Если же $g = f$, то используем приведенный выше вывод $f \to f$ из пустого списка.

Индуктивный переход. Если $n > 1$ и g — аксиома либо элемент списка Γ, f, то поступаем так же, как выше. Наконец, если g получено по правилу *modes ponens* из формул g_i, g_j $(i, j < n)$, то согласно предположению индукции имеем $\Gamma \vdash f \to g_i$, $\Gamma \vdash f \to g_j$. Заметим, что одна из формул g_i, g_j, например g_j, должна иметь вид $g_i \to g$. Для получения вывода $f \to g$ из Γ теперь достаточно рассмотреть последовательность, образованную расположенными подряд выводами $f \to g_i, f \to g_j$ из Γ, и присоединить к ней в конце формулы $(f \to (g_i \to g)) \to ((f \to g_i) \to (f \to g))$, $(f \to g_i) \to (f \to g), f \to g$. Первая из этих формул есть аксиома, полученная по схеме аксиом A2; две последние получены применением правила вывода. □

Для доказательства того, что каждая общезначимая формула алгебры логики выводима в рассматриваемой нами дедуктивной системе, нам понадобятся следующие вспомогательные утверждения.

Лемма 1.2 (доказательство от противного). *Если* Γ *— список формул и* f, g *— формулы, то из* $\Gamma, \neg f \vdash g, \neg g$ вытекает $\Gamma \vdash f$.

Доказательство. Так как $\Gamma, \neg f \vdash g, \neg g$, то по теореме дедукции $\Gamma \vdash \neg f \to g, \neg f \to \neg g$.

Вывод f из Γ следующий:

$$(\neg f \to \neg g) \to ((\neg f \to g) \to f), (\neg f \to g) \to f, f.$$

Здесь сначала используется аксиома A3, а затем — дважды правило *modus ponens*. □

Лемма 1.3 (снятие двойного отрицания). *Если* f *— формула, то* $\vdash \neg\neg f \to f$.

Доказательство. Следующее утверждение очевидно:

$$\neg\neg f, \neg f \vdash \neg f, \neg\neg f.$$

Используя лемму 1.2, получаем $\neg\neg f \vdash f$. Отсюда по теореме дедукции имеем $\vdash (\neg\neg f \to f)$. □

Лемма 1.4. *Если f — формула, то* $\vdash f \to \neg\neg f$.

Доказательство. Используя лемму 1.3 , легко показать, что $\neg\neg\neg f \vdash \neg f$. В самом деле, вывод $\neg f$ из $\neg\neg\neg f$ следующий: $\neg\neg\neg f \to$ $\to \neg f$ (по лемме 3), $\neg\neg\neg f$, $\neg f$ (по правилу *modus ponens*). Следовательно, имеем $f, \neg\neg\neg f \vdash f, \neg f$. Используя лемму 1.2, получаем $f \vdash \neg\neg f$. Отсюда по теореме дедукции имеем $\vdash (f \to \neg\neg f)$. □

Доказательства следующих трех утверждений рекомендуются в качестве самостоятельных упражнений.

Лемма 1.5. *Если f, g — формулы, то* $\vdash \neg f \to (f \to g)$.

Лемма 1.6. *Если f, g — формулы, то* $\vdash f \to (\neg g \to \neg(f \to g))$.

Лемма 1.7. *Если f, g — формулы, то* $\vdash (f \to g) \to ((\neg f \to g) \to g)$.

Для произвольной формулы g и некоторой интерпретации I языка логики высказываний обозначим посредством $(g)_I$ формулу, совпадающую с g, если значение g в интерпретации I есть И, и совпадающую с $(\neg g)$ в противном случае, т. е.

$$(g)_I = \begin{cases} g, \text{если } I(g) = \text{И}, \\ (\neg g), \text{если } I(g) = \text{Л}. \end{cases}$$

Лемма 1.8 (об интерпретациях). *Пусть f — формула логики высказываний; $x_1, \ldots, x_n$ — все переменные входящие в f, и I — некоторая интерпретация языка логики высказываний. Тогда выполняется* $(x_1)_I, \ldots, (x_n)_I \vdash (f)_I$.

Доказательство. Это утверждение будем доказывать индукцией по числу m логических связок в f.

Базис индукции. Если $m = 0$, то f совпадает с x_1 и утверждение сводится к очевидному факту $(x_1)_I \vdash (x_1)_I$.

Индуктивный переход. Пусть $m > 0$. Рассмотрим следующие два случая.

1. f имеет вид $\neg h$. Этот случай разбивается на следующие подслучаи.

А. Если $I(h) = $ И, т. е. $(h)_I = h$, то $I(f) = $ Л и, значит, $(f)_I = \neg\neg h$. По предположению индукции имеем $(x_1)_I, \ldots, (x_n)_I \vdash h$. По лемме 1.4 имеем $h \to \neg\neg h$. Применяя *modus ponens*, получаем $\neg\neg h$.

Б. Если $I(h) = $ Л, т. е. $(h)_I = \neg h$, то $I(f) = $ И или $(f)_I = f = \neg h$, и утверждение сводится к предположению индукции.

2. f имеет вид $h_1 \to h_2$. Этот случай разбивается на следующие подслучаи.

А. Если $I(h_1) = $ Л, т. е. $(h_1)_I = \neg h_1$, то $(f)_I = f = h_1 \to h_2$. По предположению индукции имеем $(x_1)_I, \ldots, (x_n)_I \vdash \neg h_1$. По лемме 1.5 имеем $\neg h_1 \to (h_1 \to h_2)$. Применяя *modus ponens*, получаем $h_1 \to h_2$.

Б. Если $I(h_2) = $ И, т. е. $(h_2)_I = h_2$, то $(f)_I = f = h_1 \to h_2$. По аксиоме А1 имеем $h_2 \to (h_1 \to h_2)$. По предположению индукции $(x_1)_I, \ldots,$ $(x_n)_I \vdash h_2$. Применяя *modus ponens* получаем $h_1 \to h_2$.

В. В случае $I(h_1) = $ И и $I(h_2) = $ Л имеем $I(f) = $ Л, $(h_1)_I = h_1$, $(h_2)_I = \neg h_2$ и $(f)_I = \neg f = \neg(h_1 \to h_2)$. По предположению индукции

$$(x_1)_I, \ldots, (x_n)_I \vdash h_1, \neg h_2.$$

По лемме 1.6 имеем

$$h_1 \to (\neg h_2 \to \neg(h_1 \to h_2))$$

Дважды применяя *modus ponens*, получаем $\neg(h_1 \to h_2)$. □

Теорема 1.2 (о полноте исчисления высказываний). *Формула логики высказываний является общезначимой тогда и только тогда, когда она выводима в рассматриваемой дедуктивной системе со схемами аксиом* A1, A2, A3 *и правилом вывода modus ponens.*

Доказательство. *Достаточность.* Если f и $(f \to g)$ — общезначимые формулы, то понятно, что и формула g — общезначимая. Следовательно, правило *modus ponens* из общезначимых формул выводит общезначимые. Поскольку, как легко видеть, схемы аксиом задают общезначимые формулы, то все выводимые формулы — общезначимы.

Необходимость. Будем обозначать посредством f^σ, где $\sigma \in \{$И, Л$\}$, формулу f в случае $\sigma = $ И, и формулу $(\neg f)$ в случае $\sigma = $ Л, т. е.

$$f^\sigma = \begin{cases} f, \text{если } \sigma = \text{И}, \\ (\neg f), \text{если } \sigma = \text{Л}. \end{cases}$$

Пусть f — произвольная общезначимая формула логики высказываний и $x_1, \ldots, x_n$ — все входящие в нее переменные. Покажем индукцией по k, что для любого целого неотрицательного k, $k \le n$, и любого набора $(\sigma_1, \ldots, \sigma_k)$ логических констант И, Л выполняется $x_1^{\sigma_1}, \ldots, x_k^{\sigma_k} \vdash f$.

Базис индукции. $k = n$, т. е. задана интерпретация $I = (\sigma_1, \ldots, \sigma_n)$. Так как $(x_i)_I = x_i^{\sigma_i}$, то по лемме 1.8 имеем $x_1^{\sigma_1}, \ldots, x_n^{\sigma_n} \vdash (f)_I = f$.

Индуктивный переход. Если данное утверждение уже доказано для некоторого k, $k > 0$, то рассмотрим произвольный набор $(\sigma_1, \ldots, \sigma_{k-1})$ констант И, Л. По предположению индукции имеем

$$x_1^{\sigma_1}, \ldots, x_{k-1}^{\sigma_{k-1}}, x_k \vdash f \text{ и } x_1^{\sigma_1}, \ldots, x_{k-1}^{\sigma_{k-1}}, \neg x_k \vdash f.$$

По теореме дедукции получаем

$$x_1^{\sigma_1}, \ldots, x_{k-1}^{\sigma_{k-1}} \vdash x_k \to f \text{ и } x_1^{\sigma_1}, \ldots, x_{k-1}^{\sigma_{k-1}} \vdash \neg x_k \to f.$$

По лемме 1.7 имеем $(x_k \to f) \to ((\neg x_k \to f) \to f)$. Дважды воспользовавшись правилом *modus ponens*, получаем $x_1^{\sigma_1}, \ldots, x_k^{\sigma_k} \vdash f$. Шаг индукции, таким образом, доказан.

При $k = 0$ окончательно получим $\vdash f$. □

1.3 Алгоритмы распознавания общезначимости формул логики высказываний

В случае логики высказываний проверка общезначимости формулы, имеющей n переменных, может быть осуществлена путем перебора всех 2^n возможных наборов логических констант, сопоставляемых этим переменным при различных интерпретациях, и установления того, что все значения формулы на этих наборах равны И. Таким образом, здесь имеется не только алгоритм перечисления всех общезначимых формул, но и алгоритм проверки общезначимости. Однако даже для такого простейшего случая остается проблема уменьшения трудоемкости распознавания общезначимости по сравнению с практически малоприемлемым для сколь-нибудь значительных n перебором всех 2^n наборов значений переменных.

К проблеме распознавания общезначимости формул логики высказываний тесно примыкает проблема решения систем логических уравнений, сводящаяся к нахождению всех наборов значений переменных, при которых заданная формула алгебры логики принимает значение И. Последняя проблема часто встречается в различных прикладных задачах дискретной оптимизации и диагностики, и поиску эффективных процедур ее решения, учитывающих в своих эвристических решающих правилах статистические особенности рассматриваемого класса задач, посвящено большое количество исследований. Мы ограничимся здесь лишь несколькими простейшими процедурами подобного рода (применительно к задаче распознавания общезначимости), поскольку они позволят нам развить технику доказательства теорем в логике высказываний, представляющую собой упрощенный аналог техники, развиваемой далее для автоматического доказательства теорем в логике предикатов.

Прежде всего опишем *алгоритм Квайна*, осуществляющий проверку общезначимости формулы f логики высказываний при помощи построения так называемого семантического дерева. Корню этого дерева — исходной вершине — приписывается формула f. Пусть уже построено некоторое подмножество вершин семантического дерева, каждой из которых сопоставлена некоторая формула. Если каждой концевой вершине данного дерева оказалась сопоставлена константа И, то процесс завершается и формула f является общезначимой.

Если некоторой концевой вершине дерева сопоставлена константа Л, то формула f не общезначима и процесс также завершается. В противном случае найдется концевая вершина v, которой сопоставлена формула, отличная от констант И, Л. Если эта формула

g не содержит переменных, то ее можно преобразовать в логическую константу, используя тождества:

$$\neg И = Л;\ \neg Л = И;\ И \vee f = И;\ Л \vee f = f;$$

$$И\ \&\ f = f;\ Л\ \&\ f = Л;\ И \to f = f;\ Л \to f = И;$$

$$f \to И = И;\ f \to Л = \neg f;\ И \sim f = f;\ Л \sim f = \neg f.$$

Если же она содержит хотя бы одну переменную x, то выполняются следующие действия:

а) находятся результаты g_1 и g_2 подстановки в формулу g вместо переменной x соответственно констант И и Л;

б) находятся результаты g_1' и g_2' упрощения формул g_1 и g_2 при помощи перечисленных выше тождеств, применяемых до тех пор, пока это возможно;

в) вводятся две новые вершины v_1 и v_2 семантического дерева, которым приписываются, соответственно, формулы g_1' и g_2'. К этим вершинам от вершины v проводятся ребра, отмеченные соответственно выражениями $x = И$ и $x = Л$.

Далее повторяется описанный выше процесс рассмотрения концевых вершин семантического дерева.

В данной процедуре остался недоопределенным выбор конкретной переменной x формулы g, по которой проводится «разбор случаев». Этот выбор в прикладных программах, решающих системы логических уравнений путем построения семантических деревьев, осуществляется на основании различных эвристических решающих правил. Простейшим таким правилом является выбор переменной, имеющей наибольшее число вхождений в g, для получения наиболее сильного упрощения при переходе к g_1' и g_2', другим полезным соображением является такой выбор переменных x, при котором сопоставленные концевым вершинам семантического дерева формулы g оказывались бы представимы, например, как $h_1\ \&\ h_2$ либо $h_1 \vee h_2$, где формулы h_1 и h_2 не имеют общих переменных. В этом случае вместо построения ветви дерева для $h_1\ \&\ h_2$ ($h_1 \vee h_2$, $h_1 \to h_2$ и т. п.) можно было бы рассмотреть независимо формируемые деревья для h_1, h_2 и из анализа их концевых вершин сделать вывод относительно общезначимости g.

Пример 1.1

С помощью алгоритма Квайна убедиться в общезначимости формулы, полученной из схемы аксиом А2:

$$(x_1 \to (x_2 \to x_3)) \to ((x_1 \to x_2) \to (x_1 \to x_3)). \qquad (1.1)$$

Решение. Дерево разбора случаев для этой формулы приведено на рис. 1.1.

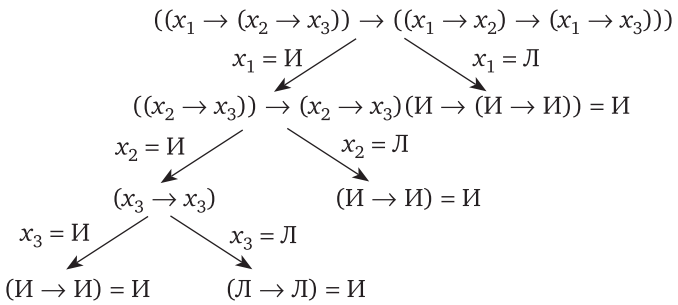

Рис. 1.1. **Алгоритм Квайна**

В прикладных ситуациях логические языки редко используются во всем многообразии своих синтаксических возможностей. Обычно их формулы приводятся эквивалентными преобразованиями к тому или иному «стандартному виду», который и сохраняется в процессе обработки информации. С точки зрения логики высказываний наиболее типичной логической стандартной формой, отражающей тенденцию к описанию ситуаций в виде конъюнкции условий, является конъюнктивная нормальная форма. Формула, представленная в этой форме, имеет вид A_1 & … & A_n, $n \geq 1$, где каждое A_i (i = 1, …, n) — формула вида $(B_{i1} \vee \ldots \vee B_{im_i})$, $m_i \geq 1$, причем $(B_{i1}, \ldots, B_{im_i})$ — переменные или отрицания переменных.

Более точно, для определения конъюнктивной нормальной формы введем сначала понятие дизъюнкта. Если $x_1, \ldots, x_m$ — различные переменные, $m \geq 1$, то формулу вида $(x_1^{\sigma_1} \vee \ldots \vee x_m^{\sigma_m})$ назовем *дизъюнктом*. Логическую константу Л по определению так же считаем дизъюнктом.

Если $A_1, \ldots, A_n$ — различные дизъюнкты ($n \geq 1$), то формула (A_1 & … & A_n) называется *конъюнктивной нормальной формой*.

Произвольную формулу логики высказываний можно преобразовать к виду конъюнктивной нормальной формы, используя следующие эквивалентные преобразования:

а) логические связки $\rightarrow$, $\sim$ устраняются при помощи тождеств

$$a \rightarrow b = \neg a \vee b;\ a \sim b = (a \,\&\, b) \vee (\neg a \,\&\, \neg b); \qquad (1.2)$$

б) отрицания в формуле «опускаются до переменных» при помощи тождеств

$$\neg\neg a = a;\ \neg(a \vee b) = \neg a \,\&\, \neg b;\ \neg(a \,\&\, b) = \neg a \vee \neg b; \qquad (1.3)$$

в) применяются преобразования дистрибутивности:

$$a \vee (b \,\&\, c) = (a \vee b) \,\&\, (a \vee c); \qquad (1.4)$$

г) устраняются повторные вхождения переменных в дизъюнктивных подформулах, а также константы И, Л (если сама формула не есть И или Л):

$$a \vee a = a;\ a \vee \neg a = \text{И};\ a \vee \text{И} = \text{И};$$
$$a \,\&\, \text{И} = a;\ a \vee \text{Л} = a;\ a \,\&\, \text{Л} = \text{Л}; \tag{1.5}$$

д) устраняются одинаковые дизъюнкты:

$$a \,\&\, a = a. \tag{1.6}$$

В процедурах автоматического доказательства теорем часто применяется метод доказательства «от противного». В этом случае для доказательства общезначимости формулы f переходят к рассмотрению формулы $\neg f$ и пытаются доказать ее невыполнимость.

Опишем *модифицированный алгоритм Квайна*, предназначенный для установления невыполнимости формулы логики высказываний g, преобразованной к виду конъюнктивной нормальной формы. Вершинам семантического дерева в этом случае будут сопоставляться не формулы, а их множества дизъюнктов.

Первоначально вводим корень семантического дерева и сопоставляем ему множество дизъюнктов формулы g.

Пусть уже построена часть семантического дерева.

Если концевой вершине дерева сопоставлено пустое множество дизъюнктов (пустое множество соответствует логической константе И), то формула g выполнима и алгоритм прекращает работу.

Если каждой концевой вершине семантического дерева соответствует множество дизъюнктов, содержащее дизъюнкт Л, то формула g невыполнима и алгоритм прекращает работу.

Если некоторой концевой вершине v сопоставлено непустое множество дизъюнктов S, не содержащее дизъюнкта Л, то в S входят дизъюнкты с переменными. Выбираем одну из таких переменных x и разбиваем S на три подкласса: S_1 — все дизъюнкты, в которые x входит без внешнего отрицания, S_2 — все дизъюнкты, в которые x входит с отрицанием, S_3 — все дизъюнкты в которых x не встречается. Далее рассматриваем множество дизъюнктов $S_x = \{d \mid d \vee \neg x \in S_2\}$ (если $\neg x \in S_2$, то здесь берется $d = \text{Л}$) и множество дизъюнктов $S_{\neg x} = \{d \mid d \vee x \in S_1\}$ (если $x \in S_1$, то здесь берется $d = \text{Л}$).

Нетрудно видеть, что невыполнимость конъюнкции дизъюнктов списка S эквивалента одновременной невыполнимости конъюнкций дизъюнктов списков $S_x \cup S_3$ и $S_{\neg x} \cup S_3$. Поэтому для продолжения построения семантического дерева вводим две новые вершины v_1 и v_2, которым сопоставляем, соответственно, множества дизъюнктов $S_x \cup S_3$ и $S_{\neg x} \cup S_3$, причем проводим к v_1 и v_2 ребра от v, отмеченные соответственно выражениями $x = \text{И}$ и $x = \text{Л}$.

Далее повторяется описанный выше процесс рассмотрения концевых вершин семантического дерева.

Пример 1.2

С помощью модифицированного алгоритма Квайна убедиться в общезначимости формулы (1.1) (см. пример 1.1).

Решение. После устранения логических связок $\to$ мы получим формулу

$$g = (\neg(\neg x_1 \vee \neg x_2 \vee x_3)) \vee (\neg(\neg x_1 \vee x_2)) \vee \neg x_1 \vee x_3.$$

Далее переходим к рассмотрению формулы $\neg g$, которая имеет вид

$$\neg g = (\neg x_1 \vee \neg x_2 \vee x_3) \,\&\, (\neg x_1 \vee x_2) \,\&\, x_1 \,\&\, (\neg x_3). \qquad (1.7)$$

Семантическое дерево для формулы $\neg g$ приведено на рис. 1.2.

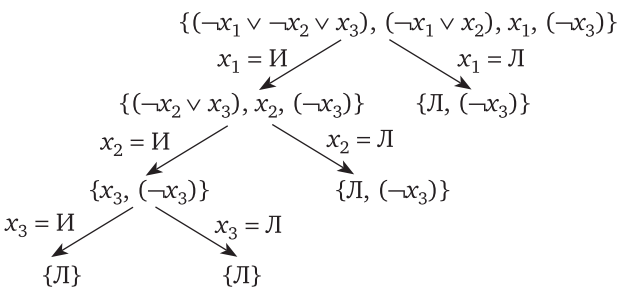

Рис. 1.2. **Модифицированный алгоритм Квайна**

Другой алгоритм распознавания невыполнимости конъюнктивной нормальной формы связан с рассмотрением специальной дедуктивной системы, использующей в качестве правила вывода так называемое *правило резолюции*. В этом случае конъюнктивная нормальная форма снова представляется как множество своих дизъюнктов, эти дизъюнкты и образуют перечень аксиом данной дедуктивной системы. Правило резолюции, будучи применено к двум дизъюнктам $x \vee B$ и $\neg x \vee C$, создает в качестве следствия новый дизъюнкт $B \vee C$ (здесь возможны вырожденные случаи отсутствия B либо C; если они оба отсутствуют, то результатом служит дизъюнкт Л). Достаточность применения данного правила вывода для распознавания невыполнимости гарантируется следующим утверждением.

Теорема 1.3. *Конъюнкция конечного семейства дизъюнктов S невыполнима тогда и только тогда, когда за конечное число применений правила резолюции из S выводится дизъюнкт* Л.

Доказательство. Доказывать будем индукцией по числу n переменных, встречающихся в дизъюнктах из S.

Базис индукции. Если $n = 0$, то утверждение очевидно, так как в этом случае непустое семейство дизъюнктов может содержать только логическую константу Л.

Индуктивный переход. Пусть утверждение уже доказано для некоторого $n \geq 0$.

Рассмотрим семейство S с $n + 1$ переменной: $x_1, x_2, \ldots, x_{n+1}$. Разобьем S на три подкласса: S_1 — все дизъюнкты, представимые в виде $x_{n+1} \vee a$, S_2 — все дизъюнкты, представимые в виде $\neg x_{n+1} \vee b$, S_3 — дизъюнкты, не содержащие x_{n+1}.

Дизъюнкты множества

$$R = \{a \vee b: x_{n+1} \vee a \in S_1, \neg x_{n+1} \vee b \in S_2\}$$

получены из дизъюнктов семейства S применением правила резолюции по переменной x_{n+1} (здесь допускаются вырожденные случаи, когда a либо b отсутствует, причем при одновременном отсутствии a и b в R включается константа Л).

Рассмотрим семейство дизъюнктов $R \cup S_3$ и покажем, что его невыполнимость эквивалентна невыполнимости S.

Если S выполнимо, то существует такая интерпретация I, что для любого дизъюнкта d из S выполняется $I(d) =$ И. Тогда понятно, что для любого дизъюнкта d' из S_3 выполняется $I(d') =$ И. Рассмотрим произвольный дизъюнкт $a \vee b$ из R, такой, что $x_{n+1} \vee a \in S_1$ и $\neg x_{n+1} \vee$ $\vee\, b \in S_2$. Если $I(x_{n+1}) =$ И, то из того, что $I(\neg x_{n+1} \vee b) =$ И, следует, что $I(b) =$ И. Если $I(x_{n+1}) =$ Л, то из того, что $I(x_{n+1} \vee a) =$ И, следует, что $I(a) =$ И. Следовательно $I(a \vee b) =$ И и $R \cup S_3$ выполнимо.

Пусть S невыполнимо, рассмотрим набор истинностных значений $\alpha_1, \ldots, \alpha_n$ переменных $x_1, \ldots, x_n$ и покажем, что на этом наборе хотя бы один из дизъюнктов семейства $R \cup S_3$ имеет значение Л, т. е. покажем, что для любой интерпретации $I = (\alpha_1, \ldots, \alpha_n)$ существует такой дизъюнкт d из $R \cup S_3$, что $I(d) =$ Л.

Так как S невыполнимо, то на наборе $\alpha_1, \ldots, \alpha_n$, И значений переменных $x_1, \ldots, x_n, x_{n+1}$ некоторый дизъюнкт d из S имеет значение Л, т. е. для интерпретации $I' = (\alpha_1, \ldots, \alpha_n,$ И) выполняется $I'(d) =$ Л. Если $d \in S_3$, то понятно, что $I(d) =$ Л и d является искомым дизъюнктом в $R \cup S_3$. Так как $I'(x_{n+1}) =$ И, то при $d \notin S_3$ остается единственная возможность $d \in S_2$, т. е. $d = \neg x_{n+1} \vee b$. При этом из того, что $I'(d) =$ Л, следует, что $I'(b) = I(b) =$ Л.

Аналогично из-за невыполнимости S на наборе $\alpha_1, \ldots, \alpha_n$, Л значений переменных $x_1, \ldots, x_n, x_{n+1}$ некоторый дизъюнкт d' из S имеет значение Л, т. е. для интерпретации $I'' = (\alpha_1, \ldots, \alpha_n,$ Л) выполняется $I''(d') =$ Л. Снова, если $d' \in S_3$, то он и есть искомый, а в противном случае (так как $I''(x_{n+1}) =$ Л) необходимо $d' \in S_1$, т. е. $d' = x_{n+1} \vee a$ и, значит, $I''(a) = I(a) =$ Л. Следовательно, и дизъюнкт $d'' = a \vee b$, принадлежащий R, будет иметь на наборе $\alpha_1, \ldots, \alpha_n$ значений переменных $x_1, \ldots, x_n$ значение Л, т. е. $I(a \vee b) =$ Л. Это завершает доказательство эквивалентности невыполнимости S и $R \cup S_3$.

Но число переменных в $R \cup S_3$ не более n, т. е. для него согласно предположению индукции имеет место эквивалентность невыполнимости и существования вывода константы Л.

Если константа Л выводима из S, то либо дизъюнкт Л принадлежит S и тогда S невыполнимо, либо константа Л выводима из $R \cup S_3$ и тогда $R \cup S_3$ невыполнимо и, следовательно, S невыполнимо.

Если S невыполнимо, то невыполнимо $R \cup S_3$, и из $R \cup S_3$, а значит, и из S выводима константа Л.

Тем самым теорема доказана. □

Пример 1.3

С помощью правила резолюции убедиться в общезначимости формулы f, заданной выражением (1.1) (см. пример 1.1).

Решение. Так же как в примере 1.2, перейдем к формуле $\neg g$, заданной выражением (1.7). Исходное множество дизъюнктов имеет вид

$$\{\neg x_1 \vee \neg x_2 \vee x_3, \neg x_1 \vee x_2, x_1, \neg x_3\}.$$

Из $\neg x_1 \vee x_2$ и x_1 по правилу резолюции выводится x_2. Из $\neg x_1 \vee \neg x_2 \vee x_3$ и x_1 выводим $\neg x_2 \vee x_3$. Из $\neg x_2 \vee x_3$ и x_2 выводим x_3. И наконец, из x_3 и $\neg x_3$ получаем Л. Следовательно, формула $\neg g$ невыполнима.

Упражнения

1.1. Укажите, какие из следующих выражений являются правильными выражениями (формулами) логики высказываний:

а) x_1;
б) (x_2);
в) $x_1 \vee x_2$;
г) $(x_1 \vee x_2)$;
д) $x_1 x_2$;
е) $(x_1 \vee x_3)\ \&\ (x_1 \vee x_2)$;
ж) $((x_1 \vee \neg x_2 \vee x_3)\ \&\ (x_1 \vee x_2))$;
з) $((x_1 \vee (\neg x_2) \vee x_3)\ \&\ (x_1 \vee x_2))$;
и) $((x_1 \vee x_2)\ \&\ (x_1 \vee x_2))$.

1.2. Расставьте скобки в следующих выражениях (с учетом приоритета операций $\neg$, &, $\vee$, $\rightarrow$, $\sim$ и левой ассоциативности всех двухместных операций) так, чтобы получились правильные выражения логики высказываний:

а) $x_1 \vee \neg x_3 \rightarrow x_1\ \&\ x_2$;
б) $x_1 \vee x_2 \vee x_3\ \&\ x_4 \sim x_1$;
в) $\neg\neg x_1 \vee \neg x_2 \rightarrow x_1$;
г) $x_2 \rightarrow \neg x_1 \rightarrow x_3\ \&\ x_1$.

1.3. Укажите, какие из следующих формул логики высказываний общезначимы, а какие — выполнимы:

а) $(x_1 \vee x_2)$;
б) $(x_1 \vee (\neg x_1))$;
в) $(x_1\ \&\ (\neg x_1))$;
г) $(((\neg x_1) \vee (\neg x_2))\ \&\ ((x_2 \vee (\neg x_1))\ \&\ x_1))$;
д) $((x_1 \vee ((\neg x_2) \vee x_3))\ \&\ ((x_1 \vee x_2)\ \&\ (\neg x_3)))$.

1.4. Постройте таблицу истинности для следующих формул логики высказываний:

а) $(x_1 \to ((x_1 \,\&\, x_2) \vee (\neg x_3)))$;

б) $((x_1 \to x_2) \,\&\, (\neg((\neg x_2) \to (\neg x_1))))$;

в) x_2;

г) $(((x_1 \vee x_2) \,\&\, (x_3 \vee (\neg x_2))) \to (x_1 \vee x_3))$.

1.5. Для каждой выполнимой формулы из упражнений 1.3 и 1.4 найдите хотя бы одну ее модель.

1.6. Для каждой формулы из упражнений 1.3 и 1.4 проверьте, является ли она выполнимой, невыполнимой, общезначимой, с помощью метода Квайна, модифицированного метода Квайна и метода резолюции.

1.7. Докажите эквивалентность следующих формул логики высказываний:

а) $(x_1 \vee (x_1 \,\&\, x_2))$ и $(x_1 \vee x_2)$;

б) $(x_1 \to x_2)$ и $((\neg x_2) \to (\neg x_1))$;

в) $((\neg x_1) \to x_1)$ и $((x_2 \to (\neg x_2)) \sim x_1)$.

1.8. Докажите леммы 1.5, 1.6 и 1.7.

1.9. Пусть f, g, h — формулы логики высказываний. Докажите справедливость следующих утверждений:

а) $(f \to g), (g \to h) \vdash (f \to h)$ (правило силлогизма);

б) $(f \to (g \to h)), g \vdash (f \to h)$;

в) $\vdash ((\neg g \to \neg f) \to (f \to g))$;

г) $\vdash ((f \to g) \to (\neg g \to \neg f))$.

Тема 2
ИСЧИСЛЕНИЕ ПРЕДИКАТОВ

2.1. Язык логики предикатов

В случае языка логики предикатов наш алфавит будет состоять из следующих групп символов:

1) счетный список предметных переменных $x_1, x_2, \ldots$;
2) счетный список предметных констант $a_1, a_2, \ldots$;
3) счетный список функциональных символов $f_1, f_2, \ldots$;
4) счетный список предикатных символов $P_1, P_2, \ldots$;
5) логические константы И, Л;
6) логические связки $\neg, \vee, \&, \rightarrow, \sim$;
7) кванторы $\forall$ и $\exists$;
8) скобки «(», «)» и запятая «,».

Каждый функциональный либо предикатный символ характеризуется своей арностью — некоторым натуральным числом. При этом предполагается, что для каждого натурального n имеется бесконечно много как предикатных, так и функциональных символов арности n.

Правильные выражения языка логики предикатов разбиваются на два непересекающихся множества — множество термов и множество формул. Дадим сначала индуктивное определение *множества термов*.

Т1. Однобуквенное слово, состоящее из предметной переменной либо предметной константы, есть терм.

Т2. Если $t_1, \ldots, t_n$ — термы и g — функциональный символ арности n, то слово $g(t_1, \ldots, t_n)$ есть терм.

Индуктивное определение *формул логики предикатов* опирается на уже введенное понятие терма.

Ф1. Однобуквенное слово, состоящее из логической константы И либо Л, есть формула логики предикатов.

Ф2. Если $t_1, \ldots, t_n$ — термы и P — предикатный символ арности n, то слово $P(t_1, \ldots, t_n)$ есть формула логики предикатов (такие формулы будем называть *атомарными* или *атомами*).

Ф3. Если f и g — формулы логики предикатов, то слова $(\neg f)$, $(f \vee g)$, $(f \,\&\, g)$, $(f \rightarrow g)$, $(f \sim g)$ суть формулы логики предикатов.

Ф4. Если f — формула логики предикатов и x — предметная переменная, то слова $(\forall x f)$ и $(\exists x f)$ суть формулы логики предикатов.

Интерпретацией языка логики предикатов будем называть четверку (M, F_1, F_2, F_3), такую, что:

а) M — непустое множество, называемое областью интерпретации;

б) F_1 — функция, сопоставляющая каждой предметной константе некоторый элемент из M, т. е. F_1: $a_1, a_2, \ldots \to M$;

в) F_2 — функция, сопоставляющая каждому функциональному символу f арности n некоторую n-местную функцию, определенную на M и принимающую значения из M, т. е. $F_2(f)$: $M^n \to M$;

г) F_3 — функция, сопоставляющая каждому предикатному символу P арности n некоторый n-местный предикат определенный на M (функцию со значениями И, Л), т. е. $F_3(P)$: $M^n \to$ {И, Л}.

Если задана интерпретация $I = (M, F_1, F_2, F_3)$, то каждый терм t языка логики предикатов определяет в этой интерпретации некоторую функцию $I(t)$, определенную на M и принимающую значения из M, а каждая формула f — предикат $I(f)$, определенный на M (в вырожденных случаях $I(t)$ может отождествляться с элементом M, а $I(f)$ — с логической константой).

Здесь используется следующее индуктивное определение.

1. Если x — предметная переменная, то $I(x)$ есть тождественная функция от переменной x, определенная на M.

2. Если a — предметная константа, то $I(a)$ есть элемент $F_1(a)$ из M.

3. Если для термов $t_1, \ldots, t_n$ уже определены $I(t_1), \ldots, I(t_n)$, f — функциональный символ арности n и $\varphi = F_2(f)$ — функция от n переменных, отображающая M^n в M, то

$$I(f(t_1, \ldots, t_n)) = \varphi(I(t_1), \ldots, I(t_n)).$$

4. $I(\text{И}) = \text{И}$, $I(\text{Л}) = \text{Л}$.

5. Если для термов $t_1, \ldots, t_n$ уже определены $I(t_1), \ldots, I(t_n)$, P — предикатный символ арности n и $\pi = F_3(P)$ — предикат от n переменных, отображающий M^n в {И, Л}, то

$$I(P(t_1, \ldots, t_n)) = \pi(I(t_1), \ldots, I(t_n)).$$

6. Если для f и g уже определены $I(f)$ и $I(g)$, то $I((\neg f))$, $I((f \vee g))$, $I((f \,\&\, g))$, $I((f \to g))$, $I((f \sim g))$ получаются применением истинностных функций для $\neg$, $\vee$, &, $\to$, $\sim$ к $I(f)$ и $I(g)$, т. е. $I((\neg f)) = \neg I(f)$, $I((f \vee g)) = I(f) \vee I(g)$, $I((f \,\&\, g)) = I(f)$ & $I(g)$, $I((f \to g)) = I(f) \to I(g)$, $I((f \sim g)) = I(f) \sim I(g)$.

7. Если уже определен предикат $I(f) = \varphi(x_1, \ldots, x_n)$, то $I((\forall x_i f)) = \psi(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ — предикат, принимающий значение И на таких наборах $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n$ элементов M, что для каждого α_i из M значение $\varphi(\alpha_1, \ldots, \alpha_n)$ есть И. Аналогично, $I((\exists x_i f)) = \xi(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ — предикат, принимающий значение И на таких

наборах $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n$ элементов M, что существует α_i из M, для которого значение $\varphi(\alpha_1, \ldots, \alpha_n)$ есть И.

Пример 2.1

Пусть $I = (N, F_1, F_2, F_3)$ — такая интерпретация, что $F_2(f_1)(x_1) = x_1 + 1$, $F_2(f_2)(x_1, x_2) = x_1 + x_2$, $F_2(f_3)(x_1, x_2) = x_1 \cdot x_2$, $F_3(P_1)(x_1)$ — предикат, равный истине точно тогда, когда x_1 — нечетное число. Описать предикаты $I(A)$ и $I(B)$ для формул A и B, задаваемых выражениями

$$A = (\forall x_1 P_1(f_2(f_1(x_1), f_3(x_1, x_2)))),$$

$$B = (\exists x_1 P_1(f_2(f_1(x_1), f_3(x_1, x_2)))).$$

Решение. В соответствии с определением интерпретации имеем: $I(f_1(x_1)) = x_1 + 1$, $I(f_3(x_1, x_2)) = x_1 \cdot x_2$, $I(f_2(f_1(x_1), f_3(x_1, x_2))) = x_1 + 1 + x_1 \cdot x_2 = x_1 \cdot (x_2 + 1) + 1$. Предикат $I(P_1(f_2(f_1(x_1), f_3(x_1, x_2))))$. принимает значение «истина», только если $x_1 \cdot (x_2 + 1) + 1$ будет нечетным. Следовательно, если x_2 будет нечетным, то $x_1 \cdot (x_2 + 1) + 1$ будет нечетным независимо от значения x_1 и, значит, при нечетном x_2 для обеих формул A и B предикаты $I(A)$ и $I(B)$ принимают значение «истина». Если x_2 четно, то в зависимости от x_1 величина $x_1 \cdot (x_2 + 1) + 1$ может быть как четной, так и нечетной. Следовательно, при четном x_2 предикат $I(A)$ принимает значение «ложь», а предикат $I(B)$ принимает значение «истина».

Вхождение переменной x в формулу f логики предикатов, расположенное внутри подформул вида $(\forall xg)$ либо $(\exists xg)$, называется *связанным*, вхождения переменных, не являющиеся связанными, называются *свободными*. Переменная, имеющая хотя бы одно свободное вхождение в формулу f, называется *свободной переменной* f. Как нетрудно видеть, предикат $I(f)$, определенный выше, зависит лишь от свободных переменных f. Формула без свободных переменных называется *замкнутой*.

В формуле $\exists x_1((\forall x_2 P_1(x_2, x_1)) \vee P_2(x_2, x_1))$ первое вхождение переменной x_2 — связанное, а второе — свободное, оба вхождения переменной x_1 связанные. Тем самым у этой формулы одна свободная переменная — x_2. Формула $((\exists x_1 P_1(x_1)) \to (\forall x_2 P_2(x_2)))$ — пример замкнутой формулы.

Если f есть формула логики предикатов, x — свободная предметная переменная формулы f и t — терм, то через $S_t^x f$ обозначим результат подстановки в формулу f терма t вместо всех свободных вхождений переменной x.

Терм t называется *свободным для переменной* x_i *в формуле логики предикатов* f, если никакое свободное вхождение x_i в f не лежит в области действия никакого квантора Qx_j, где $Q \in \{\forall, \exists\}$; x_j — переменная, входящая в терм t, т. е. если терм t свободен для переменной x в формуле f, то его можно подставить в формулу f вместо переменной x и никакая переменная терма t не окажется связанной.

Например, терм $f_1(x_1, x_3)$ свободен для x_1 в формуле

$$((\forall x_2 P_1(x_1, x_2)) \to P_2(x_1))$$

но не свободен для x_1 в формуле

$$(\exists x_3(\forall x_2(P_1(x_1, x_2) \to P_2(x_1))))$$

Лемма 2.1 (об интерпретациях). *Если f есть формула логики предикатов, x — свободная предметная переменная формулы f и t — терм, свободный для переменной x в f, то для любой интерпретации I формула $I(S_t^x f)$ определяет предикат, получающийся подстановкой в $I(f)$ функции $I(t)$ вместо переменной x.*

Доказательство этой леммы может быть получено индукцией по построению формулы f и рекомендуется в качестве самостоятельного упражнения.

Если формула f определяет в интерпретации I предикат, тождественно равный И, то она называется *истинной* в I, а I в этом случае называется *моделью* для f.

Формулу логики предикатов f называем *общезначимой*, если она истинна во всех интерпретациях, и *выполнимой*, если хотя бы в одной интерпретации она определяет не тождественно ложный предикат.

Если формула $f_1 \,\&\, \ldots \,\&\, f_n \to f_0$ общезначима, то f_0 называется *логическим следствием* формул $f_1, \ldots, f_n$, если формула $f_1 \sim f_2$ общезначима, то формулы f_1 и f_2 называются *эквивалентными*.

Формула $((\neg P_1(x_1)) \vee P_1(x_1))$ есть пример общезначимой формулы. Формула $((\neg P_1(x_1)) \vee P_1(f_1(x_1)))$ выполнима, но не общезначима.

Далее, как и в случае логики высказываний, договоримся опускать скобки, подразумевая, что $\neg$ — самая приоритетная операция, и в случае, когда порядок расстановки скобок не существенен (например, для ассоциативных операций). Кроме того, договоримся опускать скобки в формулах вида $(Q_1(Q_2A))$, где Q_1, Q_2 — любые кванторы. Например, вместо $(\forall x_1(\exists x_2(\forall x_3 P_1(x_1, x_2, x_3))))$ будем писать $\forall x_1 \exists x_2 \forall x_3 P_1(x_1, x_2, x_3)$.

2.2 Полнота исчисления предикатов

Для указания дедуктивной системы, перечисляющей все общезначимые формулы логики предикатов, воспользуемся имеющимися у нас схемами аксиом А1—А3, в которых теперь f, g, h — произвольные формулы логики предикатов, и добавим к ним следующие две схемы аксиом.

А4. Если f, g — формулы логики предикатов и x — предметная переменная, не являющаяся свободной переменной формулы f, то формула $(\forall x(f \to g)) \to (f \to (\forall x g))$ есть аксиома.

A5. Если f есть формула логики предикатов, x — свободная предметная переменная формулы f и t — терм, свободный для переменной x в f, то формула $(\forall x f) \to S_t^x f$ есть аксиома.

Правилами вывода в этой дедуктивной системе являются уже известное нам правило *modus ponens*, а также новое правило (известное как *правило обобщения*), позволяющее из формулы f выводить формулу $(\forall x f)$. Как и в случае логики высказываний, ограничиваемся только логическими связками $\neg$ и $\to$, причем из кванторов используем только квантор общности. Легко видеть, что квантор существования может быть выражен через квантор общности и отрицание: формулы $\exists x A(x)$ и $\neg(\forall x(\neg A(x)))$ эквивалентны.

Так как все аксиомы указанной дедуктивной системы общезначимы, а правила вывода сохраняют общезначимость, то все выводимые в ней формулы общезначимы. Обратно, имеет место следующее утверждение.

Теорема 2.1 (Гёделя [11] о полноте исчисления предикатов). *Каждая общезначимая формула логики предикатов является выводимой в приведенной выше дедуктивной системе*.

Таким образом, в исчислении предикатов, так же как в исчислении высказываний, существует алгоритмическая процедура, перечисляющая все общезначимые формулы (например, основанная на описанной выше дедуктивной системе для логики предикатов) и позволяющая для любой общезначимой формулы за конечное число шагов установить ее общезначимость. С практической точки зрения, однако, данная процедура является чрезмерно трудоемкой, поэтому развитие работ по автоматическому доказательству теорем в логике предикатов было связано с поиском более приемлемых в «прикладном» отношении ее модификаций.

С другой стороны, в отличие от логики высказываний, для логики предикатов справедливо следующее утверждение.

Теорема 2.2 (Чёрча [10]). *Для логики предикатов не существует алгоритма, распознающего общезначимые формулы.*

Доказательство теоремы Гёделя о полноте мы здесь не приводим, так как оно близко по своей технике к приводимому ниже доказательству полноты процедуры автоматического доказательства теорем, использующей правило резолюции. По завершении последнего мы вернемся к теореме Гёделя и дадим краткий набросок схемы ее доказательства.

2.3. Доказательство общезначимости с помощью правила резолюции

Мы опишем здесь одну из процедур, позволяющую для любой общезначимой формулы за конечное число шагов установить ее обще-

значимость. Эта процедура связана с использованием при поиске доказательства так называемого *правила резолюции* — аналога уже известного нам правила для логики высказываний.

Согласно данной процедуре, доказательство общезначимости замкнутой формулы F складывается из пяти этапов. Мы будем демонстрировать эти этапы, разбирая следующий пример.

Пример 2.2

Установить общезначимость формулы F, задаваемой выражением $F = (A_1 \,\&\, A_2 \,\&\, A_3) \to A_4$, где $A_1 = \forall x \exists y (P(x, y) \vee Q(x, y))$, $A_2 = \forall x \forall y (P(x, y) \to \to R(x, f(x, y)))$, $A_3 = \forall x \forall y (Q(x, y) \to R(x, f(x, y)))$, $A_4 = \forall x \exists y R(x, y)$.

Этап 1. Рассматривается формула $F_0 = \neg F$ и далее предпринимаются действия, направленные на установление невыполнимости формулы F_0 (т. е. формула F доказывается «от противного»).

Для примера 2.2 имеем: $F_0 = \neg F = \neg((A_1 \,\&\, A_2 \,\&\, A_3) \to A_4)$.

Этап 2. На этом этапе мы устраняем логические связки вида $\sim$, $\to$ и все кванторы перемещаются к «началу» формулы.

А именно, к формуле F_0 применяется цепочка эквивалентных (в смысле определенной выше эквивалентности формул) преобразований:

а) используются эквивалентные замены

$$(f \sim g) = (f \,\&\, g \vee (\neg f) \,\&\, (\neg g)),\ (f \to g) = (\neg f \vee g),$$

позволяющие устранить логические связки вида $\sim$, $\to$.

б) используются замены вида $(A * QxB) = Qy(A * S_y^x B)$, где $*$ — связка $\vee$ либо $\&$; Q — квантор $\forall$ либо $\exists$; A и B — формулы; y — предметная переменная, не входящая ни в A, ни в B. Кроме того, используются замены $\neg \forall xA = \exists x(\neg A)$, $\neg \exists xA = \forall x(\neg A)$.

Замены, указанные в пункте б), применяются до тех пор, пока это возможно, они осуществляют перемещение кванторов к «началу» формулы, в результате чего F_0 преобразуется в формулу F_1 вида

$$Q_1 x_1 \ldots Q_n x_n A,$$

где A — бескванторная формула; $Q_1, \ldots, Q_n$ — символы кванторов. Про F_1 говорят, что она находится в *предваренной нормальной форме*.

Для примера 2.2 имеем:

$$F_1 = \neg(\neg(A_1 \,\&\, A_2 \,\&\, A_3) \vee A_4) = \neg A_4 \,\&\, A_1 \,\&\, A_2 \,\&\, A_3 =$$

$$= \exists x_4 \forall y_4 \forall x_1 \exists y_1 \forall x_2 \forall y_2 \forall x_3 \forall y_3 A'_4 \,\&\, A'_1 \,\&\, A'_2 \,\&\, A'_3,$$

где $A'_4 = \neg R(x_4, y_4)$; $A'_1 = P(x_1, y_1) \vee Q(x_1, y_1)$; $A'_2 = \neg P(x_2, y_2) \vee R(x_2, f(x_2, y_2))$; $A'_3 = \neg Q(x_3, y_3) \vee R(x_3, f(x_3, y_3))$.

Этап 3. На этом этапе мы избавляемся от кванторов существования.

Лемма 2.2. *Пусть* $f = \forall x_1 \ldots \forall x_k \exists x_{k+1} g$ *— формула логики предикатов, где* $k \geq 0$; g *—* формула *в предваренной нормальной форме*; $f' = \forall x_1 \ldots \forall x_k g'$, *где* $g' = S^{x_{k+1}}_{\varphi(x_1,\ldots,x_k)} g$ *— результат замены всех свободных вхождений переменной* x_{k+1} *в* g *на* $\varphi(x_1, \ldots, x_k)$, φ *— функциональный символ арности* k, *не встречающийся в* g (*если* $k = 0$, *то* φ *— предметная константа*). *Тогда* f *выполнима тогда и только тогда, когда* f' *выполнима.*

Доказательство. Предположим, что формула f выполнима, пусть I — интерпретация, в которой ее значение есть И. Рассмотрим предикат $I(g) = h(x_1, \ldots, x_{k+1})$. Так как $I(\forall x_1 \ldots \forall x_k \exists x_{k+1} g) =$ И, то для любых элементов $a_1, \ldots, a_k$ области M интерпретации I существует элемент a_{k+1} этой области, такой, что $h(a_1, \ldots, a_{k+1}) =$ И.

Определим на M функцию $p(x_1, \ldots, x_k)$, полагая в указанной ситуации $p(a_1, \ldots, a_k) = a_{k+1}$. Тогда $h(x_1, \ldots, x_k, p(x_1, \ldots, x_k)) \equiv$ И на M.

Пусть I' — интерпретация, отличающаяся от I только тем, что в ней символу φ сопоставляется функция $p(x_1, \ldots, x_k)$. Используя лемму 2.1 об интерпретациях, получим, что $I'(g')$ есть результат подстановки в $I'(g) \equiv I(g) = h(x_1, \ldots, x_{k+1})$ функции $I'(\varphi(x_1, \ldots, x_k)) = p(x_1, \ldots, x_k)$ вместо переменной x_{k+1}, т. е. $I'(g') \equiv$ И на M. Таким образом, $I'(f') =$ И, и формула f' выполнима.

Обратно, если f' выполнима, то рассматриваем интерпретацию I, в которой $I(f') =$ И, т. е. $I(g') \equiv$ И на M, снова используя лемму об интерпретациях, находим, что $I(g')$ получено подстановкой в функцию $I(g) = h(x_1, \ldots, x_k, x_{k+1})$ функции $I(\varphi(x_1, \ldots, x_k)) = p(x_1, \ldots, x_k)$ вместо переменной x_{k+1}, т. е. для любых $a_1, \ldots, a_k$ из M выполнено $h(a_1, \ldots, a_k, p(a_1, \ldots, a_k)) =$ И, и $I(f) =$ И.

Лемма доказана. □

Описанное в лемме 2.2 преобразование $f = f'$ позволяет последовательно устранять кванторы $\exists$ из кванторной приставки; применяя его необходимое число раз к формуле F_1, получим в результате формулу F_2 вида $\forall x_1 \ldots \forall x_n A$, где A — бескванторная формула. Говорят, что F_2 находится в *сколемовской нормальной форме*. Заметим, что общезначимость исходной формулы F эквивалентна невыполнимости формулы F_2.

Для примера 2.2 имеем

$$F_2 = \forall y_4 \forall x_1 \forall x_2 \forall y_2 \forall x_3 \forall y_3 A''_4 \,\&\, A''_1 \,\&\, A''_2 \,\&\, A''_3,$$

где $A''_4 = \neg R(a, y_4)$; $A''_1 = P(x_1, g(y_4, x_1)) \vee Q(x_1, g(y_4, x_1))$; $A''_2 = A'_2$; $A''_3 = A'_3$.

Этап 4. На этом этапе мы приводим формулу к конъюнктивной нормальной форме.

Атомы и их отрицания называются *литерами*, дизъюнкции литер — *дизъюнктами*, логическая константа Л по определению также считается дизъюнктом. Конъюнкция различных дизъюнктов называется *конъюнктивной нормальной формой*.

На предыдущем этапе мы получили сколемовскую нормальную форму $F_2 = \forall x_1 \ldots \forall x_n A$, где A — бесквантороная формула. Теперь отбрасываем все кванторы и, используя уже описанную для логики высказываний процедуру, определяемую тождествами (1.2) — (1.6), преобразуем формулу A к конъюнктивной нормальной форме и получим формулу A' вида $\&_{i=1}^{m} D_i$, где D_i — дизъюнкты.

Для примера 2.2 имеем $A' = D_1 \,\&\, D_2 \,\&\, D_3 \,\&\, D_4$, где $D_1 = P(x_1, g(y_4, x_1)) \vee \vee Q(x_1, g(y_4, x_1))$; $D_2 = \neg P(x_2, y_2) \vee R(x_2, f(x_2, y_2))$; $D_3 = \neg Q(x_3, y_3) \vee R(x_3, f(x_3, y_3)))$; $D_4 = \neg R(a, y_4)$.

Этап 5. Как и в случае логики высказываний, для установления невыполнимости формулы A' мы введем вспомогательную дедуктивную систему, аксиомами которой будут являться определенные выше дизъюнкты $D_1, \ldots, D_m$, и докажем, что выводимость в этой системе константы Л эквивалента невыполнимости A'.

Для формулировки правил вывода данной дедуктивной системы нам понадобится понятие унифицирующей подстановки.

Подстановка — это множество пар $\sigma = \{x_1 = t_1, \ldots, x_n = t_n\}$, где n — натуральное число; $x_1, \ldots, x_n$ — предметные переменные; $t_1, \ldots, t_n$ — термы, в которых не встречаются переменные $x_1, \ldots, x_n$.

Если f — бесквантороная формула или терм, то $\sigma(f) = S^{x_1,\ldots,x_n}_{t_1,\ldots,t_n} f$ — подстановка термов $t_1, \ldots, t_n$ вместо переменных $x_1, \ldots, x_n$ в формулу f. Тем самым подстановки мы будем рассматривать как операторы на множестве бесквантороных формул и термов.

Пусть $f_1, g_1, \ldots, f_k, g_k$ — бесквантороные формулы либо термы, тогда подстановка σ называется *унифицирующей подстановкой* для множества пар $\{(f_1, g_1), \ldots, (f_k, g_k)\}$, если $\sigma(f_1) = \sigma(g_1), \ldots, \sigma(f_k) = \sigma(g_k)$.

Опишем алгоритм построения унифицирующей подстановки для множества пар $\{(f_1, g_1), \ldots, (f_k, g_k)\}$.

Выпишем систему уравнений $f_1 = g_1, \ldots, f_k = g_k$.

Для описания в явном виде всей совокупности ее решений используем следующие эквивалентные преобразования данной системы:

а) если при некотором i слова f_i, g_i имеют, соответственно вид $\varphi(t_1, \ldots, t_p)$ и $\psi(t'_1, \ldots, t'_q)$ (φ, ψ — предикатные символы, функциональные символы либо логические связки), то при $\varphi \neq \psi$ делаем вывод о несовместности системы и невозможности построения унифи-

цирующей подстановки, а при $\varphi = \psi$ (следовательно, $p = q$) заменяем уравнение $f_i = g_i$ на группу уравнений $t_1 = t'_1, \ldots, t_p = t'_p$;

б) если f_i совпадает с g_i, то уравнение $f_i = g_i$ удаляется;

в) если при некотором i одно из слов f_i, g_i есть переменная x, а другое — терм t, $x \neq t$, то в случае, когда t содержит x, делаем вывод о несовместности системы и невозможности построения унифицирующей подстановки. В противном случае заменяем уравнение $f_i = g_i$ на уравнение $x = t$. Далее подставляем t вместо x во все оставшиеся уравнения системы. Данное преобразование выполняется только при условии, что существует отличное от $x = t$ уравнение, содержащее x.

Уравнения вида $x = t$, где x — переменная, а t — терм, не содержащий x, будем называть *подстановочными*.

Преобразование в) уменьшает число переменных в уравнениях, не являющихся подстановочными, и поэтому оно применимо лишь конечное число раз.

Преобразование а), в случае применения его к уравнениям максимально возможной длины, уменьшает число уравнений данной либо большей длины, и, следовательно, его возможно применять лишь конечное число раз по завершении преобразований в).

Таким образом, процесс применения к системе уравнений преобразований а), б), в) обрывается за конечное число шагов.

Результирующая система, к которой эти преобразования неприменимы, может иметь лишь вид $x_{i_1} = \theta_1, \ldots, x_{i_p} = \theta_p$, где термы $\theta_1, \ldots, \theta_p$ не содержат переменных $x_{i_1}, \ldots, x_{i_p}$. Полученная подстановка $\sigma = \{x_{i_1} = \theta_1, \ldots, x_{i_p} = \theta_p\}$ и будет унифицирующей подстановкой для множества пар $\{(f_1, g_1), \ldots, (f_k, g_k)\}$.

Пример 2.3

Построить унифицирующую подстановку для одной пары формул $(\neg P(a, y), \neg P(x, g(x)))$.

Решение. Изначально имеем уравнение $\neg P(a, y) = \neg P(x, g(x))$.

Применяя эквивалентное преобразование а), получим $P(a, y) = P(x, g(x))$.

Еще раз применяя эквивалентное преобразование а), получим систему уравнений $a = x, y = g(x)$.

Применяя эквивалентное преобразование в), получим унифицирующую подстановку $x = a, y = g(a)$.

Замыканием бесквантворной формулы A (обозначаемым $[A]$) будем называть формулу $\forall x_1 \ldots \forall x_k A$, где $x_1, \ldots, x_k$ — все входящие в A предметные переменные.

Если определенное на множестве дизъюнктов «правило вывода» позволяет выводить из дизъюнктов $D_1, \ldots, D_r$ лишь такие следствия D_0, что формула $[D_0]$ есть логическое следствие формул $[D_1], \ldots, [D_r]$, то такое правило вывода назовем *корректным*.

Как легко видеть, если при использовании некоторого набора корректных правил вывода из дизъюнктов $D_1, \ldots, D_m$, введенных на этапе 4 , удается за конечное число шагов извлечь логическую константу Л, то формула F_2 невыполнима, а F — общезначима.

Предлагаемая здесь процедура автоматического доказательства теорем использует при поиске такого вывода константы Л следующие (как легко видеть, корректные) правила вывода.

R0 (переобозначения переменных). Из произвольного дизъюнкта D выводится дизъюнкт D', полученный из D переобозначением без отождествлений переменных, входящих в D.

R1 (правило резолюции). Если $f_1 \vee \ldots \vee f_s$ и $g_1 \vee \ldots \vee g_r$ — дизъюнкты, множества переменных которых не пересекаются, причем f_i имеет вид $P(t_1, \ldots, t_k)$, а g_j — вид $\neg P(t'_1, \ldots, t'_k)$, σ — унифицирующая подстановка для пары $(P(t_1, \ldots, t_k), P(t'_1, \ldots, t'_k))$, $i \in \{1, \ldots, s\}$, $j \in \{1, \ldots, r\}$, то выводится дизъюнкт

$$\sigma(f_1) \vee \ldots \vee \sigma(f_{i-1}) \vee \sigma(f_{i+1}) \vee \ldots \vee \sigma(f_s) \vee \sigma(g_1) \vee$$

$$\vee \ldots \vee \sigma(g_{j-1}) \vee \sigma(g_{j+1}) \vee \ldots \vee \sigma(g_r)$$

(если $s = r = 1$, то выводится константа Л).

Дизъюнкт, получаемый согласно правилу R1, называется *резольвентой* исходных дизъюнктов.

R2 (правило склеивания). Если $f_1 \vee \ldots \vee f_s$ — дизъюнкт и σ — унифицирующая подстановка для (f_i, f_j), $i \neq j$, $i, j \in \{1, \ldots, s\}$, то выводится дизъюнкт

$$\sigma(f_1) \vee \ldots \vee \sigma(f_{i-1}) \vee \sigma(f_{i+1}) \vee \ldots \vee \sigma(f_s).$$

Теорема 2.3. *Замыкание бесквантворной формулы* $\&_{i=1}^{m} D_i$ *невыполнимо тогда и только тогда, когда за конечное число шагов из дизъюнктов* $D_1, \ldots, D_m$ *при помощи правил вывода* R0, R1, R2 *может быть выведена логическая константа* Л.

Идея доказательства данной теоремы перекликается с идеей доказательства теоремы 1.3.

Вернемся к рассмотрению примера 2.2.

После этапа 4 мы получили следующую систему дизъюнктов:

$$D_1 = P(x_1, g(y_4, x_1)) \vee Q(x_1, g(y_4, x_1)), D_2 = \neg P(x_2, y_2) \vee R(x_2, f(x_2, y_2)),$$

$$D_3 = \neg Q(x_3, y_3) \vee R(x_3, f(x_3, y_3))), D_4 = \neg R(a, y_4).$$

1. Для пары $(R(x_2, f(x_2, y_2)), R(a, y_4))$ найдем унифицирующую подстановку:

$$R(x_2, f(x_2, y_2)) = R(a, y_4); x_2 = a, f(x_2, y_2) = y_4.$$

Тем самым унифицирующая подстановка имеет вид $\{x_2 = a, y_4 = f(a, y_2)\}$.

Используя эту унифицирующую подстановку, применим правило резолюции R1 к дизъюнктам D_2, D_4 и получим дизъюнкт

$$D_5 = \neg P(a, y_2).$$

Аналогично, применяя правило резолюции R1 к дизъюнктам D_3, D_4, получим дизъюнкт

$$D_6 = \neg Q(a, y_3)$$

2. Для пары $(P(x_1, g(y_4, x_1)), P(a, y_2))$ найдем унифицирующую подстановку:

$$P(x_1, g(y_4, x_1) = P(a, y_2); x_1 = a, g(y_4, x_1) = y_2.$$

Тем самым унифицирующая подстановка имеет вид $\{x_1 = a, y_2 = g(y_4, a)\}$.

Используя эту унифицирующую подстановку, применим правило резолюции R1 к дизъюнктам D_1, D_5 и получим дизъюнкт

$$D_7 = Q(a, g(y_4, a)).$$

3. Для пары $(Q(a, y_3)), Q(a, g(y_4, a))$ найдем унифицирующую подстановку: $y_3 = g(y_4, a)$

Используя эту унифицирующую подстановку, применим правило резолюции R1 к дизъюнктам D_6, D_7 и получим дизъюнкт Л.

Отсюда, согласно теореме 2.3, формула A' невыполнима, а значит, невыполнима формула F_0, а F — общезначима.

2.4 Связь с теоремой Гёделя о полноте

В описанной выше процедуре автоматического доказательства теорем была использована некоторая вспомогательная дедуктивная система, использующая три правила вывода: переобозначение переменных, правило резолюции и правило склеивания. Роль аксиом в ней играли дизъюнкты, полученные в процессе преобразований отрицания исходной теоремы. «Полнота» этой дедуктивной системы заключалась в возможности получения за конечное число шагов константы «ложь» для любого невыполнимого множества дизъюнктов. Оказывается, что использованная при установлении данной полноты техника весьма близка к технике, применяемой для доказательства теоремы Гёделя о полноте приведенной в начале раздела классической дедуктивной системы логики предикатов. Чтобы провести сопоставление, дадим здесь краткий набросок схемы доказательства теоремы Гёделя.

Прежде всего нам понадобится понятие *теории первого порядка*. Это произвольная дедуктивная система, полученная из дедуктивной системы логики предикатов добавлением некоторого (конечного

либо бесконечного) множества аксиом. Теорию первого порядка называют *противоречивой*, если в ней выводимы одновременно какая-либо формула и ее отрицание. Очевидно, в таком случае в ней выводима вообще любая формула. Одна теория первого порядка является *расширением* другой, если каждая формула, выводимая во второй теории, выводима также в первой. Наконец, введем понятие *полной теории*, т. е. такой, в которой для любой замкнутой формулы выводима либо она, либо ее отрицание.

Первым шагом доказательства теоремы Гёделя является установление следующей леммы .

Лемма 2.3 (Линденбаума). *Если теория первого порядка непротиворечива, то существует непротиворечивое полное ее расширение.*

Доказательство леммы основано на рассмотрении последовательности F_0, F_1, …, перечисляющей все замкнутые формулы логики предикатов. Начиная с исходной непротиворечивой теории T по этой последовательности строится последовательность теорий $T = T_0, T_1, \ldots$. Каждый раз, как оказывается, что формула $\neg F_i$ не выводима в теории T_i, теория T_{i+1} получается из T_i добавлением F_i в качестве аксиомы. В противном случае полагается $T_{i+1} = T_i$. Затем рассматривается теория T', множество аксиом которой является объединением множеств аксиом всех теорий $T_0, T_1, \ldots$. Как легко видеть, T' представляет собой непротиворечивое полное расширение теории T. Данная конструкция вполне аналогична конструкции, применявшейся при доказательстве теоремы Эрбрана, когда определялась бесконечная последовательность истинностных значений термов эрбрановского универсума.

Если в некоторой интерпретации логики предикатов истинны все аксиомы теории первого порядка T, то называем ее *моделью* теории T. Далее доказывается следующая лемма.

Лемма 2.4. *Всякая непротиворечивая теория первого порядка имеет модель со счетной областью.*

Пусть T — непротиворечивая теория первого порядка. Рассмотрим последовательность $F_1(x_1), F_2(x_2), \ldots$, в которой перечисляются все формулы логики предикатов, содержащие не более одной свободной переменной. Пусть также $a_1, a_2, \ldots$ — последовательность различных предметных констант, таких, что a_i не содержится в формулах $F_1(x_1), \ldots, F_i(x_i)$. Введем в рассмотрение формулу S_n вида

$$\neg\forall_{x_i} F_i(x_i) \to \neg F_i(a_i).$$

Определяем теорию T_n как результат присоединения к T аксиом $S_1, S_2, \ldots, S_n$, а теорию T' — как результат присоединения к T аксиом $S_1, S_2, \ldots$. Очевидно, что для установления непротиворечивости T' достаточно установить непротиворечивость каждой теории T_n,

$n = 1, 2, \ldots$. Последнее нетрудно установить индукцией по n. Заметим, что формула S_n эквивалентна формуле $\exists_{x_i} G_i(x_i) \to G_i(a_i)$, где $G_i(x_i)$ есть $\neg F_i(x_i)$. Такое представление объясняет смысл добавления аксиом S_n: они позволяют перейти от формулы с внешним квантором существования к формуле, полученной из нее отбрасыванием квантора и подстановкой «новой» константы вместо подкванторной переменной. Этот переход аналогичен переходу, выполнявшемуся выше при получении сколемовской нормальной формы.

После установления непротиворечивости теории T' применяем лемму Линденбаума, согласно которой она имеет непротиворечивое полное расширение T''. Используя теорию T'', непосредственно указываем интерпретацию для теории T, в которой истинны все ее аксиомы. Эта интерпретация строится аналогично тому, как строилась интерпретация в доказательстве теоремы Эрбрана. Областью интерпретации здесь служит множество M всех термов, не имеющих переменных. Очевидно, оно счетно. Интерпретацией константы служит она сама; функциональный символ f арности n интерпретируется операцией, переводящей термы $t_1, \ldots, t_n$ в терм $f(t_1, \ldots, t_n)$.

Отношение, являющееся интерпретацией предикатного символа P арности n, истинно на наборе термов $t_1, \ldots, t_n$ тогда и только тогда, когда формула $P(t_1, \ldots, t_n)$ выводима в теории T''. Доказательство того, что в указанной интерпретации будет истинной любая формула F, выводимая в теории T'', осуществляется сравнительно несложной индукцией по построению формулы F. Так как любая формула, выводимая в T, выводима также в теории T'', получаем отсюда, что построенная интерпретация является моделью для T.

Чтобы доказать теперь теорему Гёделя о полноте, рассмотрим произвольную замкнутую общезначимую формулу F логики предикатов. Если эта формула не выводима в логике предикатов, то можно присоединить ее отрицание к аксиомам и получить непротиворечивую теорию первого порядка. По лемме 2.4 она имеет модель. В этой модели одновременно должны быть истинны и общезначимая формула F, и ее отрицание. Полученное противоречие и устанавливает выводимость формулы F.

2.5. Неполнота формальной арифметики

Хотя для любой непротиворечивой теории первого порядка T существует непротиворечивое полное ее расширение T', это еще не означает существования алгоритма, распознающего по формуле, является ли она аксиомой теории T', или хотя бы перечисляющего все аксиомы этой теории. Гёдель установил, что любая непротиворечи-

вая теория первого порядка, для которой имеется алгоритм, распознающий ее аксиомы, и притом включающая в себя формализацию простейших свойств целых неотрицательных чисел, обязательно является неполной. Это утверждение, известное как теорема Гёделя о неполноте формальной арифметики, часто упоминается в связи философскими аспектами искусственного интеллекта, и мы приведем здесь краткую схему рассуждений, лежащих в основе его доказательства.

Прежде всего приведем список аксиом, присоединение которых к общим аксиомам логики предикатов дает теорию первого порядка, называемую формальной арифметикой. В этих аксиомах выделены следующие специальные символы: символ предметной константы, обозначаемый далее 0; функциональный символ арности 1, обозначаемый штрихом; функциональные символы арности 2, обозначаемые знаками сложения и умножения; предикатный символ арности 2, обозначаемый символом равенства. Напомним, что язык логики предикатов имеет счетное множество символов предметных констант и счетное множество функциональных либо предикатных символов произвольной арности. Аксиомами служат следующие формулы:

1) $x_1 = x_2 \to (x_1 = x_3 \to x_2 = x_3)$;
2) $x_1 = x_2 \to$ <<Eθν060.ωμφ>>;
3) $\neg(0 = x_1')$;
4) $x_1' = x_2' \to x_1 = x_2$;
5) $x_1 + 0 = x_1$;
6) $x_1 + x_2' = (x_1 + x_2)'$;
7) $x_1 \cdot 0 = 0$;
8) $x_1 \cdot x_2' = x_1 \cdot x_2 + x_1$;
9) $A(0) \to (\forall_x(A(x) \to A(x')) \to \forall_x A(x))$.

Здесь x, x_1, x_2, x_3 — произвольные предметные переменные; $A(x)$ — произвольная формула логики предикатов.

Областью стандартной интерпретации для формальной арифметики служит множество целых неотрицательных чисел; предметной константе 0 соответствует число 0; функциональному символу «штрих» соответствует операция прибавления единицы; сложение, умножение и равенство понимаются обычным образом. Последняя из приведенных выше схем аксиом представляет собой формализацию принципа математической индукции. Для любого целого неотрицательного числа k будем обозначать через $T(k)$ терм, полученный последовательным k-кратным применением к 0 операции $'$. В стандартной интерпретации значением такого терма будет служить число k.

Отношение $R(x_1, \ldots, x_n)$, определенное на множестве наборов целых неотрицательных чисел, называется *выразимым в формальной*

арифметике, если существует такая формула $A(x_1, \ldots, x_n)$ логики предикатов, что для любых целых неотрицательных чисел $k_1, \ldots, k_n$:

а) если $R(k_1, \ldots, k_n)$ истинно, то выводима формула $A(T(k_1), \ldots, T(k_n))$;

б) если $R(k_1, \ldots, k_n)$ ложно, то выводима формула $\neg A(T(k_1), \ldots, T(k_n))$.

Выводимость здесь понимается относительно аксиом формальной арифметики.

Функция $f(x_1, \ldots, x_n)$, определенная на множестве наборов целых неотрицательных чисел и принимающая целые неотрицательные значения, называется *представимой в формальной арифметике*, если существует такая формула $A(x_1, \ldots, x_n, x_{n+1})$, что для любых целых неотрицательных чисел $k_1, \ldots, k_n, k_{n+1}$:

а) если $f(k_1, \ldots, k_n) = k_{n+1}$, то выводима формула $A(T(k_1), \ldots, T(k_n), T(k_{n+1}))$;

б) выводима формула $\exists! x_{n+1} A(T(k_1), \ldots, T(k_n), x_{n+1})$, где знак «!» после квантора существования — сокращенное обозначение для формулы, выражающей существование и единственность.

Выделим специальный класс арифметических (т. е. определенных на множестве целых неотрицательных чисел и принимающих целые неотрицательные значения) функций, определяемый индуктивным образом. Этот класс дает формализацию понятия эффективно вычислимой арифметической функции, т. е. такой функции, для вычисления значений которой существует алгоритм. Базис индукции составляют следующие функции:

1) нуль-функция $Z(x)$, тождественно равная нулю;

2) функция прибавления единицы: $N(x) = x + 1$;

3) проецирующие функции: $U_{i,n}(x_1, \ldots, x_n) = x_i$.

Операции, порождающие новые функции из ранее построенных, таковы:

1) подстановка — получение функции

$$f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n))$$

из функций

$$g(y_1, \ldots, y_m), h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n);$$

2) примитивная рекурсия — получение функции $f(x_1, \ldots, x_n, x_{n+1})$ из функций $g(x_1, \ldots, x_n)$, $h(x_1, \ldots, x_{n+2})$ с помощью соотношений:

$$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n);$$

$$f(x_1, \ldots, x_n, x_{n+1} + 1) = h(x_1, \ldots, x_n, x_{n+1}, f(x_1, \ldots, x_{n+1}));$$

3) минимизация — получение функции $f(x_1, \ldots, x_n)$ из функции $g(x_1, \ldots, x_{n+1})$, для которой уравнение $g(x_1, \ldots, x_n, y) = 0$ имеет хотя бы один корень y при любом наборе целых неотрицательных $x_1, \ldots,$

x_n. Значением функции $f(x_1, \ldots, x_n)$ является наименьший из указанных корней y.

Функции, принадлежащие заданному таким индуктивным определением классу, называются *рекурсивными*. Если ограничиться лишь операциями подстановки и примитивной рекурсии, то возникает собственный подкласс класса рекурсивных функций, называемый классом *примитивно-рекурсивных* функций.

Оказывается, что каждая рекурсивная функция представима в формальной арифметике, а каждое рекурсивное арифметическое отношение (т. е. отношение на множестве наборов целых неотрицательных чисел, имеющее рекурсивную характеристическую функцию) выразимо в ней. Доказываются эти утверждения индукцией по определению класса рекурсивных функций. Чтобы с помощью арифметических отношений и функций можно было определять различные свойства конечных последовательностей символов в алфавите языка логики предикатов, вводится специальная нумерация этих последовательностей. Сначала символам s алфавита сопоставляются взаимно однозначно их номера $N(s)$, после чего последовательности $s_1, s_2, \ldots, s_k$ сопоставляется номер $2^{N(s_1)} \cdot 3^{N(s_2)} \cdot \ldots \cdot p_k^{N(s_k)}$. Здесь p_i — простое число с номером i, $i = 1, \ldots, k$. Легко видеть, что по своему номеру последовательность восстанавливается однозначно.

Теперь для таких свойств слов в алфавите языка логики предикатов, как, например, свойство быть термом, формулой, аксиомой, последовательностью формул, представляющей собой вывод, и т. п., можно рассматривать соответствующие арифметические отношения для номеров слов. Этот прием перевода свойств конечных последовательностей символов на арифметический язык был предложен Гёделем и называется *гёделевой нумерацией*. Номер, сопоставляемый слову указанным выше образом, называем *гёделевым номером* этого слова.

Используя утверждение о выразимости в формальной арифметике любых рекурсивных отношений, далее можно доказать выразимость в ней следующих двух отношений $W_1(u, v)$ и $W_2(u, v)$:

1) $W_1(u, v)$ истинно тогда и только тогда, когда u — гёделев номер формулы $A(x_1)$, имеющей свободную переменную x_1 (первую переменную списка предметных переменных алфавита языка логики предикатов), и v есть гёделев номер некоторого вывода в формальной арифметике формулы $A(T(u))$;

2) $W_2(u, v)$ истинно тогда и только тогда, когда u есть гёделев номер формулы $A(x_1)$, имеющей свободную переменную x_1, и v есть гёделев номер вывода в формальной арифметике формулы $\neg A(T(u))$.

Оба эти свойства очевидным образом допускают алгоритмическую проверку, и доказательство рекурсивности отношений W_1, W_2 сводится, по существу, к обычному программированию.

Рассмотрим теперь формулы $U_1(x_1, x_2)$, $U_2(x_1, x_2)$, с помощью которых выразимы отношения W_1, W_2. Построим формулу U следующего вида:

$$\forall_{x_2}(U_1(x_1, x_2) \to \exists_{x_3}(x_3 \le x_2 \,\&\, U_2(x_1, x_3))).$$

Пусть n — гёделев номер этой формулы. Строим далее формулу V, получаемую подстановкой в U терма $T(n)$ вместо переменной x_1:

$$\forall_{x_2}(U_1(T(n), x_2) \to \exists_{x_3}(x_3 \le x_2 \,\&\, U_2(T(n), x_3))).$$

Теорема Гёделя о неполноте формальной арифметики (в так называемой форме Россера, несколько усиливающей первоначальное утверждение Гёделя) утверждает, что если формальная арифметика непротиворечива, то в ней не выводима ни формула V, ни ее отрицание, т. е. она неполна.

Смысл утверждения V достаточно прозрачен: оно говорит, что если x_2 есть гёделев номер вывода утверждения V, то существует вывод отрицания утверждения V, имеющий гёделев номер, не превосходящий x_2. Это удобная в техническом отношении версия утверждения, которое утверждает свою собственную ложность. Как известно, оба допущения — об истинности либо о ложности такого рода утверждений — сразу приводят к противоречию. На этой стандартной схеме и построено доказательство теоремы. Завершающая его часть (после установления рекурсивности W_1, W_2) несложна, и подробности мы опустим.

Заметим, что приведенные рассуждения по аналогии могут быть воспроизведены и для любого расширения формальной арифметики, у которого множество аксиом рекурсивно, т. е. все такие расширения неполны.

Конечно, философская интерпретация теоремы Гёделя о неполноте как доказательства «принципиальной ограниченности аксиоматического подхода» требует большой осторожности. Рассматриваемое в ее доказательстве утверждение не несет совершенно никакой содержательной информации, утверждая лишь свою собственную ложность. Оно может быть воспринято как пример утверждения совершенно бессмысленного и именно из-за своей бессмысленности оказывающегося вне рамок действия аксиоматизации. В то же время из математической практики известны конкретные примеры вполне содержательных утверждений, таких, как континуум-гипотеза, оказавшихся вне области действия общепринятой аксиоматизации и, таким образом, выявивших ее неполноту. Теорема Гёделя оставляет открытым вопрос о том, сколь значительной может оказаться неполнота аксиоматизаций, если ограничиваться рассмотрением в них лишь достаточно осмысленных и содержательных утверждений, подобных континуум-гипотезе.

С точки зрения искусственного интеллекта, который лишь моделирует логические процессы естественного интеллекта, теорема Гёделя вообще не накладывает никаких дополнительных ограничений на первый, которые отсутствовали бы для второго. Точно так же, как человек, выявив независимость какого-либо утверждения от имеющейся аксиоматики, начинает рассматривать две независимые ветви теории либо, опираясь на практику, выбирает одну из них, так и компьютерная модель, воспроизводящая шаг за шагом логику рассуждений человека, в принципе, могла бы выполнять аналогичные действия.

2.6. Эвристики в управлении выводом

Вообще говоря, процесс поиска вывода константы Л из исходных дизъюнктов $B_1, \ldots, B_m$ при помощи правил вывода R0, R1, R2 может оказаться неприемлемо трудоемким, даже в рассматриваемом выше простом примере можно было бы указать достаточное количество альтернативных способов применения правил вывода (например, получить резольвенты дизъюнктов 1 и 2 либо 1 и 3).

Трудоемкость поиска вывода для описанной процедуры растет экспоненциально с увеличением глубины вывода, и это заставляет предпринимать исследования, направленные на нахождение принципов управления выводом следствий. Мы приведем здесь краткий обзор такого рода принципов, которые были найдены при практическом использовании указанной процедуры, а также при теоретическом ее исследовании. Эти принципы можно разбить на две группы — принципы исключения из рассмотрения некоторых «избыточных» следствий, с доказательством сохранения полноты процедуры при их применении (они получили название «стратегии очищения»), и эвристические принципы упорядочения перебора резольвент («стратегии упорядочения»). Начнем с простейших примеров таких стратегий.

1. **Стратегия предпочтения одночленов.** Как нетрудно заметить, применение правила резолюции к дизъюнктам $A \vee K_1 \vee \ldots \vee K_{m-1}$ и $\neg A' \vee K'_1 \vee \ldots \vee K'_{n-1}$ длины m и n соответственно дает резольвенту $K''_1 \vee \ldots \vee K''_{m-1} \vee K'''_1 \vee \ldots \vee K'''_{n-1}$ длины $m + n - 2$. Длина эта лишь в том случае окажется меньшей длины одного из исходных дизъюнктов, когда другой дизъюнкт имел длину 1, т. е. являлся «одночленом». Исходя из такого, вообще говоря, эвристического соображения получения следствий, более простых, чем исходные формулы, стратегия предпочтения одночленов рекомендует первоочередное применение правила резолюции к одночленам.

2. **Исключение тавтологий и уникальных литер.** В процессе вывода следствий отбрасываются дизъюнкты вида ($A \vee \neg A \vee B_1 \vee$

$\vee \ldots \vee B_k$) («тавтологии»). Если предикатный символ P встречается во всех дизъюнктах только с отрицанием либо только без отрицания, то все содержащие P дизъюнкты отбрасываются.

Стратегия предпочтения одночленов дает пример стратегии упорядочения, принцип исключения тавтологий и уникальных литер — пример стратегии очищения.

3. Стратегия первоочередного применения правила склеивания. Правило R1 применяется лишь после того, как были получены все возможные в текущей ситуации следствия с применением правила R2.

4. Стратегия использования подслучаев. Дизъюнкт D называется *подслучаем* дизъюнкта C, если существует такая подстановка σ, что C имеет вид $\sigma(D) \vee C'$. Стратегия заключается в отбрасывании всех возникающих при выводе дизъюнктов C, для которых уже найден был ранее некоторый их подслучай (данная стратегия является стратегией очищения и гарантирует получение за конечное число шагов константы Л).

5. Стратегия применения несущего множества дизъюнктов. Эта стратегия связана с выделением в исходном множестве дизъюнктов M такого, возможно большего, подмножества M', про которое известно, что оно выполнимо (например M' может быть образовано всеми аксиомами, из которых требуется извлечь заданное следствие). При выводе следствий допускается лишь такое применение правила резолюции, когда хотя бы один из несущих дизъюнктов принадлежит *несущему множеству*. Последнее определяется следующим индуктивным образом:

а) каждый элемент из $M \setminus M'$ относится к несущему множеству;

б) если резольвента получена при участии хотя бы одного элемента несущего множества, то она входит в несущее множество;

в) результаты применения правила R0 либо R2 к элементу несущего множества дают элемент несущего множества.

6. Стратегия использования линейных выводов. Линейный вывод представляет собой последовательность применений правила резолюции, определяемую диаграммой, представленной на рис. 2.1.

Здесь каждое C_i — либо элемент исходного множества дизъюнктов M, либо некоторое D_j при $j \in \{0, 1, \ldots, i-1\}$. Имеет место утверждение (Андерсон — Бледсоу), согласно которому для получения константы Л достаточно использовать лишь линейные выводы (здесь и далее мы рассматриваем лишь встречающиеся в выводах правила R1, так как именно они ответственны за экспоненциально нарастающую с увеличением глубины вывода трудоемкость поиска доказательства, не упоминая о возможных применениях «одноместных» правил R0 и R2). Заметим, что само по себе ограничение линейности вывода не устраняет древовидной схемы поиска линейного вывода при переборе различных возможностей для $C_1, C_2, \ldots, C_n$.

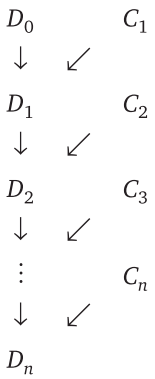

Рис. 2.1. **Линейный вывод**

7. Совместное применение стратегий использования подслучаев и слияния. Введем сначала вспомогательные понятия слияния и литеры слияния. Резольвента $\sigma(D_1') \vee \ldots \vee \sigma(D_n') \vee \sigma(D_1'') \vee \ldots \vee \sigma(D_m'')$ двух дизъюнктов $A \vee D_1' \vee \ldots \vee D_n'$ и $\neg A \vee D_1'' \vee \ldots \vee D_m''$ называется *слиянием* этих дизъюнктов, если существуют такие $i \in \{1, \ldots, n\}$ и $j \in \in \{1, \ldots, m\}$, что $\sigma(D_i') = \sigma(D_j'')$. Формула $\sigma(D_i')$ при этом называется *литерой слияния*. Андерсону и Бледсоу принадлежит утверждение о том, что при поиске вывода константы Л достаточно ограничится лишь такими линейными выводами, у которых каждое C_i (обозначения те же, что в пункте 6) либо принадлежит исходному множеству дизъюнктов M, либо является некоторым D_j, $j < i$, полученным при слиянии двух дизъюнктов, причем литерой (унифицируемой при применении правила резолюции формулой A либо $\neg A$) в C_i при получении D_i является некоторая литера слияния, а само D_i является подслучаем дизъюнкта D_{i-1}. Несмотря на существенное ограничение в использовании формул C_i, не принадлежащих M (по сравнению с пунктом 6), данное утверждение практически не устраняет эффект экспоненциального нарастания трудоемкости, так как достаточное количество альтернативных применений правил вывода на каждом шаге дает уже исходное множество M.

8. Стратегия упорядочения литер. Возможно применение при поиске вывода константы Л другого ограничения на вид линейного вывода. В этом случае разрешающей литерой в каждом D_i является первая литера, т. е. D_{i+1} возникает из $D_i = A \vee D_1' \vee \ldots \vee D_n'$ и $C_i = D_1'' \vee \ldots \vee D_{j-1}'' \vee \neg A \vee D_{j+1}'' \ldots \vee D_n''$ как

$$\sigma(D_1'') \vee \ldots \vee \sigma(D_{j-1}'') \vee \sigma(D_{j+1}'') \vee \ldots \vee \sigma(D_n'') \vee \sigma(D_1') \vee \ldots \vee \sigma(D_n'),$$

причем указанное здесь упорядочение литер в дизъюнкте D_{i+1} сохраняется. Как и в пункте 7, предполагается, что каждое C_i — либо из исходного множества дизъюнктов M, либо совпадает с некото-

рым D_j, $j < i$, являющимся дизъюнктом слияния. Полнота данной стратегии также была установлена в работах Андерсона и Бледсоу.

Иллюстрация стратегий. Для иллюстрации применения перечисленных выше стратегий рассмотрим сравнительно простой пример доказательства теорем из теории групп. Будет доказываться утверждение о том, что в ассоциативной системе, в которой уравнения вида $xa = b$ и $ay = b$ имеют левое и правое решения x и y, существует правый единичный элемент. При формулировке этого утверждения на языке логики предикатов используем трехместный предикатный символ $P(x, y, z)$, интерпретируемый как равенство $xy = z$. После преобразования к виду дизъюнктов аксиомы существования решений уравнений $xa = b$ и $ay = b$ принимают вид:

$D_1 = P(g(x, y), x, y)$ — существование решения для $xa = b$;

$D_2 = P(x, h(x, y), y)$ — существование решения для $ay = b$.

Ассоциативность выражается двумя дизъюнктами:

$$D_3 = \neg P(x, y, z) \vee \neg P(y, v, w) \vee \neg P(x, w, u) \vee P(z, v, u);$$

$$D_4 = \neg P(x, y, z) \vee \neg P(y, v, w) \vee \neg P(z, v, u) \vee P(z, w, u).$$

Наконец, отрицание утверждения о существовании правого единичного элемента приводится к виду

$$D_5 = \neg P(f(y), y, f(y)).$$

В исходной ситуации несущее множество состоит из единственного элемента D_5, при применении правил вывода будем строить лишь линейные выводы. Стратегия предпочтения одночленов дает лишь две возможности — применение правила R1 к D_5 и D_3 либо к D_5 и D_4. Выберем, например, первую из них, получим следующий дизъюнкт:

$$D_6 = \neg P(x, y, f(z)) \vee \neg P(y, z, v) \vee \neg P(x, v, f(z))$$

(по мере надобности в новых дизъюнктах проводим переобозначение переменных). Теперь несущее множество имеет уже два элемента — D_5 и D_6, и стратегия предпочтения одночленов вовлекает в рассмотрение также дизъюнкты D_1 и D_2. Применяя правило R1 к D_1 и D_6, получаем следующий дизъюнкт:

$$D_7 = \neg P(x, y, z) \vee \neg P(g(x, f(y)), z, f(y)).$$

Далее из D_7 и D_1 выводим:

$$D_8 = \neg P(x, y, z),$$

и, наконец, унифицируя D_8 с отрицанием D_2, выводим константу Л.

Разбирая пример, мы привели лишь ведущую к цели цепочку вывода. Вместе с тем здесь имелось уже достаточно большое множество альтернативных выводов, не отсекаемых перечисленными выше стратегиями: применение правила резолюции к парам (D_5, D_4), (D_2, D_6), (D_2, D_7), а также альтернативные применения этого правила к (D_1, D_6) и (D_1, D_7) дают новые дизъюнкты, которые в свою очередь приводят к новым следствиям. Таким образом, в целом сохраняется ветвящийся характер рассматриваемого в описанной процедуре логического вывода, и это ограничивает область эффективной ее применимости классом сравнительно простых задач, где проведение полного перебора является все еще реализуемым.

Попытки применения изложенной выше процедуры автоматического доказательства теорем в различных прикладных задачах, связанных с логическим выводом, привели к созданию языка ПРОЛОГ.

Не останавливаясь на технических подробностях, изложенных в руководствах по программированию на этом языке, мы приведем здесь лишь «общелогическую» схему организации и функционирования программ на языке ПРОЛОГ. Каждая такая программа представляет собой упорядоченный набор дизъюнктов (D_1, D_2, …, D_n), причем входной информацией для программ также служит некоторый дизъюнкт D_0. Программа пытается найти такую подстановку σ, для которой совокупность дизъюнктов $\{\sigma(D_0), D_1, D_2, \ldots, D_n\}$ становится невыполнимой, причем она не останавливается, найдя первую такую σ, а продолжает поиск и перечисляет все найденные σ вплоть до исчерпания всех возможностей, заложенных в ее схеме перечисления.

Сама эта схема исключительно проста — по существу, это полный перебор всех возможных линейных выводов, определяемых дизъюнктами $D_1, \ldots, D_n$, реализуемый по принципу «сначала вглубь». Более подробно, функционирование программы происходит следующим образом. Текущая ситуация описывается при помощи набора троек $S = (S_1, \ldots, S_m)$, где каждое S_i, $i = 1, \ldots, m$, имеет вид (σ_i, D_i^*, k_i); σ_i — подстановка; D_i^* — дизъюнкт; $k_i \in \{1, 2, \ldots, n + 1\}$. В исходной ситуации $m = 1$, $\sigma_1 = e$ — тождественная подстановка, $D_1^* = D_0$, $k_1 = 1$. На очередном шаге преобразований происходят рассмотрение текущей тройки $S_m = (\sigma_m, D_m^*, k_m)$, $D_m^* = K_1 \vee \ldots \vee K_s$, и перебор всех дизъюнктов, номер которых не меньше k_m. Для каждого D_j, $D_j = Q_1 \vee \ldots \vee Q_p$, предпринимается попытка унификации литер K_1 и $\neg Q_1$. Как только это удалось сделать, определяется резольвента D' дизъюнктов D_m^* и D_j, и к концу набора S присоединяется новая тройка $(\sigma_m\sigma', D', 1)$, где σ' — унифицирующая подстановка. Одновременно индекс k_m в тройке S_m заменяется на $j + 1$, и процесс возобновляется. Если просмотр всех дизъюнктов D_j закончился безрезультатно и новая тройка S_{m+1} не возникла, то либо $m = 1$, и тогда программа завершает работу (т. е. завершает процесс перечисления результирующих

подстановок), либо $m > 1$, и тогда тройка S_m отбрасывается, после чего — переход к следующему шагу. Исключительным является случай, когда дизъюнкт D' оказался константой Л. В этом случае тройка S_{m+1} не вводится, а лишь предпринимается замена k_m на $j + 1$ и происходит выдача очередного результата — подстановки $\sigma_m\sigma'$. Затем — переход к очередному шагу.

Дизъюнкт $D_i = Q_1 \vee \ldots \vee Q_p$ программы на языке ПРОЛОГ можно интерпретировать как последовательность «операторов» $\neg Q_2$, …, $\neg Q_p$, выполняемых в качестве подпрограммы для реализации условия Q_1. Здесь возникают два принципиально разных свойства, отсутствующих у операторов программирования «обычных» алгоритмических языков. Во-первых, отсутствует четкое разделение программных переменных, встречающихся в операторе, на входные и выходные, и в различных ситуациях в одной и той же программе роли таких переменных могут меняться. Во-вторых, оператор реализуется как бы в режиме перечисления своих «выходных» переменных: сначала находится первая версия набора этих значений, реализуются последующие операторы, и если при их обработке возникает в некоторый момент ложное условие, то происходит возвращение к ранее рассмотренному оператору, который выдает очередную версию набора значений выходных переменных, и т. д.

Такой «режим перечисления» существенно упрощает программирование: не только исчезают затрудняющие отладку операторы goto, но и становятся крайне редкими явно выделенные в программе конструкции с циклами, так как эти циклы реализуются автоматически. По-видимому, указанное обстоятельство и предопределило в значительной степени дальнейшее развитие языка ПРОЛОГ — в первую очередь появление в нем большого числа встроенных предикатов, реализуемых интерпретатором (либо компилируемых) без привлечения механизма логического вывода, но с сохранением указанного принципа перечисления выходных значений.

В язык ПРОЛОГ были введены также некоторые дополнительные средства управления ходом выполнения программы, и в конечном итоге программирование на нем значительно приблизилось к программированию на традиционных алгоритмических языках, хотя бы в том смысле, что написанию программы здесь также должна предшествовать разработка алгоритма, учитывающего существенным образом специфику конкретной задачи, включающая в себя оптимизацию применяемой схемы вычислений. Практическая значимость описанной выше процедуры автоматического доказательства теорем как универсального средства решения задач оказалась в результате весьма скромной, будучи явно отодвинутой на второй план предоставляемой языком ПРОЛОГ возможностью программировать «почти без циклов».

Важный этап в развитии логических методов автоматического решения задач, связанный с попытками найти универсальные, предметно-независимые способы борьбы с перебором, возникающим при поиске решения, привел к пониманию невозможности таких способов и необходимости использования в каждой конкретной предметной области своих алгоритмических конструкций для решения задач, учитывающих статистические особенности как данной предметной области в целом, так и рассматриваемых в прикладных ситуациях «потоков задач». По-видимому, выработка решающих правил в предметной области является результатом достаточно трудоемкой и сложной в логическом отношении адаптации, и лишь на уровне анализа общих принципов процессов такой адаптации можно надеяться на обнаружение предметно-независимых процедур, используемых интеллектуальными системами.

Упражнения

2.1. Укажите, какие из следующих выражений являются правильными выражениями (формулами и термами) логики предикатов:

а) x_1;
б) $(f_1(x_2))$;
в) $x_1 \vee x_2$;
г) $(x_1 \vee x_2)$;
д) $(P_1(x_1) \vee P_2(x_3)) \,\&\, (P_1(x_1) \vee P_2(x_2))$;
е) $((P_1(x_1) \vee (\neg P_1(x_2)) \vee P_1(x_3)) \,\&\, (P_1(x_1) \vee P_1(x_2)))$;
ж) $((\forall x_1(P_1(x_1) \vee P_1(x_2))) \,\&\, (\forall x_1(\forall x_2((P_2(x_1) \vee P_1(x_2)))))$.

2.2. Пусть $I = (N, F_1, F_2, F_3)$ — интерпретация, такая, что $F_1(a_n) = n$, $F_2(f_1)(x_1, x_2) = x_1 + x_2$, $F_2(f_2)(x_1, x_2) = x_1 \cdot x_2$, $F_3(P_1)(x_1)$ — предикат, равный истине точно тогда, когда x_1 кратно трем, $F_3(P_2)(x_1) = \mathrm{sign}(x_1 - 1)$. Опишите предикат $I(A)$ для формулы

$$A = ((\exists x_1 P_1(f_1(f_1(x_1, a_3), f_2(x_1, x_2)))) \,\&\, (\forall x_2 P_2(f_2(x_1, x_2)))).$$

2.3. Индукцией по построению формулы докажите лемму 2.1 об интерпретациях.

2.4. Укажите свободные и связанные вхождения переменных в следующие формулы:

а) $(\forall x_3(\forall x_1(P_1(x_1, x_2) \to P_1(x_3, x_1))))$;
б) $((\forall x_1 P_1(x_1, x_3)) \to (\forall x_3 P_1(x_3, x_1)))$;
в) $((\forall x_2(\exists x_1 P_1(x_1, x_2, f_1(x_1, x_2)))) \vee (\neg(\forall x_1 P_2(x_2, f_2(x_1)))))$.

2.5. Определите, свободен ли терм $f_1(x_1, x_2)$ для x_1 в формулах:

а) $(P_1(x_1, x_2) \to (\forall x_2 P_2(x_2)))$;
б) $((\forall x_2 P_1(x_2, a_1)) \vee (\exists x_2 P_1(x_1, x_2)))$.

2.6. Покажите, что следующие формулы не являются общезначимыми:

а) $(((\forall x_1 P_1(x_1)) \to (\forall x_1 P_2(x_1))) \to ((\forall x_1(P_1(x_1) \to P_2(x_1)))))$;
б) $((\forall x_1(P_1(x_1) \vee P_2(x_1))) \to ((\forall x_1 P_1(x_1)) \vee (\forall x_1 P_2(x_1))))$.

2.7. Покажите, что если A и B — формулы логики предикатов, то следующие формулы являются общезначимыми:

а) $(A(t) \to (\exists x A(x)))$, если терм t свободен для x в A;
б) $((\forall x A) \to (\exists A))$;

в) $((\forall x_i(\forall x_j A)) \sim (\forall x_j(\forall x_i A)))$;
г) $((\forall x A) \sim (\neg(\exists x(\neg A))))$;
д) $((\forall x(A \to B)) \to ((\forall x A) \to (\forall x B)))$;
е) $(((\forall x A)\ \&\ (\forall x B)) \sim (\forall x(A\ \&\ B)))$;
ж) $(((\forall x A) \vee (\forall x B)) \to (\forall x(A \vee B)))$;
з) $((\exists x_i(\exists x_j A)) \sim (\exists x_j(\exists x_i A)))$;
и) $((\exists x_i(\forall x_j A)) \to (\forall x_j(\exists x_i A)))$.

2.8. Докажите, что всякая формула A со свободными переменными $x_1, x_2, \ldots, x_n$ выполнима тогда и только тогда, когда выполнима формула $\exists x_1 \ldots \exists x_n A$.

2.9. Докажите, что всякая формула A со свободными переменными $x_1, x_2, \ldots, x_n$ общезначима тогда и только тогда, когда общезначима формула $\forall x_1 \ldots \forall x_n A$.

2.10. Даны формулы логики предикатов:
$A_1 = \exists y \forall x \exists z(R(x, y)\ \&\ R(x, z))$,
$A_2 = \forall z \exists x \forall y(\neg P(y) \vee R(y, x))$,
$A_3 = \forall x \forall y(P(x) \vee R(x, y))$,
$F = (A_2\ \&\ A_3) \to A_1$.
С помощью правила резолюции докажите общезначимость формулы F.

2.11. Даны формулы логики предикатов:
$A_1 = \forall x \exists y \forall z(P(y, z) \vee P(y, x))$,
$A_2 = \exists y \exists x(\neg Q(x)\ \&\ P(x, y))$,
$A_3 = \exists x \forall y \exists z(Q(z)\ \&\ P(z, y))$,
$F = A_1 \to (A_2 \vee A_3)$.
С помощью правила резолюции докажите общезначимость формулы F.

2.12. Даны формулы логики предикатов:
$A_1 = \exists x \exists y \exists z(Q(x)\ \&\ R(x, y, z))$,
$A_2 = \exists x \forall y \forall z \forall v(R(x, f(y), z) \vee R(y, f(y), v))$,
$A_3 = \forall x \exists y \forall z \forall v(P(x, z, y) \vee \neg R(x, z, v))$,
$A_4 = \forall x \forall y \forall z(\neg R(x, y, z) \vee \neg P(x, y, z) \vee Q(x))$,
$F = (A_2\ \&\ A_3\ \&\ A_4) \to A_1$.
С помощью правила резолюции докажите общезначимость формулы F.

Тема 3
КОМПЬЮТЕРНЫЙ РЕШАТЕЛЬ ПОДКОЛЗИНА

Прогресс в развитии вычислительной техники неоднократно приводил к активизации исследований по искусственному интеллекту. Наиболее важными рубежами здесь являются период первого появления электронных вычислительных машин и период распространения персональных компьютеров. Вычислительная техника стала широко доступной, а программирование приобрело массовый характер. Несмотря на это, искусственный интеллект так и не появился, а первоначальный оптимизм в отношении перспектив его создания стал постепенно вытесняться пессимизмом. Успехи и неудачи в программировании определяются на сегодняшний день границей, отделяющей те области, в которых имеются эффективные алгоритмы для решения задач, от областей, для которых эти алгоритмы не созданы. К числу последних относятся, к сожалению, не только творческая деятельность, но даже такие «простейшие» способности человека, как понимание естественного языка и изображений. Успехи здесь пока достаточно скромны.

Вместе с тем отсутствие простого общего алгоритма решения задач в какой-либо из трудных для программирования областей, например школьной геометрии, не является препятствием для решения таких задач человеком. Хотя общего алгоритма в учебниках и нет, но в каждом отдельном случае логика принятия решения вполне поддается анализу и объяснению. Обучаясь на сотнях и тысячах примеров, постоянно пополняя и корректируя свои умения, человек постепенно аккумулирует в себе необходимый ему алгоритм, нигде не изложенный в явном виде. Чтобы получить аналогичный алгоритм в компьютере, остается только следовать этому примеру. Видимо, это и есть реальный способ создавать программы «там, где нет алгоритмов». Если не овладеть таким искусством, то никакое, даже самое впечатляющее развитие вычислительной техники сколь-нибудь существенным образом положения дел с искусственным интеллектом не улучшит.

Алгоритмы, извлекаемые человеком из процесса обучения на примерах, сильно отличаются от алгоритмов обычного программирования. Прежде всего, они почти лишены циклов — на каждом шаге для решения задачи привлекаются какие-то новые знания и приемы, зачастую из совершенно различных областей. Алгорит-

мы эти плохо декомпозируются на блоки. Скорее, они представляют собой что-то типа векторного поля приемов, в котором перемещается решаемая задача и которое создается путем длительной трудоемкой «дрессировки». Срабатывание каждого приема продвигает задачу на один шаг вперед по ее траектории. До тех пор, пока не получено достаточно полной коллекции приемов, обеспечивающей с большой вероятностью «цепную реакцию» процесса решения задачи, эти алгоритмы практически бесполезны. А так как обычно критический минимум достаточно велик и насчитывает многие тысячи элементов, то возникает серьезный психологический барьер — у программиста после проработки первых сотен задач может возникнуть впечатление, что проект безнадежен и его следует бросить.

Еще одно отличие алгоритмов рассуждений от алгоритмов вычислений связано со способом кодирования информации. Вычислительная программа обычно использует технические структуры данных, семантика которых остается за кадром, так как непосредственно при вычислениях не нужна. Алгоритмы интеллектуальной системы анализируют задачу независимо друг от друга, и, чтобы они могли понимать текущий контекст, необходимо сохранить всю его семантику, используя для представления данных универсальный логический язык. Решение трудных для алгоритмизации задач — это почти всегда процесс преобразования логических структур данных или, иными словами, логический процесс.

История развития математики тесно связана с изучением процессов различных типов. Многие разделы классической «непрерывной» математики, и в первую очередь теория дифференциальных уравнений, возникли для изучения физических процессов. Изучение процессов обработки дискретной информации в электронике привело к появлению теории автоматов и других разделов математической кибернетики. К сожалению, логические процессы оказались настолько сложны, что попытки изучения их математическими методами так и не привели к созданию эффективных решателей в сколь-нибудь нетривиальных областях. Поэтому единственным подходом к созданию науки о логических процессах на сегодняшний день остается компьютерное моделирование, т. е. указанная выше «дрессировка» логических векторных полей на потоках обучающих примеров.

Создание больших решателей вручную чрезвычайно трудоемко, так как требует постоянной регулировки взаимодействий в многотысячной системе активных элементов. Такие решатели, как правило, изначально не очень качественные и требуют последующей, еще более трудоемкой оптимизации. Однако они дают материал, позволяющий начать исследования по автоматизации синтеза приемов и саморазвитию логических векторных полей, без которого невозможно было бы создание подлинно интеллектуальной системы.

Объектом изучения здесь являются процессы синтеза алгоритмов по теоремам и процессы автоматического развития теорий, ориентированного на синтез алгоритмов. Это еще одна разновидность логических процессов.

В данной теме будут даны основные представления о компьютерном решателе математических задач, разработанном А. С. Подколзиным. Основная рекомендуемая литература по этой теме составляет [5—8]. Также на сайте http://intsys.msu.ru/invest/solver доступна для загрузки компьютерная версия решателя.

3.1. Компьютерное моделирование логических процессов

Логические процессы играют исключительно важную роль в поведении любой интеллектуальной системы. Они необходимы для распознавания зрительных либо звуковых образов, управления динамикой системы, понимания языка, планирования действий, анализа потока событий, решения технических и теоретических задач. Хотя основной объем обработки информации во многих случаях берут на себя средства автоматики, использующие не логические, а «технические» структуры данных, логика играет роль диспетчера, организующего и контролирующего работу этой автоматики, а в необходимых случаях обеспечивает ее развитие.

В тех случаях, когда для решения задачи имеется заранее выработанный план действий, обычно логический вывод не требуется — вместо него создается специализированный вычислительный алгоритм. Это обычный и хорошо известный путь развития программирования, на котором создаются различные системы численного моделирования, компиляции и оптимизации технических проектов, управления сложными системами и т. п.

Потребность в логических процессах появляется лишь при отсутствии известного заранее плана действий. Тогда планирование действий происходит непосредственно во время решения задачи, причем для обеспечения возможности нахождения приемлемых вариантов система должна располагать достаточно обширным и разнообразным запасом знаний. Использование того или иного элемента этого запаса знаний заранее непредсказуемо — задача из одной предметной области может потребовать для своего решения весьма удаленных от этой области идей. По этой причине вряд ли возможно создавать подлинно интеллектуальные системы для ограниченных классов задач — например, только для распознавания изображений, или только для понимания естественного языка, или только для написания программ, синтеза «чипов» и т. д. Чтобы достаточно эффективно действовать хотя бы в одной из таких

областей, необходимо заложить в интеллектуальную систему универсальное «фундаментальное» образование, как это и происходит обычно при обучении человека. Уже один только количественный аспект этой не решенной на сегодняшний день задачи далеко выводит ее за рамки трудностей традиционного программирования. Заметим, что даже в случае обучения человека, с отлаженной и оптимизированной веками системой образования, процесс подготовки полноценного специалиста (с учетом весьма существенного дошкольного периода) занимает более 20 лет.

Попытки исследования логических процессов математическими методами привели к появлению и развитию такой науки, как математическая логика. В ее рамках удалось дать точные определения фундаментальным понятиям логического языка, аксиоматической теории, формального доказательства, алгоритма решения задачи, сложности вычислительной процедуры. Был накоплен значительный запас математических результатов, характеризующих общие свойства этих понятий. Возникли новые представления о математической строгости и формализации теорий, которые привели к переизложению многих разделов математики в виде формальных дедуктивных систем. Изучение общих свойств этих формальных теорий методами математической логики позволило решить ряд важных общематематических проблем: доказать независимость континуум-гипотезы; установить алгоритмическую неразрешимость диофантовых уравнений и др. Однако основные продвижения математической логики оказались связаны не столько с логической «динамикой», сколько с логической «статикой». Несмотря на интенсивные исследования в области процедур автоматического доказательства теорем, не удалось не только предложить эффективно работающие общие процедуры решения задач, но даже найти общие принципы, на основе которых такие процедуры могли бы быть созданы. Основной проблемой, с которой здесь пришлось столкнуться, явилась проблема трудоемкости поиска решения задачи по деревьям прямого либо обратного логического вывода, экспоненциально возрастающей с ростом глубины поиска. Всевозможные результаты по отсечению ветвей рассматриваемого дерева для уменьшения объема перебора и различные эвристические общие принципы упорядочения перебора не изменили ситуации сколь-нибудь ощутимым образом.

В сложившейся ситуации, когда поиски математических методов, позволяющих создавать эффективные решатели задач, фактически зашли в тупик, единственной возможностью создания таких решателей становится компьютерное моделирование логики рассуждений человека. Компьютерная модель здесь должна сыграть роль «микроскопа» для изучения процессов поиска решения и самообучения, с помощью которого были бы выявлены фундаментальные

принципы их организации. Возможно, что по мере развития такой «экспериментальной» теории логических процессов вновь окажутся востребованы и математические методы исследования.

3.1.1. Моделирование логической автоматики

Компьютерное моделирование логических процессов естественно начинать с моделирования автоматики, управляющей ходом рассуждений, оставляя пока в стороне значительно более сложные процессы саморазвития этой автоматики. Чтобы разрабатывать механизмы самообучения и самопроектирования, необходимо прежде всего получить определенный запас созданных вручную «эталонных» алгоритмических конструкций, моделирующих процессы решения задач. По-видимому, без таких «образцов для подражания» трудно определить необходимый для эффективного функционирования запас архитектурных стандартов, который могла бы использовать процедура самопрограммирования системы. Многократно предпринимавшиеся ранее попытки создания саморазвивающихся интеллектуальных систем, основанных на тех или иных простых общих принципах, без анализа реальной логической автоматики во всей ее сложности и разнообразии оказывались вполне бесплодными.

В самых общих чертах процесс расшифровки логической автоматики, обеспечивающей решение задач в некоторой предметной области, можно представить происходящим по следующей схеме. Рассматривается некоторая обучающая последовательность задач в этой области, решение которых известно эксперту, осуществляющему развитие компьютерной модели. Предпринимается анализ траектории решения очередной задачи из выбранной последовательности. Если эта задача не представляет собой простейший тест, для которого имеется заранее известный стандартный план действий, то процесс ее решения складывается из отдельных этапов, состоящих из локального планирования и реализации выработанного плана действий.

Каждый такой этап локального планирования анализируется, и для него предлагается некоторый гипотетический алгоритм, приводящий в рассматриваемой ситуации к тому же результату. Эти алгоритмы накапливаются в компьютерной модели, так что в новой задаче некоторые из них могут сработать и предложить свою версию развития решения. Каждое такое срабатывание анализируется на предмет целесообразности, и в решающие правила алгоритма локального планирования вводятся необходимые коррективы. Эта процедура повторяется для многих сотен задач предметной области — до тех пор, пока сложившаяся мозаика алгоритмов локального планирования действий не обнаружит достаточно устойчивого и эффективного поведения. В целом процесс напоминает обучение

нейросистемы методом «поощрений и наказаний»; отличие состоит лишь в более сложной процедуре коррекции, которая не сводится к изменению каких-либо числовых параметров, как у нейрокомпьютера, а требует привлечения логики для очередного перепроектирования алгоритмов планирования.

Многообразие алгоритмов локального планирования создает своего рода логическое векторное поле, в котором перемещается решаемая задача — от исходной формулировки к ответу. Разумеется, это лишь упрощенная аналогия; отдельные шаги в таком перемещении могут представлять собой целые иерархии вложенных логических процессов, с различными циклами ограниченного поиска и отбора. Однако она отражает главное обстоятельство: по завершении реализации каждого локального плана предпринимается независимое рассмотрение всей текущей ситуации заново для выработки следующего плана. К сожалению, на сегодняшний день неизвестно другого способа создания эффективных логических векторных полей автоматического решения задач кроме весьма трудоемкого процесса их «дрессировки» на тысячах обучающих примеров.

Так как алгоритмы локального планирования (для краткости далее условимся называть такие алгоритмы *приемами решения задач*) берут всю инициативу на себя, не имея какого-либо стоящего над ними управляющего центра (ведь никто, кроме конкретного приема, и не располагает достаточной для принятия решения в области его компетенции информацией), то особо важную роль приобретает точный учет взаимодействий различных приемов в цепи решения задачи. Трудоемкость этого учета в среднем должна быть пропорциональна квадрату числа приемов, по крайней мере в той предметной области, где проводится обучение. Разумеется, адекватно большим должен быть и обучающий материал — иначе качество работы решателя будет весьма посредственным.

По-видимому, единственной возможностью преодоления этого недостатка логической автоматики (как и вообще любой автоматики, ограниченной определенными разумными рамками) является развитие средств ее автоматического пополнения и оптимизации, которые подключались бы при решении задачи по мере надобности. Анализ многих и многих математических задач так называемого нестандартного характера показывает, что выход за рамки накопленных запасов приемов (обычно связанный с попыткой скомбинировать несколько известных теорем и извлечь из них новую идею) является, в общем, обыденным явлением и даже по своей трудоемкости иногда вполне сопоставим с работой автоматики стандартных приемов. Впрочем, автоматика мышления в поведении любых интеллектуальных систем играет огромную роль, и лишь в ситуациях аварийного типа включаются дополняющие ее средства более высоких уровней.

Описанная выше общая схема моделирования логической автоматики была реализована на примере процессов решения математических задач. Причины, по которым на первом этапе была рассмотрена математика, вполне понятны: именно в ней при решении задач возникают наиболее сложные и разнообразные логические процессы, причем имеются огромные запасы обучающего материала. В последнее время начато моделирование логических процессов и в ряде нематематических областей. Было проработано свыше 9 тыс. задач из таких разделов, как элементарные алгебра и геометрия, математический анализ, аналитическая геометрия, линейная алгебра, дифференциальные уравнения, теория вероятностей, комплексный анализ, элементарная физика, школьные текстовые задачи (с синтаксическим и семантическим анализом текста). На основе этого обучающего материала сформирована база приемов решателя задач, насчитывающая в настоящее время более 25 тыс. приемов.

Фактически возникла мощная система символьной компьютерной математики нового типа, позволяющая не только получать ответы, но и прослеживать ход решения по шагам, а при необходимости вмешиваться в процесс решения. В отличие от традиционных систем компьютерной математики, где накапливались тысячи функций для пользователя, который должен был вручную находить их в каталоге системы и применять к текущей задаче, новая система накапливает тысячи приемов, которые самостоятельно анализируют задачу и принимают решение о выполнении преобразований. Пользователь здесь может вообще не интересоваться содержимым базы приемов. Разумеется, это более высокая степень автоматизации, и как следствие нового подхода, системе становятся доступны значительно более сложные задачи.

Преимущества нового подхода становятся особенно заметны в тех разделах, где велика роль логического вывода. В элементарной алгебре это решение уравнений и неравенств с параметрами; в математическом анализе — качественное исследование функций с помощью пределов и производных, исследование сходимости рядов, вычисление кратных интегралов и применение их для нахождения площадей и объемов. Разумеется, наиболее интересными с точки зрения моделирования логических процессов оказались элементарная и аналитическая геометрии. В распространенных системах компьютерной математики эти разделы представлены совсем слабо, причем элементарная геометрия по существу отсутствует, а аналитическая — связана лишь с выполнением стандартных операций явно одношагового характера.

Работа над моделированием процессов решения математических задач потребовала создания целого технологического комплекса, позволившего существенно приблизить язык для программирования приемов к языку для формулировки теорем предметной обла-

сти. Фактически прием задается как теорема, снабженная системой указателей для компилятора, генерирующего на основе этой теоремы программу приема. Задачей компилятора является синтез такой программы, которая была бы способна усматривать возможность применения теоремы даже в неявных, замаскированных ситуациях, оценивать целесообразность такого применения и осуществлять его.

Программирование приемов имеет рекурсивный характер: в описании приема допускаются обращения к всевозможным вспомогательным процедурам и задачам, применяемым к фрагментам теоремы. Это означает, что при реализации приема может быть использована не только та теорема, на которой он основан, но и сотни других теорем, востребованных его вспомогательными операторами. Использование стандартной математической записи для изображения теоремы приема, а также сопровождение геометрических теорем чертежами позволили приблизить наглядность описания приема к наглядности текста решаемой задачи. Сочетание этой наглядности с развитием системы семантической трассировки процесса решения, выдающей на экран подробные объяснения выполняемых преобразований, значительно уменьшило трудоемкость процесса анализа решений и коррекции приемов.

Вместе с тем новый язык для записи приемов не просто создает определенные преимущества для процесса обучения решателя. В действительности его приближенность к логическому языку предметной области ориентирована в первую очередь на последующие исследования в области автоматического синтеза приемов; на весь круг проблем, относящихся к пограничному слою между логикой и алгоритмами.

Фактически развитие техники обучения сделало доступным продолжение обучения системы ее пользователям так же, как и в случае обычных систем компьютерной математики, которые предоставляют пользователю средства для программирования новых функций. Отличие здесь заключается в том, что вместо мало приспособленных для логических процессов традиционных средств процедурного программирования развит мощный комплекс программирования продукционного. Чтобы сравнить уровень используемого языка с уровнем известного языка логического программирования ПРОЛОГ, заметим лишь, что программы на промежуточном языке уровня языка ПРОЛОГ возникают как результат компиляции описания приема; естественно, они значительно больше и в сравнении с первоначальным описанием совершенно нечитабельны.

3.1.2. Логический язык для представления задач

Собственно работе по моделированию логических процессов в математике предшествовало решение ряда важных технических

проблем, определившее общую архитектуру компьютерной системы. Прежде всего необходимо было выбрать логический язык для представления задач. Разумеется, для адекватного моделирования процессов решения такой язык должен быть возможно более приближен к естественному математическому языку. Языки математической логики, создававшиеся для теоретического исследования свойств аксиоматических теорий, обычно вводились как можно менее избыточным образом — для упрощения связанных с ними формулировок теорем и доказательств этих теорем. Использование таких языков в решателе привело бы к неоправданному усложнению формулировок и повлекло бы существенные расхождения между моделью и моделируемым процессом.

Фактически оказалось, что, несмотря на огромное количество работ в математической логике, посвященных формализации различных разделов математики, никакой практически пригодной формализации «сквозного» характера, сохраняющей все введенные для удобства логических вычислений в обычной математической практике условности, создано не было. Поэтому для решателя пришлось создавать свой собственный логический язык, пополняемый по мере проработки новых предметных областей. Одним из важных наблюдений, которые здесь были сделаны, оказался определенный конфликт между требованиями к языку, предъявляемыми к нему традиционными для формальной логики соображениями логической аккуратности, с одной стороны, и соображениями вычислительной целесообразности — с другой.

Аккуратная логическая запись иногда становится избыточно громоздкой, и в определенных контекстах ее можно заменить упрощенными суррогатами, достаточными для корректных вычислений. Возникло, таким образом, явление «контекстной семантики» языка, когда смысл того или иного утверждения задачи определяется лишь в контексте всей задачи в целом. Простейшим примером такого рода является обычное использование в математическом анализе утверждений вида $f(x) = O(g(x))$, для понимания смысла которых в контексте должно содержаться что-нибудь вроде $x \to 0$.

Учет контекста при обработке логических структур данных позволяет вводить в язык различного рода нечеткие понятия, что часто бывает необходимо в нематематических областях. Для уточнения их смысла в конкретной ситуации можно обращаться к сопровождающей информации, как логической, так и технической (например, анализировать изображения, протоколы и т. п.). Разумеется, какие-либо формальные аксиоматические теории для таких понятий будут отсутствовать, однако в процессе «дрессировки» системы на обучающем материале можно выработать достаточно хорошие алгоритмические правила правдоподобных рассуждений, аппроксимирующие действия эксперта. Таким образом, созданный

аппарат обучения баз приемов никоим образом не ограничен рамками хорошо формализованных предметных областей.

Начатый при рассмотрении математических задач процесс создания некоторого всеобъемлющего логического языка, используемого решателем, предполагает продолжение и на нематематические области, включающие нечеткие понятия и нечеткие рассуждения. Для работы с нечеткой логикой, по-видимому, потребуются адаптивные алгоритмические конструкции, компенсирующие нечеткость понятий аккуратным учетом контекста и накапливающие «четкие» аппроксимации нечетких понятий в виде сложных логических описаний образов, стоящих за такими понятиями. Заметим, наконец, что работа над указанной выше широкой логической формализацией, с развитием многообразия сопутствующих приемов, совершенно необходима для создания системы, способной адекватно понимать естественные языки.

Чтобы при обучении решателя легко распознавались отклонения его реального поведения от желаемого, необходима хорошая визуализация процесса решения. В частности, при отображении на экране математических утверждений, сформулированных на внутреннем логическом языке решателя, необходим перевод этих утверждений на язык общепринятой математической символики. Обратный перевод также является весьма существенным, так как ускоряет ввод нового обучающего материала и новых приемов. В системе реализован редактор математических формул, обеспечивающий быстрый ввод математических утверждений и выражений, а также осуществляющий перевод во внутреннее логическое представление и обратно. Размеры элементов вводимой формулы определяются и корректируются в процессе ввода автоматически; широко используется ввод длинного термина с помощью нескольких нажатий клавиш (в большинстве случаев — первые две-три буквы термина). Это позволяет набирать даже весьма громоздкие на вид выражения с помощью сравнительно малого числа нажатий клавиатуры.

Так как для подавляющего большинства рассматриваемых в неэлементарной математике понятий не предусмотрена специальная символика, пришлось вводить в формульном редакторе словесное их представление (как правило, такое понятие F используется в скобочной записи вида $F(t_1, \ldots, t_n)$). Логика действий формульного редактора достаточно сложна, и хотя он реализован в духе обычного процедурного программирования, но фактически сам является небольшим решателем. По-видимому, для увеличения динамики обмена информацией с системой было бы полезно более явным образом ввести логические процессы в процедуры визуализации и перевода формульно-графических данных на внутренний язык, обучая соответствующие редакторы примерно так же, как обучаются решатели задач.

3.1.3. Формализация понятия задачи

Следующий вопрос после выбора логического языка — формализация понятия задачи, достаточная для работы с обучающим материалом. На начальном этапе исследований по искусственному интеллекту предпринимались многочисленные попытки дать математически строгие определения понятия задачи, решения задачи и ответа на задачу.

В самом общем виде задача представлялась как логическое условие $P(x)$ на элементы x некоторого универсального множества объектов U; ответ на задачу — как утверждение логического языка, определяющее в некоторых «явных» терминах конкретный элемент x множества U, для которого $P(x)$ истинно; решение задачи — как цепочка преобразований исходного условия, преобразующая его к виду ответа. Вводя в универсум U классы объектов, в таком виде легко представить не только задачу поиска единственного элемента некоторого множества, удовлетворяющего заданному условию, но и задачу описания всех таких элементов.

Однако это простое и, казалось бы, весьма общее представление о задаче сразу же сталкивается с трудностями при сопоставлении его даже с простейшими реальными задачами по элементарной алгебре. Так, при решении задачи на упрощение алгебраического выражения вовсе не накладывается какого-либо строгого ограничения $P(x)$ на предъявляемый школьником ответ x — например, не требуется предъявления доказательства того, что этот ответ минимален по числу символов или по какому-либо другому функционалу сложности выражения. Все, что в таких случаях требуется, — это применять определенный запас стандартных приемов упрощения до тех пор, пока дальнейшие попытки не будут давать каких-либо улучшений, и выдать полученное выражение в качестве ответа. Этот и другие примеры приводят к выводу, что более адекватным является представление о задаче как о сочетании некоторого строгого условия на допустимость ответа с рядом целевых установок на его оптимизацию, учитываемых при решении по мере возможности. Тогда понятие ответа становится субъективным, зависящим от уровня обученности системы. Впрочем, при аккуратной формализации целевых функционалов появляется возможность некоторой «объективизации», заключающейся в последовательном обучении или самообучении для повышения качества ответов.

Даже в хорошо формализованных областях реально используемое понятие задачи является в значительной степени нечетким. Целевая установка на проведение исследования в математике или задание на разработку проекта в технике, как правило, требует существенного доопределения, опирающегося на всю сумму знаний из рассматриваемой области. Значительную часть таких знаний составляют различного рода условности и эвристические приемы, не-

обходимые для адекватной расшифровки и уточнения задания уже в процессе его реализации. Поэтому при разработке архитектуры решателя следует в первую очередь четко определить лишь структуры данных, которые будут определять «текущее состояние» задачи в процессе ее решения, а допустимые шаги преобразования задачи и способ усмотрения ответа на нее, достигаемого в конце цепочки таких преобразований, уточнять по мере проработки конкретного обучающего материала.

Применительно к логическим процессам понятие задачи должно быть определенным образом ограничено. Если речь идет о разработке технической системы, написании программы, составлении плана действий и т. п., то синтезируемый объект обычно задается не на логическом, а на специализированном техническом языке. Такое задание часто имеет иерархический характер, причем на каждом уровне структуры данных приспособлены для быстрых вычислений, необходимых при компиляции и оптимизации проекта. Собственно логические процессы возникают тогда, когда заранее созданных средств вычислительной автоматики оказывается недостаточно. В этом случае из всего контекста технической информации извлекаются данные, необходимые для «локального» логического рассмотрения , формулируется задача, и после ее решения вносятся необходимые коррективы в технические структуры данных.

Роль решателя задач, реализующего логические процессы, здесь можно уподобить роли головки машины Тьюринга, выполняющей вычисления на ленте, — она сугубо локальна. С этой точки зрения процессы проектирования или исследования представляют собой макрозадачи — обращения к решению задач (в указанном выше узком смысле применения логического процесса для получения ответа) в них являются лишь отдельными атомарными шагами и тесно переплетаются с нелогическими процессами. Вообще, взаимодействие логических и нелогических процессов в интеллектуальных системах происходит практически постоянно. Для объектов, упоминаемых в логических структурах данных, создаются модели (мысленные или иные). Наблюдение за структурой и функционированием моделей подсказывает гипотезы, направляющие ход рассуждений; при проектировании — позволяет поставить новые задачи, после решения которых предпринимается коррекция модели, и т. п. Важной проблемой, связанной с использованием нелогических моделей объектов и процессов, оказывается проблема перевода результатов наблюдения за моделью на логический язык. Частными случаями этой проблемы являются распознавание изображений и анализ динамических образов.

Предварительная классификация «логических типов» задач, возникшая при анализе математических предметных областей, приве-

ла к рассмотрению четырех *основных типов задач*: на доказательство, на преобразование, на описание, и на исследование. Любая такая задача имеет, прежде всего, некоторый (возможно, пустой) список утверждений относительно встречающихся в ней известных объектов, истинность которых считается априори данной. Эти утверждения будем называть *посылками* задачи. Типичные примеры списков посылок — перечисление условий на известные параметры в системе уравнений; логическое описание чертежа в геометрической задаче на вычисление; список данных утверждений в задаче на доказательство.

В зависимости от типа задачи список посылок сопровождается рядом дополнительных элементов. В случае задачи на доказательство добавляется то утверждение, относительно которого требуется установить, что оно является следствием посылок; будем называть такое утверждение *условием* задачи на доказательство.

В случае задачи на преобразование добавляется то выражение (называемое ее условием), которое должно быть преобразовано к некоторому специальному виду в предположении истинности посылок. В отличие от задачи на доказательство, здесь возникает необходимость сформулировать целевую установку, уточняющую желаемые вид и оптимизацию ответа (упростить; разложить на множители; проинтегрировать, и т. п.). Эта формулировка уже не относится к тому логическому языку «предметного уровня», на котором задаются посылки и условие. Хотя ее можно было бы задавать на некотором логическом языке, предназначенном для записи утверждений о логических структурах данных, на практике оказалось более удобным представлять целевую установку задачи как список специальных технических «пометок», называемых далее *целями* задачи. Это объясняется тем, что учетом целевых «пометок» занимается автоматика, управляющая логическим процессом и, как всякая автоматика, использующая свои узкоспециализированные нелогические структуры данных.

Задача на описание — это обобщение таких типов задач, как решение системы уравнений или неравенств. У нее, кроме списка посылок, имеется также некоторый список утверждений с неизвестными — они называются далее *условиями* задачи. Требуется дать описание всех или части значений неизвестных, при которых выполнены условия. Как и в случае задачи на преобразование, списки посылок и условий задачи на описание приходится сопровождать целевой установкой. Она уточняет вид искомого описания; определяет, нужно ли получить полное описание или достаточно лишь частичного (например, единственного примера значений неизвестных).

Ответ задачи на описание обычно достигается в процессе последовательных преобразований ее списка условий. Однако во мно-

гих случаях для получения ответа бывает необходимо накапливать некоторое многообразие следствий объединенного списка ее условий и посылок. В таком режиме решаются системы уравнений: извлекая следствия из исходных уравнений, иногда удается получить равенства, указывающие значения неизвестных. Этот же режим определения значений неизвестных путем вывода следствий типичен для геометрических задач на вычисление. Занесение следствий непосредственно в список условий задачи нежелательно, так как впоследствии пришлось бы расчищать этот список, исключая все избыточные его элементы, что потребовало бы дополнительных вычислительных затрат на проверку избыточности. Поэтому в структуре данных задачи на описание предусмотрен специальный накопитель следствий условий и посылок. Роль такого накопителя играет вспомогательная задача на исследование (см. далее), вводимая в процессе решения задачи на описание. Изначально она имеет своими посылками все посылки и условия задачи на описание.

Задача на исследование, в дополнение к списку посылок, имеет лишь целевую установку, уточняющую направленность логического вывода в этом списке. При решении ее исходное логическое описание некоторой ситуации в том или ином смысле упрощается и пополняется утверждениями, представляющими интерес в контексте целевой установки. Этот процесс обрывается либо по исчерпании возможностей добавления к имеющейся «картине» каких-либо ценных новых фактов, либо при получении следствий, требующих немедленного возвращения к внешней задаче, для которой предпринимается исследование.

Заметим, что в решателе задачи на исследование используются только как вспомогательные — например, в роли накопителя следствий условий и посылок задачи на описание; в роли накопителя следствий при доказательстве от противного некоторого утверждения и т. д. Различные математические задачи на исследование — такие как исследование поведения функции вещественной переменной с помощью пределов и производных; исследование вида кривой или поверхности, заданной своим уравнением, и т. п. — оказалось целесообразно представлять в виде задачи на описание, сводя ее решение практически целиком к выводу следствий в связанной с ней задаче на исследование и отбирая затем в качестве результата лишь некоторые из полученных фактов.

Кроме того, отметим, что проведение исследований согласно заданной целевой установке для получения новых теоретических утверждений вообще не является задачей в рассматриваемом здесь узком смысле и никак не связано с применением введенных выше задач на исследование. Различные теоретические факты в математике являются замкнутыми логическими конструкциями, не имеющими общих варьируемых параметров, в то время как в рамках

одной задачи обычно объединяются лишь такие утверждения, которые относятся к общим варьируемым переменным. Поэтому попытки организации исследований в теории как процесса логического вывода в списке посылок некоторой задачи не выявили никаких преимуществ и лишь привели к неоправданному замедлению вычислений. Процесс исследований в теории представляет собой макрозадачу, для которой постановка обычной задачи и ее решение являются лишь атомарным шагами.

Кроме логических структур данных, задача содержит также некоторые вспомогательные технические структуры данных, вводимые в процессе работы самим решателем и используемые для сохранения информации, направляющей его действия. Прежде всего это пометки, указывающие на различные ранее предпринимавшиеся неудачные попытки, блокирующие их повторение, а также сообщения управляющего характера, которыми обмениваются между собой приемы. Для ускорения поиска в задаче объектов, связанных определенным образом с заданным объектом, создаются древовидные адресные конструкции. Работа приемов сопровождается потоком многократных обращений к различным вспомогательным процедурам, осуществляющим быстрые логические вычисления. Результаты таких обращений сохраняются в специальных буферах, что существенно ускоряет работу системы и упрощает организацию взаимодействия между приемами. Фактически эти буферы создают что-то типа самоорганизующейся сетевой структуры данных, позволяющей быстро определять свойства объектов и связи между ними. Их применение оказалось особенно эффективным в геометрических задачах, насчитывающих сотни посылок.

3.1.4. Приемы

Преобразования задачи, а также ввод и рассмотрение различных вспомогательных задач в процессе решения обеспечиваются *приемами* — процедурами локального анализа ситуации, способными усматривать целесообразность определенных действий и выполнять эти действия. Таких приемов в процессе обучения накапливаются многие тысячи, и для обеспечения быстрого поиска нужного приема необходима специальная организация базы приемов. Большинство приемов, используемых при решении задач, допускает явное указание такого понятия, что для возможности применения приема необходимо появление данного понятия в задаче. Это позволяет организовать базу приемов по принципу энциклопедии: за каждым понятием логического языка закрепляется сравнительно небольшая группа приемов этого понятия. Как показывает опыт, исключение составляет лишь крайне незначительная группа приемов, для которых не выделяется какого-либо естественного «инициализирующего» понятия.

Очередной шаг решения задачи состоит в последовательном, понятие за понятием, просмотре всего текста задачи, с обращением для каждого текущего понятия к поиску внутри группы его приемов. Эта группа невелика, и во многих случаях ее можно организовать по древовидному принципу, получая таким образом дополнительное ускорение поиска. Преимуществом данного подхода является то, что при обучении системы новым разделам мы практически не уменьшаем скорости ее работы на ранее проработанных разделах. Это происходит потому, что «новые» понятия не встречаются в «старых» задачах, и наличие даже очень большого числа «новых» приемов никак не сказывается на времени поиска «старых», связанных с другими понятиями. Фактически здесь снимаются сколь-нибудь сильные ограничения на рост числа приемов и появляется принципиальная возможность создавать гигантские логические системы, имеющие фундаментальное разностороннее образование. Заметим, что нынешняя версия решателя, охватывающая многие разделы элементарной и высшей математики и насчитывающая более 25 тыс. приемов, занимает всего лишь порядка нескольких десятков мегабайт, в то время как используемая организация базы приемов и размеры жесткого диска компьютера позволяют довести ее размеры до гигабайт без существенного ухудшения быстродействия.

Таким образом, рабочий цикл решателя состоит в сканировании текста задачи — последовательном просмотре встречающихся в задаче понятий и обращении для текущего понятия к его программе, объединяющей в себе все закрепленные за понятием приемы. Эта процедура представляет собой что-то типа внутреннего «логического зрения», обеспечивающего контроль за развитием событий. Как и человек, в процессе решения система предпринимает «мысленное рассмотрение» объектов задачи и связей между ними для выработки плана ближайших действий.

Чтобы выбрать из всех возможных действий наилучшее, необходимо каким-то образом сравнивать между собой результаты применения различных приемов в текущем контексте. Для этого естественно ввести числовые оценки приоритетности, выбирая каждый раз, например, прием с наименьшей такой оценкой. Вообще говоря, нерационально при сканировании задачи находить все возможные варианты применения приемов и лишь по окончании сканирования отбирать лучший — это может потребовать рассмотрения слишком большого числа вариантов, причем по своей априорной ценности анализируемые варианты тоже будут сильно различаться. К меньшим вычислительным затратам можно прийти, если разбить все приемы на группы или уровни, объединяя более-менее равноценные, и поиск нужного приема вести с последовательным увеличением номера уровня, обрывая его, как только найден применимый в текущей ситуации прием.

Фактически здесь оценкой приоритетности приема становится номер уровня. Так как обработка каждого уровня приемов требует своего цикла сканирования задачи, то число уровней целесообразно сделать не очень большим; в решателе это число равно 16, причем, как правило, срабатывание приема происходит на первых 5—6 уровнях. В действительности один и тот же прием может в различных ситуациях относиться к различным уровням — номер уровня определяется, в зависимости от контекста, самой программой приема. Если этот уровень не соответствует тому, для которого выполняется текущее сканирование задачи, то дальнейшие действия по рассмотрению приема обрываются и система переходит к другим приемам.

Важную роль при сканировании задачи играет управление переключением внимания. Вообще говоря, различные элементы описания задачи неравноценны при поиске очередного приема. Одни из них находятся в задаче давно, другие — только что занесены либо изменены. Вероятность обнаружить при рассмотрении первых ситуацию, требующую немедленных действий, обычно значительно ниже, чем для вторых. Чтобы учесть такую статистическую неоднородность элементов задачи и уменьшить среднее время поиска приема, элементы задачи (посылки и условия) снабжаются весами — целыми неотрицательными числами, указывающими номер того уровня сканирования, до которого ранее доходило рассмотрение элемента. Как только элемент изменяется, его вес уменьшается до нуля; веса новых элементов также полагаются равными нулю.

При сканировании, связанном с приемами i-го уровня, игнорируются все посылки и условия, веса которых больше i — они образуют «теневую» часть задачи. По мере прохождения i-го уровня веса тех элементов задачи, для которых не произошло срабатывание приема, автоматически увеличиваются до $i + 1$. Как только срабатывает какой-либо прием и возникают элементы задачи с весом, равным нулю, сканирование задачи возобновляется начиная с нулевого уровня. При этом в «зону внимания» попадут только те элементы задачи, которые имеют нулевой вес, т. е. только что были изменены либо введены.

Эта простая автоматика уже обеспечивает в большинстве случаев разумную стратегию переключения внимания и значительно увеличивает быстродействие системы. В особенности это ощутимо для задач геометрического типа, имеющих сотни посылок и условий. Конечно, необходим ряд дополнительных средств, обеспечивающих переключение внимания в специальных случаях. Некоторые из этих средств включены в «общую автоматику» решателя; другие — реализованы непосредственно в приемах, которые могут избирательно изменять веса различных элементов задачи и таким образом управлять зоной внимания.

3.1.5. Язык для записи приемов

Следующий важный вопрос, возникающий после выбора механизма сканирования задачи как диспетчера, организующего работу базы приемов, — это вопрос о языке для записи приемов. В приеме выделяются такие основные части, как описание ситуации, в которой логически допустимы реализуемые им действия; описание ситуации, в которой эти действия целесообразны, и описание процедуры, собственно реализующей действия приема. Первые две части предполагают использование некоторого логического языка, на котором формулируются соответствующие условия. Эти условия относятся не к объектам предметной области, рассматриваемым в задаче (числам, точкам, векторам и т. п.), а к тем структурам данных, с помощью которых задача представлена в системе: к термам логического языка, вхождениям в эти термы, к спискам термов, к различного рода техническим пометкам и конструкциям, вводимым решателем для управления процессом.

Чтобы формулировать такие условия, необходимо обеспечить достаточно богатый набор понятий логического языка, ориентированных на используемые в решателе структуры данных, и прежде всего — на задачи и их элементы. Именно этот набор и составил основную часть нового языка ЛОС (*логический описатель ситуаций*) для записи приемов; чтобы задавать те действия, которые должны быть выполнены приемом в уже «опознанной» им ситуации, оказалось достаточно присоединить к логической части языка сравнительно небольшое количество кодирующих типовые преобразования конструкций.

После того как на логическом языке сформулирована ситуация, в которой применение приема возможно и целесообразно, необходимо обеспечить некоторый алгоритм, реализующий в задаче поиск этой ситуации. В описании ситуации должны фигурировать объекты, идентифицированные еще до начала поиска, — сама задача; выделенное в ней при сканировании вхождение понятия; текущий уровень сканирования и т. п.

Описание представляет собой последовательность утверждений $P_1, \ldots, P_n$. Некоторые P_i определяют условия на ранее идентифицированные объекты. Так как все эти объекты относятся к текущим структурам данных решателя, истинность условий проверяется непосредственно и не требует какого-либо логического вывода. Для проверки этой истинности в интерпретаторе языка имеются специальные подпрограммы.

Другие P_i вводят в рассмотрение новые объекты, связанные заданным образом с уже введенными до этого объектами. Если такие объекты определяются неоднозначно, то при поиске ситуации необходимо перечислять все возможные варианты. При программировании на обычном алгоритмическом языке процедурного типа

здесь нужно было бы использовать операторы цикла, причем вложенность циклов была бы равна числу неоднозначно определенных переходов к «новым» объектам в цепочке P_1, …, P_n. Чтобы избежать таких громоздких конструкций, в языке выделен ряд специальных отношений между объектами, для которых перечисление реализуется автоматически.

При обработке «перечисляющего» утверждения P_i сначала рассматривается первая версия выбора новых объектов и предпринимается попытка для выбранных объектов продвинуться вправо по цепочке P_1, …, P_n. Если в некоторый момент дальнейшее продвижение, ввиду ложности условия, становится невозможным, то происходит откат к последнему «перечисляющему» утверждению и определение очередной версии выбора объектов. Этот принцип «перечисления по умолчанию» позволяет в качестве исполняемого текста программы на языке ЛОС использовать само логическое описание искомой ситуации и таким образом существенно повысить степень «читабельности» программы.

Заметим, что из совсем других соображений принцип реализации операторов с перечислением значений их выходных переменных возник в известном алгоритмическом языке ПРОЛОГ. В отличие от языка ЛОС, ПРОЛОГ изначально ориентирован на работу с объектами предметного, а не структурного уровня. Поэтому центральное место в языке ПРОЛОГ занимает логический вывод, основанный на процедуре унификации. Этот логический вывод становится совершенно излишним при работе с элементами структуры данных, допускающими непосредственную проверку условий, и какого-либо аналога его в языке ЛОС не предусмотрено. В то же время в языке ПРОЛОГ отсутствует целый ряд важных возможностей, предусмотренных в языке ЛОС для эффективной работы со сложными логическими описаниями образов «структурного» уровня и для использования режима сканирования задачи. Язык ЛОС ориентирован на создание автоматики, управляющей логическими процессами, в то время как при разработке языка ПРОЛОГ эта автоматика осталась почти «за кадром».

Впрочем, оба языка — и ЛОС, и ПРОЛОГ — относятся примерно к одному уровню и обладают настолько существенными недостатками, что работу над созданием языка для записи приемов пришлось продолжить уже после того, как на языке ЛОС был создан весьма эффективный решатель задач по элементарной алгебре. Как правило, значительную часть программы приема на языке ЛОС составляет описание вида тех утверждений или выражений, которые могут быть идентифицированы с фрагментами заданной теоремы предметной области. На практике редко бывает так, чтобы идентифицируемые логические конструкции в точности совпадали с теми, которые имеются в теореме.

Например, при идентификации квадратного трехчлена $ax^2 + bx + c$ коэффициент a может оказаться равным единице, и тогда в задаче вместо произведения ax^2 будет записана степень; сама эта степень может не иметь в точности вида квадрата, а лишь иметь четный коэффициент показателя степени, и т. д. Чтобы, несмотря на эти различия, идентифицировать теорему, приходится фактически в программе приема описывать не ее вид, а вид некоторого класса теорем, получающихся из нее различными вариациями. Текст программы оказывается, во-первых, весьма громоздким и, во-вторых, достаточно удаленным от исходной записи теоремы. Еще большую громоздкость этому тексту придают различные вставки, связанные с решающими правилами приема.

Поэтому, несмотря на все предоставляемые языком ЛОС и интерфейсом его редактора возможности для быстрого чтения и понимания логических описаний ситуаций, все же в сколь-нибудь невырожденных случаях описания приемов оказываются трудночитаемыми. Заметим, что хотя язык ПРОЛОГ, казалось бы, и ориентирован на запись программы в виде совокупности теорем предметной области, однако он тоже не позволяет преодолеть указанной трудности. Если теоремы идентифицируются без учета возможных вариаций, то программа оказывается заведомо слабой; если начинается их учет, то и на языке ПРОЛОГ для задания общего вида некоторого класса теорем придется от предметного уровня перейти к структурному (к которому язык ЛОС более приспособлен) и, таким образом, полностью утратить наглядность исходного текста теоремы.

Чтобы восстановить утерянную при погоне за эффективностью работы приема наглядность и компактность записи, был создан новый язык, уровень которого существенно выше, чем у языков ЛОС или ПРОЛОГ. Прием на этом языке задается как теорема предметной области, снабженная некоторой «алгоритмизирующей» разметкой. На основе разметки компилятор осуществляет необходимое обобщение вида теоремы, формулирует описание на языке ЛОС ситуаций, в которых она должна быть применена, и добавляет к нему операторы языка ЛОС, реализующие применение. Фактически этот язык имеет два независимых логических уровня: предметный уровень, на котором представлена теорема, и структурный уровень, на котором формулируются условия целесообразности применения теоремы. Оба эти уровня предполагают использование полноценного логического языка, однако логические записи структурного уровня, в отличие от записей уровня предметного, не участвуют в каком-либо логическом выводе — они используются только для проверки условий при принятии решения.

Роль «алгоритмизирующей» разметки приема весьма разнообразна — это и указание способа применения теоремы, и определение необходимых правил ее обобщения, и указание условий целе-

сообразности применения, и уточнение алгоритмических средств, используемых для обработки посылок теоремы, и указание на «технические» сообщения, передаваемые при срабатывании данного приема другим приемам, и указатели переключения внимания, и многое другое. Эта разметка представляет собой что-то вроде генотипа приема, элементы которого допускают независимое варьирование в достаточно широких пределах и по которому компилятор создает фактически реализуемую программу приема на языке ЛОС. Такая аналогия позволила назвать новый язык — ГЕНОЛО́Г (*генетический язык логического программирования*).

Описание приема на языке ГЕНОЛОГ оказалось значительно более компактным и понятным, чем на языке ЛОС. На экране изображается в стандартной математической записи теорема приема, под которой размещается в нескольких окнах сопровождающая теорему информация. Во многих случаях эта информация занимает всего несколько строчек, в то время как текст создаваемой компилятором программы на языке ЛОС составляет несколько полных экранов. Трудоемкость чтения приема и его коррекции по сравнению с языком ЛОС уменьшилась на порядок.

В отличие от обычных языков программирования, язык ГЕНОЛОГ не представляет собой сколь-нибудь завершенного явления. По существу, он является коллекцией типовых алгоритмических конструкций, используемых при переходе от теоремы к программе. Эта коллекция, достаточно богатая на текущий момент, продолжает (хотя все реже и реже) пополняться при рассмотрении новых предметных областей. Соответственно, при переходе к новой предметной области обычно приходится затрачивать некоторые усилия на развитие компилятора языка ГЕНОЛОГ. Как показывает опыт, эти затраты несопоставимо малы в сравнении с той последующей экономией, которую дает применение языка ГЕНОЛОГ при обучении решателя в данной области.

Главным доводом в пользу применения и развития языка ГЕНОЛОГ является, впрочем, даже не наглядность представления приема и проистекающие из нее преимущества для ручной «дрессировки» решателя на тысячах задач из различных разделов, а возможность приблизиться вплотную к тому источнику, из которого приемы возникают, — к логическому языку предметной области. Накопленное в решателе многообразие приемов можно рассматривать как задачник на извлечение приемов из теорем. Оно допускает некоторую предварительную классификацию приемов по типам, определяющим в общих чертах, какая именно цель достигается при использовании теоремы.

Ряд элементов описания приема на языке ГЕНОЛОГ удается автоматически синтезировать по теореме и типу приема. Это используется при ручном обучении решателя как средство пополнения тек-

ста приема различными стандартными элементами и существенно облегчает процесс ввода приема. Однако значительная часть элементов описания приема не может быть непосредственно извлечена из теоремы и общей установки на синтез приема. Эти элементы возникают для регулировки поведения приема в контексте всей базы приемов, и для их создания нужно развивать аппарат анализа взаимодействий между приемами, а также аппарат анализа решений, осуществляющий тестирование приема на серии задач.

3.1.6. Переход от теоремы к приему

При переходе от теоремы к приему прежде всего требуется определить класс возможных типов приемов, создание которых по данной теореме имеет смысл. Во многих случаях такая целевая характеризация теоремы может быть выполнена «из общих соображений» и реализована сравнительно несложной процедурой. Однако чаще всего сама теорема возникает для вполне определенной целевой нагрузки, и истоки приема расположены гораздо глубже — уже в самом процессе логического вывода, пополняющего запасы теорем.

Математическая логика не позволила сколь-нибудь существенно продвинуться в понимании динамики реальных математических теорий. Теория в ней обычно представляется как множество всех вообще утверждений, которые можно получить из аксиом с помощью правил вывода, или как множество всех утверждений, истинных в допустимых интерпретациях. За исключением вырожденных случаев, такое множество является бесконечным и раз навсегда заданным. В то же время реальная математическая теория не просто конечная — заносимые в нее истинные утверждения подвергаются тщательному отбору.

По-видимому, объяснение принципов этого отбора должно было бы основываться на целевой ориентации развития теории — т. е. на анализе того алгоритмического аппарата решения задач, который она предлагает. Попытки обучения решателей показывают, что этот аппарат чрезвычайно сложен и эвристичен по своей сути. По существу, его можно уподобить «дрессированному хаосу». Включение такого аппарата в рамки теоретических построений математической логики весьма проблематично, а замена его простыми суррогатами не решает проблемы. В то же время наличие эффективного решателя позволяет начать хотя бы экспериментальные исследования динамики теорий.

Новые утверждения в математической теории возникают в процессе логического вывода. Этот процесс основан на применении некоторых правил вывода к ранее полученным утверждениям либо к аксиомам. Простейший принцип отбора в состав теории получаемых таким образом утверждений состоит в удалении тех утверждений, применение которых при решении задач (после синтеза

их приемов) сводились бы к последовательному применению уже имеющихся приемов. Эту проверку «на избыточность» легко организовать, если использовать решатель для упрощения новых утверждений — сохраняться будут только те, которые после упрощения не выродились в логическую константу «истина».

Разумеется, целевая установка используемой здесь задачи на упрощение должна быть весьма аккуратно согласована с целевой установкой процесса поиска новых теорем. Такого рода проверка новых фактов на нетривиальность — в общем, обычное явление в практике математических исследований. Заметим, что последовательное применение правила вывода и решения задачи на упрощение приводит нас к более сложной процедуре вывода, использующей всю силу аппарата решателя, — к своего рода «приему вывода». В общем, применение процедуры упрощения на выходе обычных правил вывода уже позволяет существенно уменьшить поток получаемых следствий, сохраняя только наиболее ценные из них. Следует заметить, что эксперименты с процессами логического вывода, использующими некоторую несложную процедуру упрощения (редуцирования), и попытки теоретического изучения этих процессов предпринимаются в математической логике уже достаточно давно (см. работы по *rewriting theory*). Однако логический вывод с упрощением не устраняет экспоненциального роста числа получаемых следствий с ростом глубины вывода и годится лишь для получения серии первых, простейших следствий из исходных аксиом теории.

В нормальной практике математических исследований получению новых результатов обычно предшествует постановка задачи либо формулировка гипотезы. Лишь в простейших случаях новые утверждения получаются путем непосредственного вывода из ранее известных. Поэтому естественно предположить, что главным механизмом для развития теории служат не правила вывода, а более сложные процедуры, включающие в себя средства для постановки задач и выработки гипотез на основе анализа имеющихся теорем и использующие решатель для работы с поставленными ими задачами. Такие процедуры будем далее называть *приемами вывода теорем*.

Приведенная выше конструкция с последовательным применением правила вывода и задачи на упрощение является лишь частным примером такого рода приемов. Чтобы получать другие типы приемов вывода теорем, можно использовать в качестве обучающего материала различные уже известные утверждения из предметной области, предлагая возможно более простые объяснения логики «открытия» этих утверждений путем постановки вспомогательных задач и формулировки гипотез. Эта работа была проделана для некоторых простейших разделов (алгебра логики, алгебра множеств,

свойства арифметических операций), в результате чего возникло около сотни различных приемов вывода.

Даже для такой сравнительно сложной теоремы, как формула Кардано, удалось найти весьма простую и, по существу, общелогическую процедуру, извлекающую ее (разумеется, с помощью решателя) непосредственно из тождества для куба суммы. Продолжение этой работы могло бы привести к созданию значительной по своим размерам базе приемов вывода теорем, необходимой для процедур автоматического развития теории и синтеза приемов. Способность извлекать из исходных утверждений неожиданные и сильные следствия — за счет длинных цепочек логических вычислений, реализуемых решателем, — позволила бы взять такой базе приемов на себя роль «генератора идей».

3.2. Логический язык решателя задач

В системе используется логический язык открытого типа, пополняемый в процессе обучения. Для каждого нового понятия вводятся свои правила образования с его помощью корректных синтаксических конструкций и свои соглашения об их смысловой интерпретации, учитываемые при создании приемов, использующих данное понятие.

Алфавит языка состоит из элементов двух типов — логических символов и символов переменных. *Логические символы* обозначают конкретные понятия (отношения между объектами, операции над ними, имена объектов, логические связки, кванторы и т. д.); *символы переменных* служат для обозначения варьируемых объектов. Символы каждого типа пронумерованы последовательными натуральными числами. В компьютерной реализации для этих номеров зарезервированы диапазоны от 1 до $2^{16} - 1$. Логические символы по существу отождествлены со своими номерами, и в этом смысле все они изначально имеются в алфавите логического языка. Однако используемыми на текущий момент считаются только те из них, для которых введено название — слово либо словосочетание, имеющее не более 24 букв либо цифр. Такие названия хранятся в специальном файле и легко могут быть изменены. Внутренние алгоритмы системы этих названий не используют — они нужны только для отображения логических символов на экране в различных диалогах.

В языке рассматриваются конструкции двух типов: утверждения и выражения. *Выражения* используются для обозначения объектов; *утверждения* — для формулировки свойств объектов и отношений между ними. Как выражения, так и утверждения представляют собой записи со скобками (термы), определяемые следующим индуктивным образом:

1) каждое однобуквенное слово, состоящее из логического символа либо символа переменной, является термом;

2) если $t_1, \ldots, t_n$ — термы; $n \geq 1$; f — логический символ, то слово $f(t_1 \ldots t_n)$ есть терм.

Помимо этого общего правила образования новых термов с каждым используемым логическим символом f будут связываться свои дополнительные ограничения на элементы $t_1, \ldots, t_n$, при которых новый терм представляет собой осмысленное утверждение или выражение.

Далее мы в качестве примеров приведем логические символы языка решателя задач в таких разделах математики, как алгебра множеств, элементарная алгебра и элементарная геометрия. Описание логических символов остальных разделов математики и физики, таких как математический анализ, аналитическая геометрия, линейная алгебра, комплексный анализ, теория вероятностей, элементарная физика, можно найти в работе [8, гл. 2].

3.2.1. Общелогические понятия

Начнем с перечисления наиболее общих логических символов языка, сохраняя для них те же названия, которые используются в системе. Все эти названия будем выделять в тексте кавычками.

Прежде всего выделим логические константы «истина», «ложь», а также логические связки «и», «или», «не», «эквивалентно», позволяющие строить новые утверждения «и($A_1 \ldots A_n$)»; «или($A_1 \ldots A_n$)»; «не(A_1)»; «эквивалентно($A_1 A_2$)» из ранее построенных утверждений $A_1, \ldots, A_n$.

Логическая связка «если-то» используется только в сочетании с квантором общности: «длялюбого($x_1 \ldots x_n$ если $A_1 \ldots A_m$ то A_0)», где $x_1, \ldots, x_n$ — попарно различные переменные; $A_0, A_1, \ldots, A_m$ — некоторые утверждения. Это обусловлено тем, что в бесквантворной ситуации предпочтительнее использовать связки «или», «и», «не», к которым она легко сводится. В случае $m = 0$ запись приобретает вид «длялюбого($x_1 \ldots x_n A_0$)». Аналогичным образом записывается утверждение существования значений $x_1, \ldots, x_n$, при которых истинно A_0: «существует($x_1 \ldots x_n A_0$)».

Чтобы обеспечить возможность компактной записи утверждения о существовании и единственности этих значений, введена модификация квантора существования «Существует($x_1 \ldots x_n A_0$)» (логический символ «Существует» — с большой буквы). Символы «длялюбого», «существует» и «Существует» — связывающие. Переменные $x_1, \ldots, x_n$ образуют в приведенных выше утверждениях связывающую приставку.

Простейшими выражениями языка являются однобуквенные слова, образованные переменными либо логическими символами. Во втором случае считается, что выражение имеет своим значением

тот символ, из которого оно состоит. Такое соглашение упрощает использование логического языка для описания вида его собственных конструкций. Хотя из него и вытекает, что константы «истина» и «ложь» одновременно должны считаться и утверждениями, и выражениями, особых неудобств в дальнейшем это не создает.

Для задания значения путем разбора случаев используется условное выражение «вариант($A\ t_1\ t_2$)». Если утверждение A является истинным, то значение этого выражения равно значению выражения t_1, иначе оно равно значению выражения t_2. Аналогичным образом рассматриваются условные утверждения «альтернатива($A\ B_1\ B_2$)». Если утверждение A истинно, то истинность условного утверждения совпадает с истинностью утверждения B_1, иначе — с истинностью утверждения B_2.

Условные утверждения оказались практически неиспользуемыми в формулировках и решениях задач, однако они весьма часто применяются в логических описаниях структурного уровня — внутри программ приемов решателя.

Используются всего две конструкции, вводящие связанные переменные при построении новых выражений. Первая — «класс($x_1 \ldots x_n\ A$)» — определяет класс всех наборов ($x_1 \ldots x_n$), на которых утверждение A является истинным; вторая — «отображение($x_1 \ldots x_n\ A\ t$)» — задает функцию, определенную на множестве всех таких наборов ($x_1 \ldots x_n$), для которых утверждение A является истинным, и принимающую на этом множестве значения, определяемые выражением t.

Для задания интеграла, производной, предела, суммы по множеству значений параметра и т. п. используются обычные операции над функциями, не вносящие сами по себе каких-либо связанных переменных.

К списку используемых в языке общелогических конструкций добавим также равенство «равно($t_1\ t_2$)» и выражение «значение($f\ t$)», определяющее значение функции f в точке t (в обычной записи — $f(t)$).

3.2.2. Алгебра множеств

Утверждения «множество(A)»; «принадлежит($t\ A$)»; «содержится (A, B)» означают соответственно, что A есть множество; t — элемент этого множества; множество A является подмножеством множества B. Для условия непересечения двух множеств A, B введено обозначение «непересек(A, B)».

Пустое множество обозначается посредством логического символа «пусто». Выражения «объединение($A_1 \ldots A_n$)», «пересечение($A_1 \ldots A_n$)», «разность(A, B)», «прямоепроизведение($A_1 \ldots A_n$)» используются для обозначения простейших операций над множествами.

Иногда приходится ссылаться на некоторый произвольный элемент непустого множества, не конкретизируя этот элемент. Для обозначения такого элемента введено выражение «элемент(A)». Аналогично выражение «внешнийэлемент(A)» обозначает некоторый абстрактный элемент, не принадлежащий множеству A.

Для задания конечной последовательности элементов A_1, …, A_n используется выражение «набор(A_1…A_n)». Всюду далее, говоря о наборах, мы отождествляем это понятие с понятием конечной последовательности. В тех случаях, когда приходится вводить операции либо отношения с переменным числом операндов (за исключением двуместных ассоциативных операций), обычно они представляются как одноместные операции либо отношение над набором. В частности, конечное множество, состоящее из элементов A_1, …, A_n, обозначается посредством выражения «перечень(набор(A_1…A_n))». Для пустого набора (последовательности длины 0) введено обозначение «пустоеслово».

Для результата последовательной записи наборов A_1, …, A_n используется выражение «конкатенация(A_1…A_n)». Результаты добавления к началу либо концу набора A элемента t обозначаются, соответственно, «префикс(t, A)» и «суффикс(A, t)». Чтобы выделить из набора (A_1…A_n) поднабор (A_i…A_j), будем использовать обозначение «поднабор (A, i, j)». Для обозначения результата вставки в указанный набор элемента B перед i-м элементом служит выражение «Вставка(A, i, B)».

Утверждение «семействомножеств(A)» означает, что A есть функция, принимающая в качестве своих значений множества. Выражения «объединениевсех(A)», «пересечениевсех(A)» обозначают объединение и пересечение множеств для всевозможных значений аргумента функции A, принадлежащих ее области определения.

Обычным образом используются выражение «мощность(A)», утверждение «конечное(A)» и константы «счетное», «континуум».

Перечислим обозначения, введенные для работы с числовыми множествами.

Числовой промежуток с концами a, b и указателями c, d отнесения этих концов к промежутку (0 — конец не включается в промежуток, 1 — включается) обозначается «промежуток(a, b, c, d)». В частности, таким образом могут задаваться и бесконечные промежутки; для обозначения их концов (и в других аналогичных случаях) используются логические символы «минусбеск» и «плюсбеск» (напомним, что в данном разделе описывается только внутреннее логическое представление, используемое решателем; при отображении на экране процесса решения задач числовые промежутки изображаются традиционным образом, с использованием круглой скобки для указания невключения конца в промежуток и квадратной — если конец относится к промежутку).

Множество вещественных чисел во внутреннем представлении обозначается посредством конструкции «промежуток(минусбеск, плюсбеск, 0, 0)», хотя на экране оно прорисовывается как *R*. Утверждение «числовойотрезок(*A*)» означает, что *A* есть отрезок на числовой прямой (включая концы).

Множества целых, целых неотрицательных, натуральных и рациональных чисел обозначаются, соответственно, «целые», «целыенеотрицательные», «натуральные» и «рациональные». Множество целых чисел, состоящее из элементов i, $i + 1$, …, j, обозначается «номера(i, j)». Набор тех же самых элементов обозначается «наборномеров(i, j)». Множество всех членов арифметической прогрессии с первым элементом a и знаменателем p обозначается «арифмпрогрессия(a, p)».

Утверждения «нижняягрань(x, A)», «Нижняягрань(x, A)», «верхняягрань(x, A)», «Верхняягрань(x, A)» означают, соответственно, что x есть нижняя либо верхняя грань числового множества A, причем заглавная буква указывает на строгую верхнюю грань, а ее отсутствие — на нестрогую. Утверждения «огрснизу(*A*)» и «огрсверху(*A*)» означают, что множество ограничено снизу либо, соответственно, сверху. Точная нижняя и точная верхняя грани обозначаются «инф(*A*)» и «суп(*A*)». Утверждения «наибольший(x *A*)» и «наименьший(x *A*)» указывают наибольший либо наименьший элемент x множества A.

Обычным образом используются выражения «граница(*A*)», «внутренность(*A*)», «замыкание(*A*)».

Для работы с функциями используются обозначения «функция(f)» (утверждение о том, что f есть функция); «область(f)» (область определения функции f); «значения(f)» (множество значений функции f); «Отображение (f, A, B)» (утверждение о том, что f есть функция, определенная на множестве A и принимающая значения в множестве B); «взаимнооднозначно(f)» (утверждение о том, что функция в различных точках своей области определения принимает различные значения).

Образ множества M для функции f обозначается «образ(f, M)»; прообраз множества M — «прообраз(f, M)»; прообраз элемента t — «слой(f, t)».

Для задания конкретных функций используются следующие выражения:

- «конст(M, b)» обозначает константную функцию, принимающую во всех точках своей области определения M значение b;
- «См(a, b)» обозначает функцию, определенную на одноэлементном множестве $\{a\}$ и принимающую на нем значение b;
- «тождфунк(a)» обозначает тождественную функцию, определенную на множестве a;
- «таблица(f_1…f_n)» обозначает результат «объединения» функций f_1, …, f_n, принимающих на пересечении своих областей определения одинаковые значения;

- «доопределение(f, M, b)» обозначает результат доопределения функции f в тех точках множества M, где она еще не определена, значением b;
- «сужение(f, M)» обозначает сужение функции f на множество M;
- «перестановка(f, A)» означает, что функция f представляет собой взаимно однозначное отображение начального отрезка натурального ряда на конечное множество A.

Число переменных многоместной функции f обозначается «числопеременных(f)»; функция, получающаяся из f подстановкой констант набора b вместо переменных, номера которых (начиная с единицы) перечислены в наборе a, обозначается «подфункция(f, a, b)». Утверждение «существперем(i, f)» означает, что i-я переменная функции f является существенной.

3.2.3. Элементарная алгебра

Все рассматриваемые в этом разделе операции и отношения являются вещественнозначными; для работы с комплексными числами предусмотрены альтернативные обозначения.

Для представления десятичного числа $a_1 \ldots a_n$ (a_1, …, a_n — цифры) в виде терма используется запись «величина($a_1 \ldots a_n$)». Если число дробное, то одно из a_i в этой записи является символом запятой (запятая введена в число логических символов только для данной цели; при изображении на экране термов в стандартной математической записи вместо запятой используется точка). Заметим, что в случае $n = 1$ символ «величина» не используется, а число представляется цифрой. Для простейших операций применяются обозначения «плюс($a_1 \ldots a_n$)» (здесь и далее $n \geq 2$); «минус(a)»; «умножение($a_1 \ldots a_n$)»; «дробь(a_1, a_2)»; «степень(a_1, a_2)»; «максимум($a_1 \ldots a_n$)»; «минимум($a_1 \ldots a_n$)». Операция вычитания не введена, так как все равно при решении задач ее приходится выражать через «плюс» и «минус», а прорисовка вычитания в стандартной математической записи неотличима от выражения с «плюсом» и «минусом».

Сумма и произведение конечного семейства значений обозначаются «суммавсех(F)» и «произведениевсех(F)». Здесь F — функция, определенная на некотором конечном множестве и принимающая в качестве своих значений суммируемые либо перемножаемые величины. При прорисовке в стандартной математической записи эти выражения изображаются обычным образом, с помощью знаков $\sum$ и $\prod$.

Неравенства записываются в виде «меньше(a, b)»; «больше(a, b)»; «меньшеилиравно(a, b); «большеилиравно(a, b)». Утверждение «число(a)» означает, что a есть вещественное число. Логические символы «е» и «пи» обозначают соответствующие числовые константы.

Логарифм числа b по основанию a обозначается «логарифм(a, b)». Для прочих элементарных функций используются обозначения: «синус(a)»; «косинус(a)»; «тангенс(a)»; «котангенс(a)»; «секанс(a)»; «косеканс(a)»; «арксинус(a)»; «арккосинус(a)»; «арктангенс(a)»; «арккотангенс(a)»; «модуль(a)». Выражение «сигнум(a)» считается равным минус единице для отрицательных чисел, не определенным в нуле и равным единице для положительных чисел. «Сигнум(a)» доопределяется в нуле единицей и равен нулю для отрицательных чисел. Для гиперболических функций используются обозначения «гипсинус(a)»; «гипкосинус(a)»; «гиптангенс(a)»; «гипкотангенс(a)».

Утверждения «целое(a)», «натуральное(a)», «четное(a)», «рациональное(a)», «простое(a)» уточняют тип вещественного числа a. Выражения «числитель (a)» и «знаменатель(a)» определяют, соответственно, целочисленный числитель и натуральный знаменатель рационального числа a (после сокращения соответствующей дроби). Утверждение «делит(m, n)» означает, что целое число m делит целое число n. Выражения «нод($n_1 \ldots n_k$)» и «нок($n_1 \ldots n_k$)» означают, соответственно, наибольший общий делитель и наименьшее общее кратное целых чисел n_1, …, n_k. Утверждение «взаимнопросты(m n)» означает взаимную простоту целых чисел m и n. Выражение «целаячасть(a)» обозначает целую часть числа a. Для обозначения вычета целого числа m по натуральному модулю n служит выражение «вычет(m, n)». Выражение «числосочетаний(m, n)» обозначает число сочетаний из m по n; выражение «факториал(n)» — факториал целого неотрицательного числа n.

Формальные многочлены требуют в логическом языке специального представления — они, разумеется, не являются функциями, но не являются также и формулами. Последние суть слова в конечном алфавите, и множество их всего лишь счетно, в то время как множество формальных многочленов над полем вещественных чисел континуально. Поэтому многочлены рассматриваются как абстрактные объекты, задаваемые тройкой (X, F, K). Здесь $X = (x_1, \ldots, x_n)$ — упорядоченный набор символов переменных; F — поле; K — функция, определенная на некотором конечном подмножестве M множества n-ок целых неотрицательных чисел и принимающая значения из поля F. Эта функция сопоставляет набору степеней переменных x_1, …, x_n коэффициент соответствующего одночлена, входящего в многочлен. Для обозначения многочлена используется выражение «многочлен(X, F, K)». Разумеется, для отображения формального многочлена на экране предусмотрено более удобное обозначение: $\mu_{x_1 \ldots x_n}(S)$, где S — обычным образом записанная сумма одночленов. Переход от одного представления многочлена к другому выполняется автоматически. Утверждения «Многочлен(A)» и «веществмногочлен(A)» используются для указания на то, что A

есть многочлен (в первом случае — общего вида, во втором — над полем вещественных чисел).

3.2.4. Элементарная геометрия

Почти все вводимые в элементарной геометрии понятия относятся одновременно и к двумерному, и к трехмерному случаю. Использование в задаче понятия, имеющего смысл только для двумерного случая, автоматически означает, что эта задача целиком планиметрическая. Чтобы ускорить работу решателя при рассмотрении планиметрических задач, в список посылок таких задач обычно вводится (как правило, автоматически самим решателем) техническая пометка «планиметрия». Эту пометку можно рассматривать как вспомогательное фиктивное утверждение, указывающее на существование некоторой общей плоскости, к которой относятся все рассматриваемые в задаче точки.

Утверждение «точка(*A*)» означает, что *A* есть точка трехмерного пространства (в планиметрическом случае эта точка относится к некоторой не уточняемой плоскости данного пространства). Выражение «прямая(*AB*)» обозначает прямую, проходящую через точки *A*, *B*. В случае совпадения этих точек условимся считать, что данное выражение обозначает какую-то произвольную прямую, проходящую через рассматриваемую точку. Аналогично выражение «плоскость(*ABC*)» обозначает плоскость, проходящую через точки *A*, *B*, *C*. Обучение решателя выполнялось таким образом, что он может работать только с прямыми и плоскостями, заданными явным образом через пару (тройку) точек с помощью указанных выражений. Поэтому в условиях задач не рекомендуется обозначать прямые и плоскости вспомогательными переменными. Тем не менее в языке предусмотрены используемые в некоторых специальных случаях утверждения «Прямая(*A*)» и «Плоскость(*A*)», указывающие, что *A* является, соответственно, прямой либо плоскостью.

Выражения «отрезок(*AB*)» и «интервал(*AB*)» обозначают, соответственно, отрезок и интервал с концами в точках *A*, *B*. Выражения «луч(*AB*)» и «обратныйлуч(*AB*)» обозначают, соответственно, луч с началом в точке *A*, проходящий через точку *B*, и луч, обратный данному лучу.

Расстояние между точками *A* и *B* обозначается «расстояние(*AB*)». Расстояние от точки *A* до прямой либо плоскости *B* обозначается, соответственно, «расстдопрямой(*A*, *B*)» и «расстдоплоскости(*A*, *B*)». Расстояние между двумя замкнутыми множествами точек *A* и *B* обозначается «расстмежду(*A*, *B*)».

Величина угла с вершиной в точке *B*, стороны которого проходят через точки *A* и *C*, обозначается «угол(*ABC*)». Эта величина измеряется от 0 до π. Угол как фигура в этом случае обозначается «Угол(*ABC*)». Утверждение о том, что луч *AD* является биссектрисой

угла *ABC*, записывается в виде «биссектриса(*ABCD*)». Прямая, на которой лежит данная биссектриса, обозначается «Биссектриса(*ABC*)».

Утверждения «параллельны(*A*, *B*)» и «перпендикулярно(*A*, *B*)» используются для указания на параллельность либо перпендикулярность прямых либо плоскостей *A*, *B* (допускаются любые сочетания: прямая — прямая; прямая — плоскость; плоскость — плоскость).

Для указания на то, что две точки *A* и *B* лежат по одну сторону от прямой либо плоскости *C*, используется утверждение «однасторона(*A*, *B*, *C*)». Утверждение «разныестороны(*A*, *B*, *C*)» указывает, что они лежат по разные стороны. Если *C* — прямая, то в обоих случаях утверждение указывает также и на принадлежность точек *A*, *B* общей плоскости с этой прямой. Если одна из точек попадает на прямую (плоскость), то считается, что точки одновременно лежат и по одну, и по разные стороны от прямой (плоскости). Утверждение «симметричны(*A*, *B*, *C*)» означает, что точки *A* и *B* расположены симметрично относительно точки, прямой либо плоскости *C*. Утверждение «осьсимметрии(*A*, *B*)» означает, что прямая *A* является осью симметрии плоской фигуры либо тела *B*. Утверждение «плоскостьсимметрии(*A*, *B*)» означает, что плоскость *A* является плоскостью симметрии тела *B*.

Выражение «полоса(*A*, *B*)» обозначает полосу между двумя параллельными прямыми *A* и *B*. Выражения «полуплоскость(*A*, *B*)» и «обрполуплоскость(*A*, *B*)» обозначают полуплоскости, лежащие по одну сторону от прямой *A*; первая из них содержит точку *B*, а вторая — не содержит этой точки (хотя плоскость этой полуплоскости содержит точку *B*).

Утверждение «треугольник(*ABC*)» означает, что точки *A*, *B*, *C* суть вершины треугольника (различны и не лежат на одной прямой). Аналогично утверждения «параллелограмм(*ABCD*)», «ромб(*ABCD*)», «прямоугольник(*ABCD*)», «квадрат(*ABCD*)», «трапеция(*ABCD*)» означают, что четверка точек *A*, *B*, *C*, *D* составляет набор вершин четырехугольника соответствующего типа. В первых четырех случаях допускаются произвольные циклические перестановки; в последнем случае вершины *A*, *D* должны лежать на большем (нижнем) основании трапеции. Заметим, что в решателе предусмотрены два обозначения для трапеции: указанное выше предполагает, что оба угла при нижнем основании не больше 90 градусов; в обозначении «Трапеция(*ABCD*)» это предположение отсутствует.

Утверждение «четырехугольник(*ABCD*)» означает, что четверка точек *A*, *B*, *C*, *D* образует вершины выпуклого четырехугольника. Если *a* — набор точек, то утверждения «многоугольник(*a*)»; «правмногоугольник(*a*)» означают, что данные точки образуют последовательно проходимые вершины некоторого многоугольника (во втором случае — правильного). Выражение «фигура(a)» обозначает множество всех точек многоугольника, ограниченного ло-

маной, проходящей через точки набора *a*. Утверждение «центр(*P*, *a*)» означает, что точка *P* является центром области *a*.

Утверждения «Медиана(*ABCD*)» и «Высота(*ABCD*)» означают, что точка *D* является основанием, соответственно, медианы либо высоты треугольника *ABC*, проведенной из вершины *A*. Утверждение «Биссектреуг(*ABCD*)» означает, что точка *D* является основанием биссектрисы этого же треугольника, проведенной из вершины *B*.

Выражения «площадь(*a*)» и «периметр(*a*)» обозначают площадь и периметр области *a* (во втором случае — только имеющей вид многоугольника). Выражение «длина(*a*)» обозначает длину линии *a*.

В двумерном случае окружность с центром в точке *A* обозначается либо как «окружность(*AB*)», где *B* — некоторая точка на этой окружности, либо как «окр(*Ar*)», где *r* — радиус. Аналогичные обозначения «круг(*AB*)» и «круградиуса(*Ar*)» используются для круга. В трехмерном случае задание окружности либо круга дополняется указанием третьей точки, лежащей в ее (его) плоскости. Здесь используются обозначения «Окружность(*ABC*)» и «Круг(*ABC*)»; *C* — дополнительная точка, фиксирующая плоскость.

Дуга окружности с центром *A*, имеющая концевые точки *B*, *C*, обозначается либо «дуга(*ABC*)» (меньшая из двух дуг), либо «большаядуга(*ABC*)» (большая из двух дуг). В случае диаметрально противоположных точек *B*, *C* большая и меньшая (условно) дуги выбираются некоторым неуточняемым образом, причем они различны. Выражение «дугаугла(*ABC*)» обозначает дугу, на которую опирается вписанный угол *ABC*.

Чтобы задавать область со сложной границей, образованной множеством фрагментов стандартного вида (отрезки прямых, дуги окружностей и т. п.), используется запись «Фигура(перечень(набор(A_1…A_n)))», где A_1, …, A_n — обозначения данных фрагментов.

Выражения «сектор(*ABC*)» и «сегмент(*ABC*)» обозначают, соответственно, сектор и сегмент круга, имеющего центр в точке *A* и определяемые крайними точками *B*, *C*, лежащими на окружности. Выбирается меньшая из дуг между данными точками (в случае равенства дуг берется неуточняемым образом одна из них). Выражение «кольцо(*ABC*)» обозначает кольцо с центром в точке *A*, внутренняя граница которого проходит через точку *B*, а внешняя — через точку *C*.

Утверждение «касательная(*A*, *B*)» означает, что прямая *A* является касательной к окружности *B*. Утверждения «внешкасательная(*A*, *B*, *C*)» и «внутркасательная(*A*, *B*, *C*)» означают, что прямая *A* является, соответственно, внешней либо внутренней общей касательной окружностей *B* и *C*. Утверждения «внешкасаются(*A*, *B*)» и «внутркасаются(*A*, *B*)» означают, что окружности *A* и *B* касаются друг друга, соответственно, внешним либо внутренним образом.

Утверждение «вписана(*A*, *B*)» означает, что окружность *A* вписана в многоугольник *B* (последний обычно задается выражением вида «фигура(...)»). Аналогично утверждение «описана(*A*, *B*)» означает, что окружность *A* описана около многоугольника *B*.

Утверждение «подобны(*A*, *B*)» означает, что многоугольники, наборы вершин которых суть *A* и *B*, подобны. Заметим, что здесь вместо обозначающего многоугольник выражения «фигура(*A*)» используется выражение *A*, определяющее набор его вершин. Утверждение «Подобны(*A*, *B*)» дополнительно фиксирует соответствие между вершинами, при котором имеет место подобие: вершине A_i соответствует вершина B_i.

Утверждение «выпукло(*A*)» указывает на выпуклость множества *A*.

В дополнение к указанным выше введен ряд специальных планиметрических обозначений для операций, позволяющих при решении задач на построение выражать новые элементы чертежа через уже известные. Выражение «тчк(*ABr*)» обозначает точку, отложенную на ориентированной прямой *AB* от точки *A* на величину *r* (при неотрицательном значении — на луче *AB*, иначе — на продолжении этого луча). Выражение «доляотрезка(*ABa*)» обозначает точку *C* ориентированной прямой *AB*, для которой отношение ориентированных длин отрезков *AC* и *AB* равно *a*.

Выражение «перпендикуляр(*AB*)» обозначает прямую, проведенную через точку *B* перпендикулярно прямой *A*, а выражение «параллелпрямая(*AB*)» — прямую, проведенную через указанную точку параллельно прямой *A*. Выражение «параллель(*AbB*)» обозначает прямую, параллельную прямой *A*, находящуюся от нее на расстоянии *b* и расположенную в той же полуплоскости, что и точка *B*. Наконец, выражение «поворотпрямой(*ABCa*)» обозначает прямую, получаемую при повороте луча *AB* на угол *a* в направлении полуплоскости, содержащей точку *C*.

3.3. Представление задач в решателе

3.3.1. Структуры данных, используемые для представления задачи

Как уже говорилось ранее, в решателе рассматриваются задачи четырех типов — на доказательство, на описание, на преобразование и на исследование. Начнем с описания используемых для их представления структур данных. Задача *Z* любого типа представлена набором $(p_1, \ldots, p_n)$, первый элемент p_1 которого есть логический символ — название типа задачи. Эти названия суть логические символы «доказать», «описать», «преобразовать» и «исследовать».

Элемент p_2 есть набор $(f_1, \ldots, f_m)$ утверждений, называемых посылками задачи. Посылки перечисляют априори известные ограни-

чения на фигурирующие в задаче объекты — то, что считается в задаче «данным». Список посылок обязательно непуст; если никаких исходных допущений не делается, то он предполагается состоящим из логической константы «истина».

Элемент p_3 есть набор $(a_1, \ldots, a_m)$ целых неотрицательных чисел, называемых весами посылок (a_i — вес посылки f_i) и используемых для переключения внимания при поиске очередного приема; в исходной ситуации веса посылок равны нулю.

Элемент p_4 — набор $(K_1, \ldots, K_m, K_0)$, элементы K_i которого при $i \neq 0$ суть наборы комментариев к i-й посылке, а K_0 есть набор комментариев ко всему списку посылок. Комментарии суть технические пометки, делаемые решателем в процессе работы для сохранения различной информации, направляющей ход решения. В исходной ситуации все K_i пусты.

Компоненты p_1 — p_4 являются общими для задач всех перечисленных выше типов. Прочие компоненты определяются в зависимости от типа задачи следующим образом.

1. **Задачи на доказательство.** В этом случае $n = 7$. Элемент p_5 представляет собой утверждение, называемое условием задачи. Его истинность и требуется доказать, предполагая истинность посылок. Элемент p_6 — целое неотрицательное число, называемое весом условия задачи; как и веса посылок, оно используется для переключения внимания в процессе решения задачи. Элемент p_7 — последний элемент задачи на доказательство — набор комментариев к задаче в целом. Они, как и комментарии к посылкам, представляют собой технические пометки, вводимые решателем в процессе работы. Решая задачу на доказательство, решатель может выдать в качестве ответа либо логический символ «истина» (если он убедится, что условие действительно является логическим следствием посылок), либо логический символ «отказ» (если его попытки останутся безуспешными).

Заметим, что решатель при рассмотрении задачи на доказательство не выдает сообщений о ложности условия либо о том, что оно не является следствием посылок (хотя для ускоренной выдачи отказа он и может попытаться установить эти обстоятельства). Задача на доказательство служит только для «односторонней» попытки установления истинности утверждения — именно в такого рода вспомогательных попытках обычно и возникает надобность при решении других задач. Если же нужно провести «двустороннюю» проверку — выяснить, является ли некоторое утверждение истинным либо ложным, то для этого используется задача на описание с пустым списком неизвестных (подробнее об этом см. ниже).

2. **Задачи на описание.** Здесь $n = 9$. В виде задачи на описание оформляются различные задачи, у которых требуется преобразовать заданным образом некоторый список утверждений — условий зада-

чи. Преобразования этого списка условий осуществляются в предположении истинности списка посылок. Как правило, условия содержат неизвестные, и в задаче требуется получить явное описание (полное либо неполное — в зависимости от целевой установки задачи) допустимых значений таких неизвестных. Элемент p_5 задачи на описание представляет собой список $(g_1, \ldots, g_k)$ ее условий. Элемент p_6 — набор $(b_1, \ldots, b_k)$ весов условий. Элемент p_7 — набор $(Q_1, \ldots, Q_k, Q_0)$ списков комментариев. Здесь Q_i, $i = 1, \ldots, k$, представляет собой список комментариев к условию g_i; Q_0 — список комментариев ко всей задаче в целом.

Элементы p_5, p_6, p_7 совершенно аналогичны элементам p_2, p_3, p_4, характеризующим посылки задачи. Элемент p_8 представляет собой набор технических пометок, называемых целями задачи. Ответ, получаемый в процессе решения задачи на описание, постепенно складывается в списке ее условий, так что при успешном завершении преобразований в конце решения выдается группа соединенных логической связкой «и» заключительных условий. При неудаче в качестве ответа выдается логический символ «отказ».

Класс утверждений, являющихся ответами на задачу, определяется в зависимости от набора ее целей. При этом некоторые из целей могут не накладывать каких-либо явных ограничений на ответ, а лишь указывать те или иные установки на оптимизацию, влияющие на принятие решений в процессе преобразований. Фактически список целей задачи представляет собой установку на управление базой приемов решателя, осуществляющей рассмотрение задачи. За исключением особых случаев принимается ряд соглашений о связи ответа задачи на описание с ее исходными условиями и посылками. Обычно задача на описание имеет цель, указывающую множество ее неизвестных, причем вхождение этих неизвестных в посылки задачи не допускается. Условия задачи должны являться следствиями ответа и посылок задачи; при этом ответ можно рассматривать как частичное либо полное описание множества наборов значений неизвестных, удовлетворяющих условиям задачи.

Достаточно часто при решении задачи на описание может оказаться полезным и даже необходимым рассмотрение совместных следствий условий и посылок. Так как ответ задачи извлекается из списка ее условий, то регистрация таких вспомогательных следствий в списке условий нежелательна — она приведет к чрезмерному его увеличению и потребует сложной процедуры «расчистки» условий после определения значений неизвестных. Поэтому в структуру задачи на описание пришлось ввести специальный накопитель совместных следствий условий и посылок. Роль этого накопителя (элемент p_9) играет вспомогательная задача на исследование, называемая блоком анализа задачи на описание. Изначально блок анализа отсутствует и вводится решателем в процессе рассмо-

трения задачи. В исходной ситуации элемент p_9 может быть равен либо логическому символу «пустоеслово» (общий случай), либо, в особых случаях, логическому символу «0» — последнее означает запрет на ввод блока анализа при решении.

3. Задачи на преобразование. В этом случае $n = 8$. Элемент p_5 — выражение, называемое условием задачи. Это выражение нужно преобразовать к некоторому виду, определяемому целевой установкой задачи. Как правило, требуется, чтобы исходное и результирующее выражения были равны в предположении истинности списка посылок. Иногда это требование может нарушаться; самый простой пример такого рода — использование задачи на преобразование для получения асимптотической оценки, когда результирующее выражение лишь асимптотически равно исходному. Более того, из-за технических причин в некоторых случаях удобно применять вспомогательные задачи на преобразование, условиями которых являются не выражения, а утверждения. Элемент p_6 — целое неотрицательное число, называемое весом условия. Элемент p_7 — набор комментариев к задаче. Элемент p_8 — набор целей задачи. Некоторые из целей накладывают ограничения на вид ответа; другие — определяют установки на оптимизацию ответа. Ответом задачи на преобразование служит либо результирующее ее условие (если оно удовлетворяет определяемым целями ограничениям), либо логический символ «отказ».

4. Задачи на исследование. В этом случае $n = 5$. Элемент p_5 — набор целей задачи. Эти цели управляют процессом вывода представляющих интерес следствий из посылок задачи, упрощения посылок и удаления избыточных посылок. В результате таких действий исходная задача на исследование Z преобразуется в некоторую новую задачу Z', и при исчерпании средств, отведенных для решения задачи Z, в качестве ответа на нее выдается задача Z'. По существу, такая выдача ответа является фикцией — в действительности внешняя процедура, обратившаяся к решению задачи на исследование, обычно имеет непосредственную ссылку на нее и по этой ссылке способна получать всю необходимую информацию о произошедших изменениях.

В исходной ситуации веса посылок и условий задачи, предъявляемой решателю, равны нулю; блок анализа и комментарии к задаче (ее посылкам, условиям) отсутствуют. Вспомогательные задачи, вводимые решателем, могут иметь ненулевые исходные веса посылок и непустые списки комментариев. Это обеспечивает необходимую преемственность, сохраняет ранее накопленную информацию, которая может понадобиться при решении вспомогательной задачи. Изменение весов, формирование комментариев и блока анализа осуществляются системой в процессе решения задачи.

3.3.2. Целевые установки задач

Целевая установка задачи определяется ее списком целей. Она имеется у всех типов задач, за исключением задач на доказательство. С технической точки зрения целевая установка обеспечивает управление поведением многих тысяч независимо функционирующих приемов, решающих задачу. Так как каждый прием рассматривает в своих решающих правилах лишь сравнительно небольшой фрагмент целевой установки и достижение ответа происходит в результате интегрального действия многих различных приемов, то варьирование списка целей задачи предоставляет достаточно большие степени свободы для управления коллективным поведением приемов. Пополнение списка возможных типов целей задач осуществляется, как правило, в связи с рассмотрением конкретной предметной области и не приводит к какой-либо модификации приемов, относящихся к другим предметным областям.

Перечислим некоторые наиболее часто встречающиеся типы целей, а в следующем подразделе данной темы на большом количестве примеров проиллюстрируем способы формального представления задач в конкретных областях.

Заметим, что цель задачи обычно представляется либо логическим символом, либо набором $(\varphi, A_1, \ldots, A_m)$, где φ — логический символ, называемый заголовком цели; $A_1, \ldots, A_m$ — некоторые объекты. В последнем случае указанный набор будем записывать далее как «$\varphi A_1 \ldots A_m$».

Пусть сначала Z — задача на описание. Если Z имеет цель «неизвестные $x_1 \ldots x_k$», где $x_1, \ldots, x_k$ — попарно различные переменные, то $x_1, \ldots, x_k$ называются *неизвестными* задачи Z. Прочие свободные переменные посылок и условий называются *параметрами* задачи. Задача на описание может и не иметь неизвестных, причем в зависимости от прочих целей в этом случае необходимо либо установить эквивалентность условий одной из логических констант «истина», «ложь», либо осуществить определенную переформулировку условий. Истинность или ложность посылок задачи Z должна однозначно определяться при указании значений всех их свободных переменных, отличных от неизвестных (как правило, неизвестные задачи Z вообще не встречаются в посылках, хотя допускается фиктивное в указанном смысле появление их в качестве свободных переменных посылок).

Если Z имеет цель «параметры $y_1 \ldots y_k$», где $y_1, \ldots, y_k$ — некоторые из неизвестных задачи, то переменные $y_1, \ldots, y_k$ называются *несущественными неизвестными* задачи Z. Несущественные неизвестные задачи представляют собой те из неизвестных, для которых требуется установить лишь сам факт существования удовлетворяющих условиям их значений, не находя эти значения явным образом. Более точно, произвольный ответ на задачу Z представляет собой такое

утверждение f, что для любых значений свободных переменных посылок задачи Z, при которых эти посылки истинны, из истинности f вытекает существование таких значений не входящих в f несущественных неизвестных задачи, что все условия задачи Z истинны. Несущественные неизвестные задачи являются «исключаемыми» неизвестными и обычно не входят в ответ. Если задача имеет цель «прямойответ», то ответ f не может содержать вспомогательных переменных, вводимых для обозначения объектов, существование которых вытекает из истинности посылок задачи. В этом случае описание допустимых значений неизвестных осуществляется только в терминах объектов, упоминавшихся в посылках и условиях задачи.

Если задача Z имеет цель «пример», то ответ f представляет собой утверждение, построенное при помощи логических символов «и», «или» из утверждений вида «равно(x t)», где x — неизвестная задачи Z; t — выражение, не зависящее от неизвестных, а также из некоторых утверждений, не зависящих от неизвестных задачи Z. При преобразовании f к виду дизъюнктивной нормальной формы ни в одну из элементарных конъюнкций не входят два различных утверждения вида «равно(x t)», соответствующих одной и той же неизвестной x. Данная цель используется в тех случаях, когда требуется указать конкретный набор значений неизвестных задачи, при которых истинны ее условия, быть может, с дополнительными ограничениями на объекты, упоминаемые в посылках задачи. Если такие ограничения недопустимы, то вводится дополнительная цель «полный».

Более подробно, если задача Z имеет цели «полный», «пример», то при замене всех входящих в ответ f утверждений вида «равно(x t)», где x — неизвестная задачи, на логический символ «истина» должно получаться утверждение f', являющееся следствием посылок задачи. Если же задача Z, имеющая цель «полный», не имеет цели «пример», то из истинности посылок этой задачи вытекает эквивалентность истинности ответа f существованию значений несущественных неизвестных, не являющихся свободными переменными f, при которых истинны все условия задачи Z.

Если задача Z имеет цель «явное», то ее ответ f представляет собой утверждение, относящееся к классу так называемых явных описаний значений неизвестных этой задачи. Для определения класса явных описаний в каждой из рассматриваемых предметных областей выделяются семейства конечных множеств $\{f_1, \ldots, f_s\}$ утверждений, называемых *атомарными описаниями* переменной x; эта переменная является параметром каждого из утверждений $f_1, \ldots, f_s$. Так, атомарными описаниями переменной x считаются, например, множества $\{x = t_1\}$, $\{x \in t_1\}$, $\{t_1 < x, x < t_2\}$, $\{\exists k(k$ — целое $\& \; x = \pi k)\}$,

{x — целое} и т. п. Здесь t_1, t_2 — произвольные выражения, не имеющие параметра x.

Явными описаниями значений неизвестных задачи Z считаются утверждения, построенные при помощи логических символов «и», «или» из элементов атомарных описаний неизвестных этой задачи и не зависящих от неизвестных утверждений, причем такие, что в результате преобразования к виду дизъюнктивной нормальной формы каждая их элементарная конъюнкция, при подходящей группировке членов, приобретает вид «и($A_1(x_1)$…$A_s(x_s)$ B)», где для любого $i = 1, \ldots, s$ $A_i(x_i)$ — конъюнкция утверждений атомарного описания неизвестной x_i задачи Z; x_i не является параметром утверждений $A_{i+1}(x_{i+1}), \ldots, A_s(x_s)$; B — утверждение, не зависящее от неизвестных.

В некоторых случаях бывает полезной цель «и», при наличии которой ответ f на задачу Z имеет вид «и(равно(x_{j_1} t_1)…равно(x_{j_m} t_m) g_1…g_s)», где $m + s - 1 \geq 1$; $x_{j_1}, \ldots, x_{j_m}$ — различные неизвестные задачи Z; $t_1, \ldots, t_m, g_1, \ldots, g_s$ не зависят от неизвестных задачи. Указанный вид ответа определяет подстановку выражений $t_1, \ldots, t_m$ вместо переменных $x_{j_1}, \ldots, x_{j_m}$; такая подстановка может использоваться, например, при решении системы уравнений путем выражения части ее неизвестных $x_{j_1}, \ldots, x_{j_m}$ через остальные неизвестные.

Для ограничения зависимости ответа от использованных при формулировке задачи переменных применяется цель «известно y_1… y_k», где $y_1, \ldots, y_k$ — переменные, не являющиеся неизвестными задачи; $k \geq 0$. При наличии такой цели каждая свободная переменная ответа f представляет собой либо неизвестную задачи Z, либо некоторую переменную y_i, $i = 1, \ldots, k$. Переменные $y_1, \ldots, y_k$, выделенные целью указанного вида, называются *исходными данными* задачи.

В качестве примера цели, определяющей установку на оптимизацию, приведем цель «упростить», активизирующую попытки редактирования найденного ответа путем разрешения условий на параметры относительно этих параметров, упрощения входящих в ответ выражений и упрощения логической структуры ответа.

При формулировке исходной задачи может быть использована цель «одз», указывающая на необходимость решения задачи в области допустимых значений (ОДЗ) неизвестных и параметров. Эта цель, по сути дела, является лишь относящимся к интерфейсу системы техническим приемом, позволяющим в неявной форме задавать часть условий либо посылок задачи. Она инициирует расширение списков условий и посылок задачи рядом утверждений, характеризующих область допустимых значений переменных задачи, после чего удаляется.

Приведем несколько примеров часто встречающихся комбинаций целей задач на описание. Список {«неизвестные x_1…x_k», «полный», «пример»} встречается у задач Z, введенных при решении

задачи на доказательство существования значений переменных x_1, …, x_k, удовлетворяющих некоторому утверждению f. При этом в задаче Z требуется найти конкретные значения x_1, …, x_k, быть может, выразив их через вспомогательные обозначения для объектов, существование которых вытекает из посылок. Последнее объясняет отсутствие цели «прямойответ», блокирующей приемы, вводящие такие вспомогательные обозначения.

Список {«неизвестные $x_1 \ldots x_k$», «полный», «явное», «прямойответ»}, как правило, вместе с целью «упростить» встречается у задач, в которых требуется описать в явной форме все значения неизвестных, удовлетворяющие условиям (например, задачи на решение систем уравнений или неравенств; задачи на определение геометрического места точек в геометрии и т. п.). Если некоторые из условий такой задачи представляли собой утверждения о существовании некоторых объектов y_1, …, y_m, то при ее решении возможно появление вспомогательных задач, неизвестными которых, помимо x_1, …, x_k, становятся также y_1, …, y_m. В этом случае вспомогательная задача имеет наряду с перечисленными выше цель «параметры $y_1 \ldots y_m$».

Список {«неизвестные $x_1 \ldots x_k$», «полный», «явное», «прямойответ», «известно $y_1 \ldots y_k$»} возникает в тех случаях, когда значения неизвестных требуется выразить через исходные данные y_1, …, y_k, представляющие собой, вообще говоря, лишь часть параметров задачи. Как правило, в этом случае список условий задачи имеет вид «равно(x_1 t_1)», …, «равно(x_k t_k)», где t_1, …, t_k — выражения, не зависящие от неизвестных, но зависящие от некоторых параметров задачи, не являющихся исходными данными y_1, …, y_k. Примером такого рода задач являются геометрические задачи на вычисление, посылки которых описывают некоторый геометрический чертеж; выделяются известные величины y_1, …, y_k (переменные), характеризующие этот чертеж, и требуется определить ряд связанных с ним неизвестных величин t_1, …, t_k. Встречающиеся в описании чертежа переменные, обозначающие точки, хотя формально и являются известными, не должны входить в ответ задачи.

Если требуется проверить истинность утверждений при заданных посылках, то используется задача на описание, единственным условием которой служит данное утверждение, а цели суть «полный», «явное», «прямойответ» (обычно добавляется цель «одз»). Неизвестные задачи отсутствуют, и при решении ее будут предприниматься попытки заменить проверяемое утверждение на логическую константу «истина» либо «ложь». Эта константа и будет выдана в качестве ответа.

Цели задач на преобразование оказываются тесно связаны с конкретными предметными областями (например, разложение на множители либо преобразование к виду суммы одночленов в элементарной алгебре; приведение входящих в преобразуемое выражение

тригонометрических функций к общему аргументу в тригонометрии; нахождение асимптотики выражения при отыскании пределов и т. п.). В одной задаче, как правило, сочетаются несколько таких целей (например, цель «разложитьнамножители» и цель «неизвестные $x_1 \ldots x_k$», указывающая, что разложение на множители выражения t необходимо для решения уравнения $t = 0$ относительно $.x_1, \ldots, x_k)$.

Единственной целью сравнительно общего характера для задач на преобразование является цель «упростить», выражающая тенденции к определенной стандартизации выражения. Впрочем, понятие «упростить» трактуется приемами решателя эвристически — в контексте прочих обстоятельств, сопровождающих задачу, так что в различных случаях упрощение одного и того же выражения может дать различные результаты. В то же время цель «упростить» заменяет собой большое число целей частного характера: задачи на нахождение численного значения выражения, на вычисление производной, предела, интеграла и т. п. могут формулироваться как задачи на преобразование, имеющие единственную цель «упростить». Как и в случае задачи на описание, допускается формулировка исходной задачи на преобразование с целью «одз», определяющей присоединение к списку посылок совокупности утверждений, описывающей область допустимых значений параметров.

Целевая установка задачи на исследование, возникающей в большинстве случаев как блок анализа задачи на описание, определяется спецификой этой внешней задачи. В такой целевой установке присутствует цель «неизвестные $x_1 \ldots x_k$», перенесенная из задачи на описание и направляющая вывод следствий таким образом, чтобы увеличить «степень разрешенности» утверждений относительно переменных $x_1, \ldots, x_k$. В случае задач на исследование, возникших при доказательстве утверждений «от противного», используется цель «противоречие», активизирующая попытки установить противоречивость посылок.

3.3.3. Примеры формулировки задач для решателя

Чтобы сделать более понятной приведенную выше общую схему формализации задач для решателя, приведем примеры постановки задач из различных разделов. Хотя в данном пункте будет часто использоваться «скобочная» форма записи, соответствующая внутреннему логическому языку решателя, в действительности при постановке задачи и просмотре процесса ее решения применяется приближенная к стандартной математическая запись. Минимальные необходимые для ввода новых задач сведения об интерфейсе, позволяющем работать с решателем и использующем стандартную запись, будут приведены далее.

Алгебра множеств. Начнем с простейшего раздела — алгебры множеств. Если требуется доказать, например, истинность тождества $c \cup (b \setminus a) = (b \cup c) \setminus a$, предполагая истинными утверждения $a \subseteq b$ и $b \cap c = \varnothing$, то мы приходим к задаче на доказательство, имеющей посылки «множество(a)», «множество(b)», «множество(c)», «содержится(a b)», «равно(пересечение(b c) пусто)» и единственное условие «равно(объединение(c разность(b a)) разность (объединение(b c) a))». Заметим, что первые три посылки, указывающие тип значений переменных (множество), обычно формируются автоматически, и вручную их вводить не нужно. В последующих примерах будем для краткости опускать такие посылки.

Следующий пример — задача на упрощение выражения $(c \cap b) \cup \cup (c \cap a) \setminus d$ в предположении, что выполняется $a \subseteq b$. Она оформляется как задача на преобразование, имеющая посылку «содержится(a b)» и условие «объединение(пересечение(c b) разность(пересечение(c, a) d))». Эта задача имеет цели «упростить» и «одз». Ответом на нее служит выражение «пересечение(b c)».

Простейшим примером задачи на описание может служить задача на решение уравнений $a \cap x = b$, $a \cup x = c$ в множествах, если для известных множеств a,b,c выполнены соотношения $b \subseteq a$, $a \subseteq c$. Посылками такой задачи служат утверждения «содержится(b a)», «содержится(a c)», дополненные утверждениями о том, что a, b, c — множества. Условиями являются утверждения «равно(пересечение(a x) b)», «равно(объединение(a x) c)» и «множество(x)». Как и в случае посылок, обычно добавление условий, указывающих тип значения неизвестных (например, «множество (x)»), выполняется автоматически; в последующих примерах такие условия опускаем. Задача имеет цели «полный», «явное», «прямойответ», «одз», «неизвестные x» и «упростить». Ответом задачи служит утверждение «равно(x объединение (b разность(c a)))».

Другая ситуация в задачах на описание — когда требуется найти не полное описание всех допустимых значений неизвестных, а привести хотя бы один пример таких значений. Так, если нужно найти пример множества x, удовлетворяющего соотношению $a \cup x = b$ в предположении, что для известных множеств a, b выполняется $a \subseteq b$, то рассматривается задача на описание, имеющая посылку «содержится(a b)» и условие «равно(объединение(a x) b)». Эта задача имеет цели «полный», «пример», «неизвестные x», «прямойответ» и «одз». Ответом на нее может служить, например, утверждение «равно(x b)».

Элементарная алгебра. Примеры формулировки задач по элементарной алгебре начнем с вычисления значений константных выражений. В простейших случаях здесь применяется обычная задача на преобразование с целью «упростить». Она имеет единственную вырожденную посылку — логическую константу «истина». Ус-

ловием служит упрощаемое выражение. Представление о том, что является упрощением, по существу, эвристическое. Оно возникло в процессе рассмотрения многих конкретных примеров на упрощение и соответствующих этим примерам коррекций действий решателя. Так, при упрощении выражения $\frac{(5+\sqrt{3})}{4-\sqrt{3}}$ решатель избавляется от иррациональности в знаменателе и выдает ответ $\frac{9\sqrt{3}+23}{13}$; при упрощении выражения $\operatorname{arctg}\left(\frac{1}{3}\right)+\operatorname{arctg}\left(\frac{1}{5}\right)+\operatorname{arctg}\left(\frac{1}{7}\right)+\operatorname{arctg}\left(\frac{1}{8}\right)$ выдается ответ $\pi/4$ и т. п. В зависимости от задачи исходное выражение может остаться неизмененным либо (что дает обучающий материал для продолжения оптимизации приемов) быть преобразованным к худшему виду. Впрочем, в стандартных задачах с «хорошим» ответом такое явление уже сейчас наблюдается достаточно редко.

Если требуется найти точное целочисленное значение выражения, в котором встречаются степени с большим показателем, факториалы и т. п., то рекомендуется цель «упростить» сопровождать целью «число» (в интерфейсе решателя для ввода такой целевой комбинации предусмотрен специальный пункт меню). В этом случае снимаются блокировки на действия, приводящие к длинным десятичным записям, имеющие место в случае обычной задачи на упрощение. Интерпретатор языка ЛОС позволяет выполнять точные арифметические действия с десятичными записями чисел, имеющими до 3000 знаков.

Если требуется получить приближенное значение константного выражения, содержащего арифметические операции, степени, логарифмы, прямые и обратные тригонометрические функции, причем в этом приближенном значении нужно иметь заданное число точных знаков после запятой, то применяется задача на преобразование, имеющая цели «упростить» и «числоценка *N*», где *N* — число точных знаков после запятой. Производя вычисления, решатель получает гарантированные верхнюю и нижнюю оценки рассматриваемого выражения, увеличивая число цифр до тех пор, пока после округления «к ближайшему» в них не совпадут требуемые *N* цифр после запятой.

Наконец, для получения приближенной оценки константного выражения возможно использование встроенного в компьютер математического сопроцессора (14 значащих цифр). Здесь используется комбинация целей «числзначение» и «выч».

В случае задач на упрощение неконстантных алгебраических выражений, как и для константных, используется сочетание целей «упростить» и «одз». Например, для упрощения в ОДЗ выражения

$$\frac{a}{a^2+b^2}-\frac{b(a-b)^2}{a^4-b^4}$$

создается задача на преобразование, имеющая своим условием это выражение, посылками — утверждения «число(a)» и «число(b)» и указанные две цели. На нее выдается ответ $\frac{1}{a+b}$. Необходимые условия на ОДЗ (отличие знаменателей от нуля) вводятся решателем в список посылок самостоятельно и в процессе решения изменяются так, чтобы не нарушалось соответствие между ними и «обслуживаемыми» ими подвыражениями условия задачи. Собственно в ответ результирующие условия на ОДЗ для параметров не включаются, но они легко могут быть считаны при пошаговом просмотре решения.

Задачи разложения на множители имеют целевую установку «упростить», «разложитьнамножители», «одз». Так, для разложения на множители выражения $a^2 - b^2 - c^2 - 2bc$ создается задача на преобразование с этими целями, условием которой служит данное выражение, а посылки указывают тип значения переменных a, b, c. Заметим, что задачи на преобразование тригонометрических выражений к виду, удобному для логарифмирования, формулируются с теми же целями, что и обычные алгебраические задачи разложения на множители. Если требуется разложить на множители выражение, разрешая использование комплексных чисел, то кроме указанных выше трех целей добавляется четвертая — «видУмножение».

Чтобы разложить алгебраическую дробь в сумму простейших дробей, используется задача на преобразование, имеющая цели «простейшиедроби», «упростить», «одз». Например, решая задачу на преобразование с указанными целями и условием

$$\frac{1}{(a^2-a+1)(a^2+2a+3)},$$

решатель выдает ответ

$$\frac{3a+4}{19(a^2+2a+3)}+\frac{5-3a}{19(a^2-a+1)}.$$

Наконец, для упрощения алгебраических выражений с «раскрыванием скобок» предусмотрена целевая установка «упростить», «раскрытьскобки», «одз». Так, решая задачу на преобразование с данными целями и условием $a(b-c)+c(a-d)+d(c-b)$, решатель выдает ответ $ab - bd$. Хотя в этом ответе и можно было бы вынести за скобку общий множитель b, но цель «раскрытьскобки» блокирует такое действие.

Если требуется упростить некоторое выражение, вместо параметров которого должны быть подставлены другие выражения, то такую подстановку не обязательно делать непосредственно — достаточно указать в посылках задачи равенства, определяющие

подставляемые выражения. Например, если имеется задача на преобразование с условием

$$-4a^2x^{\frac{1}{m}+\frac{1}{n}}+\left(x^{\frac{1}{m}}+x^{\frac{1}{n}}\right)^2$$

и посылкой $x=(\sqrt{a^2-1}+a)^{\frac{2mn}{m-n}}$ (не считая автоматически добавляемых посылок, указывающих тип значений переменных), то фактически будет упрощаться результат подстановки в условие выражения для x, определяемого посылкой.

При решении задачи на упрощение алгебраического выражения может возникнуть разбор подслучаев, в результате чего ответ будет иметь вид «условного» выражения. Так, при упрощении выражения

$$\sqrt{a+2\sqrt{2a-4}}+\sqrt{a-2\sqrt{2a-4}}$$

решатель выдает ответ «$2\sqrt{2}$ при $a<4$, иначе $2\sqrt{a-2}$» (для большей наглядности вместо скобочной конструкции «вариант(…)» мы использовали здесь стандартную запись, прорисовываемую формульным редактором решателя).

Задачи на суммирование также оформляются как стандартные задачи на упрощение. Например, задача на преобразование с целями «упростить», «одз», условием $\sum_{m=1}^{n}(2m-1)^3$ и посылкой «натуральное(n)» приводится к ответу $(2n^2-1)n^2$. Заметим, что в этих задачах важно указывать в посылках, что значениями параметров, определяющих границы суммирования, служат целые числа, и сопровождать такие параметры всеми необходимыми неравенствами.

Задачи на решение уравнений, неравенств, систем уравнений и неравенств практически всегда имеют целевую установку «полный», «явное», «прямойответ», «одз», «упростить», «неизвестные $x_1…x_n$». Например, чтобы решить уравнение

$$\frac{\sqrt{1+a^2x^2}-ax}{\sqrt{1+a^2x^2}+ax}=\frac{1}{b^2}$$

относительно неизвестной x, создается задача на описание с указанной выше целевой установкой, условиями которой служат данное уравнение и утверждение «число(x)» (последнее создается автоматически процедурами интерфейса решателя). Эта задача имеет посылку «число(a)». Решая задачи с параметрами, решатель аккуратно анализирует все возможные случаи, объединяя полученные для них результаты в общем ответе. Так, в указанном уравнении ответ будет объединять два подслучая: $a\neq 0, b\neq 0, x=\frac{b^2-1}{2a|b|}$ и ($b=-1 \vee b=1$), $a=0$ с «вынесенной за скобку» общей частью «число(x)».

Если в задаче требуется найти все значения параметров, при которых система уравнений либо неравенств имеет хотя бы одно решение, то условие записывается с помощью квантора существования. Например, для нахождения всех значений параметра a, при которых имеет решение система уравнений

$$\begin{cases} y(ax-1) = 2|x+1| + 2xy, \\ xy + 1 = x - y, \end{cases}$$

вводится задача на описание с условием

$$\exists xy(y(ax-1) = 2|x+1| + 2xy \;\&\; xy + 1 = x - y).$$

Целевая установка у нее — такая же, как и выше (но роль неизвестной играет a).

Аналогично если нужно найти все значения параметров, при которых система уравнений либо неравенств имеет заданное число решений, то используется описатель «класс», с помощью которого обозначается множество решений рассматриваемой системы. Так, для отыскания значений параметра a, при которых система

$$\begin{cases} \log_2 y + 2 = \log_2(x+3y), \\ y = 2(x-a)^2 + x + 2a - 4 \end{cases}$$

имеет ровно два решения, создается задача на описание с условием «мощность(класс($x\,y(\log_2 y + 2 = \log_2(x+3y)$ & $y = 2(x-a)^2 + x + 2a - 4$))) = 2». Если требуется найти значения параметров, при которых две системы (два уравнения и т. п.) имеют одинаковые множества решений, то условие записывается в виде эквивалентности под квантором общности. Например, чтобы найти значения параметра a, при которых множество решений уравнения $4(\cos x)^2 = a^2 - 6$ совпадает со множеством решений уравнения $1 - \cos(2x) = a/6$, используется задача на описание с условием

$$\forall x(4(\cos x)^2 = a^2 - 6 \Leftrightarrow 1 - \cos(2x) = a/6).$$

Заметим, что в перечисленных примерах под кванторами и описателем «класс» автоматически вводятся дополнительные утверждения «число(…)», уточняющие тип значений связанных переменных.

Если требуется определить число решений уравнения (системы) в зависимости от значений параметров, то возникает уже не задача на описание, а задача на преобразование. Условием такой задачи служит выражение, задающее мощность множества корней, а цели — стандартные: «упростить» и «одз». Решатель пытается получить явное представление для множества корней, решая рассматриваемое уравнение (систему), либо иными средствами определяет мощность этого множества (например, анализируя с помо-

щью производных и пределов интервалы монотонности). Так, если нужно определить число решений уравнения

$$\sqrt{\sqrt{x+\frac{1}{4}}+x+\frac{1}{2}}=a,$$

то условием задачи на преобразование будет выражение

$$\text{«мощность(класс}(x\sqrt{\sqrt{x+\frac{1}{4}}+x+\frac{1}{2}}=a))\text{»}.$$

Ответ, получаемый здесь решателем, имеет вид «1, если $1/4 \le a$, иначе 0».

При доказательстве тождества либо неравенства создается задача на доказательство, условием которой является это тождество либо неравенство, а посылки перечисляют ограничения на параметры. Так как задача на доказательство не имеет списка целей, то указание на пополнение списка посылок ограничениями на область допустимых значений передается ей не в виде цели «одз», а в виде комментария «одз». Это делается автоматически интерфейсом решателя. В качестве примера приведем задачу на доказательство с условием $64ab(a+b)^2 \le (\sqrt{a}+\sqrt{b})^8$ и посылками «число(a)», «число(b)». Неотрицательность параметров a, b регистрируется в списке посылок уже при решении задачи, когда обрабатывается комментарий «одз».

Элементарная геометрия. Вычислительные задачи по планиметрии внешне напоминают задачи на решение уравнений и неравенств — в обоих случаях требуется определить значения «неизвестных» величин. Однако здесь имеются существенные различия, приводящие к необходимости особой логической формализации. Прежде всего, параметры посылок геометрической задачи бывают двух типов — числовые, про которые обычно предполагается, что они «известны», и параметры — обозначения точек чертежа. Последние, хотя и содержатся в посылках, «известными» никоим образом не считаются и в ответ входить не должны. Более того, те величины, которые требуется определить в геометрической задаче, обычно изначально явно выражены через ее «точечные» параметры, так что с точки зрения обычной «алгебраического типа» задачи на описание ситуация вырожденная. Эти обстоятельства потребовали ввести для геометрических задач на вычисление специальную цель «известно $a_1 \ldots a_n$», которая указывает все параметры a_1, …, a_n, которые могут встречаться в ответе (возможен случай $n = 0$, и тогда цель сводится к логическому символу «известно»). Изначально имеющаяся информация о чертеже перечисляется в списке посылок задачи; условиями служат равенства, явно выражающие неизвестные величины через обозначения точек чертежа.

В качестве простейшего примера рассмотрим следующую задачу. В треугольнике ABC длина стороны AC равна a; угол BAC равен b, а угол BCA равен c. Требуется вычислить площадь данного треугольника. Эта ситуация формализуется в виде задачи на описание, посылками которой служат следующие утверждения: «треугольник(ABC)»; «расстояние(AC) = a»; «угол(BAC) = b»; «угол(BCA) = c». Они дополняются вводимыми автоматически утверждениями о типе значения: «точка(A)», «точка(B)», «точка(C)», «число(a)», «число(b)», «число(c)». Задача имеет условия «x = площадь(фигура (A B C))» и «число(x)». Целевая установка ее состоит из целей «неизвестные x»; «известно a b c»; «полный»; «прямойответ»; «явное»; «упростить»; «одз».

При формулировке утверждений, описывающих чертеж геометрической задачи, следует учитывать некоторую ограниченность созданного на текущий момент запаса понятий. Так, например, отсутствует в явном виде понятие «медиана треугольника». Если в условии задачи говорится про медиану, то нужно вводить в рассмотрение новую точку — основание медианы — и указывать в посылках, что расстояния ее до концов стороны треугольника равны, а сама эта точка лежит на данной стороне (т. е. на отрезке с соответствующими концами). Впрочем, последнее условие можно ослабить: задать принадлежность точки не стороне, а прямой, на которой лежит эта сторона. Приведем несколько примеров простых задач, в которых встречаются такого рода «стандартные» для решателя формулировки геометрических условий.

Пусть имеется треугольник ABC, у которого длина стороны AB равна 6, длина стороны BC равна 8; медианы, проведенные из вершин A и C, взаимно перпендикулярны. Требуется найти площадь треугольника. Посылки соответствующей задачи на описание таковы: «треугольник(ABC)»; «расстояние(AB) = 6»; «расстояние(BC) = 8»; «D ∈ отрезок(BC)»; «расстояние(BD) = расстояние(CD)»; «E ∈ отрезок(AB)»; «расстояние(AE) = расстояние(BE)»; «перпендикулярно(прямая (CE) прямая (AD))». Условием служит равенство «x = площадь(фигура(ABC))». Задача имеет цели «неизвестные x», «известно», «полный», «прямойответ», «явное», «упростить», «одз».

Аналогичные примеры приведем для биссектрисы и высоты треугольника. В первом из них даны длины a, b, c сторон BC, AC, AB треугольника ABC. Биссектрисы треугольника, проведенные из вершин B и C, пересекаются в точке D. Требуется найти, в каком отношении точка D делит биссектрису, проведенную из вершины B. Посылками задачи для этого примера служат утверждения «треугольник(ABC)»; «расстояние(BC) = a»; «расстояние(AC) = b»; «расстояние(AB) = c»; «биссектриса($ABCR$)» (введена точка R — основание биссектрисы угла B); «R ∈ отрезок(AC)» (точка R лежит на стороне AC);

«биссектриса(*ACBP*)»; «*P* ∈ отрезок(*AB*)» (аналогично для биссектрисы угла *C* введено основание *P*); «*D* ∈ отрезок(*BR*)»; «*D* ∈ отрезок(*CP*)» (введена в рассмотрение точка *D* пересечения биссектрис). Условие задачи имеет вид «*x* = расстояние(*BD*)/расстояние(*DR*)». Цели задачи: «неизвестные *x*»; «известно *a b c*»; «полный»; «явное»; «прямой-ответ»; «одз»; «упростить».

В другом примере известно, что длина стороны *BC* треугольника *ABC* равна 8; длины высот треугольника, проведенных из вершин *A* и *B*, равны, соответственно, 4 и 6,4. Требуется найти длины сторон *AB* и *AC*. Посылки задачи: «треугольник(*ABC*)»; «расстояние(*BC*) = 8»; «*D* ∈ прямая(*BC*)» (введено основание *D* высоты, проведенной из *A*); «перпендикулярно(прямая(*AD*) прямая(*BC*))» (высота перпендикулярна основанию); «*E* ∈ прямая(*AC*)» (основание второй высоты); «перпендикулярно(прямая(*BE*) прямая(*AC*))»; «расстояние(*AD*) = 4»; «расстояние(*BE*) = 6.4». Условия этой задачи имеют вид: «*x* = расстояние(*AB*)»; «*y* = расстояние(*AC*)».

Заметим, что в планиметрических задачах обычно имеется фиктивная посылка «планиметрия». Как правило, она вводится автоматически после набора задачи, однако иногда ее приходится вводить вручную (последовательным нажатием клавиш «п», «и»). Эта посылка необходима для ускорения проверок принадлежности рассматриваемых прямых и точек общей плоскости. В особых случаях ее отсутствие может также привести к выдаче отказа на задачу.

Для указания на то, что точка принадлежит внутренности фигуры, ограниченной системой лучей и отрезков, используются утверждения «однасторона(…)»; «разныестороны(…)». Так, рассмотрим следующую задачу. Внутри угла *BAC*, величина которого равна *a*, взята точка *M*. Из нее опущены перпендикуляры на стороны угла, причем известно, что основания перпендикуляров отстоят от вершины угла на расстояния *p* и *q*. Нужно найти длины перпендикуляров. Данная ситуация описывается следующей системой посылок: «угол(*BAC*) = *a*»; «однасторона(*C*, *M*, прямая(*AB*))»; «однасторона(*B*, *M*, прямая(*AC*))» (условие принадлежности точки *M* углу сформулировано как пара условий: эта точка лежит по ту же сторону от одной стороны угла, что и направляющая точка другой стороны); «перпендикулярно(прямая (*PM*) прямая(*AB*))»; «*P* ∈ прямая(*AB*)» (*P* — основание первого перпендикуляра); «перпендикулярно(прямая(*QM*) прямая(*AC*))»; «*Q* ∈ прямая(*AC*)» (*Q* — основание второго перпендикуляра); «расстояние(*AP*) = *p*»; «расстояние(*AQ*) = *q*»; 0 < *a*; *a* < π (невырожденность угла); 0 < *p*; 0 < *q* (эти два неравенства уточняют, что точка *M* находится во внутренности угла). Условия задачи: «*x* = расстояние(*PM*)»; «*y* = расстояние(*QM*)». Заметим, что условие принадлежности точки углу могло быть сформулировано и непосредственным образом как «*M* ∈ Угол(*BAC*)». Приведенная выше более громоздкая формули-

ровка нужна здесь лишь как иллюстрация возможного применения предикатов «однасторона» и «разныестороны».

Геометрические задачи на доказательство формулируются почти так же, как и задачи на вычисление: в посылках задается чертеж; условием служит утверждение, которое требуется доказать. Например, задача на доказательство того, что каждый выпуклый четырехугольник, диагонали которого суть биссектрисы его внутренних углов, является ромбом, имеет следующие посылки: «четырехугольник(*ABCD*)»; «биссектриса(*BADC*)»; «биссектриса(*BCDA*)»; «биссектриса(*ABCD*)»; «биссектриса(*ADCB*)». Условие этой задачи — «ромб(*ABCD*)».

Геометрические задачи на построение и на определение геометрического места точек хорошо формализуются в виде обычных задач на описание, имеющих стандартную целевую установку «неизвестные $x_1 \ldots x_n$», «полный», «явное», «прямойответ», «упростить», «одз». В качестве примера рассмотрим задачу нахождения геометрического места всех таких точек *C* на данной прямой *AB*, сумма расстояний которых до точек *A*, *B* наименьшая. Предполагается известным, что расстояние от *A* до *B* равно 50. Она формализуется как задача на описание, имеющая посылку «расстояние(*AB*) = 50» и условие «Минимум(отображение(*C* точка(*C*) расстояние(*AC*) + расстояние(*BC*)) прямая(*AB*) *x y*)». Неизвестные этой задачи — *x*, *y*; значение первой из них равно наименьшей сумме рассматриваемых расстояний; значение второй — множеству точек, в которых достигается эта наименьшая сумма. Решатель выдает на задачу ответ «*y* = отрезок(*AB*); *x* = 50».

Другой пример — задача на построение треугольника *ABC* по известным длинам *a*, *b*, *c* его сторон *AB*, *AC*, *BC*. Она формализуется в виде задачи на описание с условиями «треугольник(*ABC*)»; «расстояние(*AB*) = *a*»; «расстояние(*A C*) = *b*»; «расстояние(*BC*) = *c*». Неизвестными этой задачи служат *A*, *B*, *C*. Решатель выдает на данную задачу ответ «$0 < a + b - c$, $0 < a + c - b$, $0 < b + c - a$, точка (*C*), $A \in$ окр(*Cb*), $B \in$ окр(*Aa*) $\cap$ окр(*Cc*)». Напомним, что «окр(*Ab*)» обозначает окружность с центром в точке *A*, имеющую радиус *b*. Разумеется, в задачах на построение вместо циркуля и линейки используется ряд вспомогательных элементарных операций и отношений, позволяющих последовательно определять новые точки по ранее найденным.

Примеры формулировок задач для других разделов математики можно найти в работе [8, параграф 3.3].

3.3.4. Интерфейс ввода задач

Оглавление задачника. В решателе имеется сборник задач, в котором сохраняются (для нужд тестирования и регулировки) рассматривавшиеся при обучении задачи либо вводятся новые за-

дачи. Этот сборник снабжен древовидным оглавлением, разбивающим его на разделы, подразделы и т. п. Вход в оглавление задачника осуществляется из главного меню при нажатии клавиши «з» (здесь и далее, если не оговорено особо, все буквенные названия клавиш — русские) либо нажатии левой кнопки мыши на пункте меню «Оглавление задачника». После этого происходит переход к тому пункту оглавления, с которым велась работа при предыдущем пребывании в задачнике. Переход между пунктами оглавления выполняется с помощью клавиш курсора: нажатие клавиши «курсор влево» приводит к переходу в надраздел текущего раздела; клавиши «курсор вверх» и «курсор вниз» позволяют выбирать текущий пункт в текущем разделе; клавиша «курсор вправо» приводит к просмотру содержимого текущего пункта.

Для ускоренного выбора пункта оглавления можно использовать курсор мыши: нажатие левой клавиши на выбранном пункте приводит к переходу внутрь этого пункта; нажатие (в любом месте) правой клавиши мыши — к возвращению в надраздел. Пункты каждого раздела пронумерованы, причем если пункт соответствует входу в подраздел, то после его номера стоит закрывающая скобка, если же пункт соответствует отдельной задаче задачника, то после его номера стоит точка. Из любого пункта оглавления можно вернуться к главному меню, нажав клавишу *End*. Пункты корневого раздела имеют названия «Дискретная математика», «Алгебра множеств», «Элементарная алгебра», «Элементарная геометрия», «Математический анализ», «Дифференциальные уравнения», «Аналитическая геометрия» и др.

Оглавление задачника организовано таким образом, что все его концевые пункты (соответствующие отдельным задачам) собраны в специальные концевые разделы. Все пункты такого концевого раздела — только концевые. Обычно в концевом разделе перечисляются только номера задач, после которых стоят три тире. Прочие (неконцевые) пункты оглавления имеют текстовые подзаголовки. В принципе, с любым пунктом оглавления (концевым либо неконцевым) можно связать произвольные текстовые примечания. Для этого достаточно выбрать нужный пункт и нажать клавишу «р» — возникает курсор текстового редактора (особенности текстового редактора решателя см. в последующих разделах, посвященных его интерфейсу). После завершения редактирования (оно происходит при нажатии клавиши *Enter*) сохраняется измененный текст пункта.

После выбора текущего концевого пункта и нажатия клавиши «курсор вправо» происходит переход к просмотру задачи этого пункта. Все задачи одного и того же концевого раздела соединены при таком просмотре в одну ленту, которую можно прокручивать вверх и вниз при помощи клавиш «курсор вверх», «курсор вниз» и *PageUp*, *PageDown*, не выходя обратно в оглавление задачника. Одна зада-

ча отделяется на этой ленте от другой сплошной горизонтальной линией. Кроме указанных выше клавиш, при просмотре задач одного концевого раздела удобно использовать сочетания клавиш *Ctrl* + *PageUp* и *Ctrl* + *PageDown*. Первое из них позволяет переходить к просмотру начала предыдущей задачи, второе — к просмотру начала следующей задачи.

Просмотр задач и основные операции над задачами. Область между горизонтальными линиями, ограничивающими описание задачи (для последней задачи нижняя линия отсутствует), называется *полем задачи*. Элементы задачи размещаются в ее поле следующим образом. Вначале идут посылки задачи. Затем располагается текст-формульный элемент, задающий целевую установку задачи. Далее (кроме задач на исследование) идут условия задачи (в случае задач на доказательство либо преобразование — единственное условие). Если задача ранее была решена системой, то в конце располагается найденный на нее ответ. Задача по геометрии может быть сопровождена чертежом, который размещается до ее посылок. Если задача не имеет невырожденных посылок (отличных от константы «истина» и простейших указателей на типы значений переменных), то целевая установка размещается непосредственно в начале поля.

В некоторых разделах (теория вероятностей, элементарная физика, комбинаторика) логическая формализация текста задачи выглядит громоздко и трудна для восприятия. Поэтому в решателе предусмотрена возможность сохранять также исходный текст задачи. Если этот текст был сохранен и при просмотре задачи нажимается клавиша «н», то текст появляется на экране. Данная операция адресуется той задаче, к которой относится верхняя из имеющихся на экране отделяющих горизонтальных линий. Чтобы вернуться к просмотру формальной версии условия, нажимается клавиша *End* или *Esc*.

Существуют различные режимы просмотра задачи. По умолчанию задачи прорисовываются с помощью стандартной математической записи и с пропуском ряда простейших посылок и условий. Чтобы перейти к режиму, в котором используется внутренний логический язык (далее называемому скобочной записью), нажимается сочетание клавиш *Ctrl* + «с»; оно же возвращает обратно в режим стандартной математической записи. Для перехода в режим полного просмотра посылок и условий и выхода из этого режима служит клавиша «ы».

Для выполнения различных операций с задачей и ее элементами выполняется выбор задачи либо ее элементов с помощью мыши. Курсор мыши подводится к выбираемому элементу либо помещается в поле выбираемой задачи. При выборе элемента задачи нажимается левая клавиша мыши, при выборе всей задачи — правая. Чтобы произошел выбор задачи, на экране обязательно должна быть прорисована ее верхняя отделяющая линия. Выбранный эле-

мент перекрашивается в голубой цвет. Чтобы отменить выбор элемента, выполняется повторно такое же нажатие клавиши мыши. Если нужно отменить выбор сразу всех выбранных на текущий момент элементов и задач, нажимается клавиша *Space* («пробел»). Можно выделять не только отдельную посылку либо условие задачи, но и ее фрагмент. Для этого, сразу после выделения посылки либо условия *F*, следует нажать клавишу «курсор вправо». Тогда будет выделен (перекрашен в голубой цвет) первый операнд *F*. Далее клавишами «курсор вправо», «курсор влево» можно переходить от операнда к операнду на одном уровне; клавишей «курсор вниз» — переходить к подоперандам текущего операнда; клавишей «курсор вверх» — возвращаться к надоперанду. Нажатие любой клавиши, отличной от клавиш курсора, завершает данный режим выделения фрагмента.

Перечислим некоторые операции, которые можно совершать после выделения терма задачи.

Прежде всего, нажатие клавиши «с» позволяет переходить от стандартной математической записи этого терма к скобочной и обратно. При нажатии клавиши *Enter* включается режим вставки формульным редактором нового терма задачи непосредственно перед выделенным термом (если он располагается до целевой установки, то становится посылкой задачи, если после — условием). Ниже будут приведены основные правила использования формульного редактора при вводе элементов задачи.

Если нужно вставить новый терм перед выделенным термом с помощью текстового редактора (т. е. в скобочной записи), то нажимается клавиша «Т». Для изменения ранее набранного терма имеется три возможности. Первая из них — использование текстового редактора для работы с текстом скобочной записи этого терма. Для входа в этот режим нажимается *Ctrl* + «т». Вторая — использование формульного редактора для повторного набора новой версии терма. Новая версия набирается непосредственно под старой, которая во время набора сохраняется, а затем удаляется. Для входа в этот режим используется сочетание клавиш *Ctrl* + «ф». Наконец, можно войти в формульный редактор для продолжения набора выделенного терма, начиная с его конца. При необходимости (используя клавишу *Backspace*) здесь можно сначала исключить часть ранее набранных последних символов, а затем добавить новые. Для входа в такой режим нажимается «ф». Чаще всего для изменения ранее введенного терма применяется первый режим (скобочной записи), так как он позволяет вносить изменения в любом месте текста.

Чтобы удалить выделенный элемент задачи, нажимается сочетание клавиш *Ctrl* + *Del*. Если была выделена вся задача, то нажатие *Ctrl* + *Del* удаляет всю задачу целиком. Выделенную задачу можно скопировать нажатием клавиши «к». Если выделена только одна за-

дача, то копия располагается в конце текущего списка задач (т. е. текущего концевого раздела оглавления задачника). Если нужно разместить эту копию перед некоторой ранее введенной задачей списка (текущего либо относящегося к другому разделу), то перед нажатием клавиши «к» выделяется также последняя задача.

Чтобы исключить из задачника ранее найденный на выделенную задачу ответ, нажимается сочетание клавиш *Ctrl + F4*. Выделенный терм задачи можно скопировать, выделив предварительно тот терм, перед которым нужно поместить копию (если ее нужно поместить в конец списка посылок либо списка условий, то выделяется вся задача) и нажав затем клавишу *Insert*. Аналогичным образом можно перенести выделенный терм на новое место; для этого служит сочетание клавиш *Ctrl + Insert*.

Если выделить несколько термов либо фрагментов термов задачи, то их можно использовать при наборе новых термов формульным редактором. Следует лишь запомнить порядок, в котором они выделялись. Если в режиме формульного редактора нажать сначала клавишу *Insert*, а затем номер (начиная с 1 и не более 9) выделенного указанным образом терма, то произойдет автоматическая вставка его в набираемый терм начиная с текущей позиции. Такую вставку заданного терма в процессе набора можно выполнять многократно.

Ввод новой задачи. Ввод новой задачи инициируется одним из двух способов. Первый из них состоит в том, чтобы перейти в конец просматриваемого списка задач и нажать клавишу «з» (если нужно ввести новую задачу перед ранее введенной, то сначала выделяется последняя задача, а затем нажимается «з»). Тогда возникает новая горизонтальная линия, которая является первым элементом новой задачи.

Второй способ начать ввод новой задачи — выйти из просмотра списка задач в концевой раздел оглавления задачника, соответствующий данному списку, и ввести новый его концевой пункт, который автоматически становится заготовкой новой задачи. Если этот пункт вводится в конце раздела, то нажимается клавиша «к»; если же его нужно ввести перед некоторым другим пунктом, то происходит выбор последнего (клавишами «курсор вверх», «курсор вниз») и нажимается «К». При входе в список задач через новый концевой пункт оглавления попадаем на начало новой задачи, которое состоит из единственного ее элемента — верхней горизонтальной отделяющей линии. Если новая задача вводится перед ранее введенной нажатием клавиши «з», то она автоматически при этом выделяется (ее верхняя линия окрашена в голубой цвет).

После того как введена верхняя отделяющая линия задачи, можно переходить к вводу посылок, условий и целевой установки задачи. В принципе, их допустимо набирать в произвольной после-

довательности. Рекомендуемым является следующий порядок, при котором уменьшается число операций с клавиатурой: сначала вводятся посылки задачи, затем целевая установка и в конце — условие (условия). Чертеж геометрической задачи обычно вводится в последнюю очередь либо формируется автоматически (путем нажатия клавиши «Ч»), хотя можно начинать ввод задачи и непосредственно с чертежа. Для ввода очередной посылки нажимается клавиша *Enter* — чтобы войти в формульный редактор, либо «Т» — чтобы войти в текстовый редактор (последнее выполняется лишь в исключительных случаях, если для рассматриваемых логических символов еще не обеспечена их прорисовка формульным редактором).

Вырожденные посылки, указывающие тип значений переменных либо необходимые для указания области допустимых значений, вводить вообще не нужно — они создаются решателем автоматически (кроме случаев, когда тип значения переменной не подсказывается выражениями из описания задачи, в которых эта переменная встречается).

По окончании ввода посылок (если посылок нет, то сразу же) начинается формирование целевой установки задачи. Здесь имеется два режима — с использованием оглавления типов целевых установок и путем непосредственного набора перечня целей текстовым редактором. Рекомендуется использовать первый режим. Для входа в него нажимается клавиша «ц». Эта же клавиша используется для изменения ранее введенной целевой установки — предварительно надо выделить задачу либо убедиться, что задача является последней в текущем списке задач. После нажатия клавиши «ц» на экране появляется раздел оглавления типов целевых установок, с которым происходила работа в предыдущий раз. Это оглавление (и вообще все оглавления решателя) устроено аналогично оглавлению задачника.

Корневой раздел оглавления типов целевых установок содержит следующие пункты.

1. «Доказать утверждение». Этот пункт выбирается при создании задачи на доказательство. После выбора его («курсор вверх», «курсор вниз») и нажатия клавиши «курсор вправо» автоматически выбирается тип задачи «доказать» и происходит возвращение в набор текста задачи.

2. «Проверить истинность утверждения». Здесь создается задача на описание, предназначенная для проверки истинности ее единственного условия. Цели ее — «полный», «явное», «прямойответ», «одз». Возможные ответы (кроме отказа) — «истина» либо «ложь». Как и выше, для выбора данной целевой установки нажимается клавиша «курсор вправо».

3. «Преобразовать выражение». Этот пункт является подразделом оглавления, в котором собраны различные целевые установки задач на преобразование (см. ниже).

4. «Найти значения неизвестных». Этот пункт является подразделом оглавления, в котором собраны различные целевые установки задач на описание (см. ниже), связанных с нахождением неизвестных.

5. «Исследовать свойства объекта». Этот пункт является подразделом оглавления, в котором собраны различные целевые установки задач на общую характеризацию свойств объектов различных типов. Все они оформляются как задачи на описание, причем ответом служит некоторая группа утверждений, полученных в процессе анализа ситуации и дающих типовую характеризацию требуемого вида.

6. «Переформулировать условие». Создается задача на описание, имеющая цели «полный», «прямойответ», «редакция». Это эквивалент словесной формулировки «найти условия, при которых …», означающей необходимость эквивалентного преобразования исходной системы условий к более простому и явному виду.

7. «Построить график». Пункт позволяет создавать задачу на построение графика. Это задача на преобразование с целями «числзначение», «график». Ее условие — выражение, определяющее зависимость, для которой строится график. При решении задачи инициируется диалог для определения варьируемой переменной, промежутка ее значений и значений фиксированных параметров. Для работы с возникающим после этого графиком предусмотрен специальный интерфейс, позволяющий изменять масштаб, параметры и область просмотра (подробнее о нем — в нижеследующих разделах). После выбора данного пункта и нажатия клавиши «курсор вправо» происходит возвращение в набираемую задачу, где прорисовывается строка «Построить график:».

8. «Игровые задачи». Пункт является подразделом оглавления, который на текущий момент используется для обучения решателя игре в шахматы. Предусмотрена инициализация шахматной партии либо создание заданной позиции для ее анализа решателем. Обучение находится на начальном этапе, и использовать эту возможность интерфейса рекомендуется лишь тем, кто захотел бы самостоятельно продолжить развитие решателя.

Подраздел «Преобразовать выражение» оглавления типов целевых установок содержит следующие пункты.

1. «Упростить выражение в области допустимых значений». Этот пункт позволяет вводить задачу на преобразование с целями «упростить», «одз». Такой набор целей — стандартный для большинства задач на преобразование, включая вычисление пределов, производных, интегралов и определителей матриц.

2. «Вычисление значения константного выражения». Этот пункт является подразделом оглавления, в котором собраны различные

целевые установки задач на нахождение точного либо приближенного численного значения константного выражения (см. ниже).

3. «Раскрыть скобки». Создается задача на преобразование, ориентированная на приведение выражения к виду суммы одночленов. Она имеет цели «упростить», «раскрытьскобки», «одз».

4. «Разложить на вещественные множители». Создается задача на преобразование, в которой нужно разложить выражение на вещественные множители. Она имеет цели «упростить», «разложитьнамножители», «одз».

5. «Разложить на комплексные множители». Аналогично предыдущему, но допускается переход к выражениям с мнимой единицей. Задача имеет те же цели, что и выше, к которым добавляется цель «видУмножение». В случае появления комплексных множителей вместо вещественной операции «умножение» возникает комплексная операция «Умножение».

6. «Представить в виде суммы простейших дробей». Вводится задача на преобразование, в которой нужно представить выражение в виде суммы простейших дробей. Предполагается, что это выражение имеет единственную переменную (в действительности процедуры решателя рассчитаны на приведение к виду суммы простейших дробей выражения с несколькими переменными относительно явно указанной одной из них, что применяется, например, при интегрировании, однако это — «внутренняя» возможность, не вынесенная во внешний интерфейс). Цели задачи — «упростить», «простейшиедроби», «одз».

7. «Разложить в ряд степенной ряд». Создается задача на разложение заданного выражения по заданной переменной x в степенной ряд в окрестности заданной точки t. Здесь имеется в виду получение бесконечной суммы, содержащей все члены ряда. Если в посылках задачи содержится утверждение «комплексное(x)», то предпринимается попытка получить разложение в комплекснозначный ряд Тейлора либо Лорана. После выбора данной целевой установки («курсор вправо») происходит возвращение в текст набираемой задачи, к которому добавляется строка «Разложить в степенной ряд по переменной». Под этой строкой в левой части экрана размещается курсор формульного редактора, которым нужно ввести переменную разложения x. После ввода этой переменной (ввод завершается нажатием клавиши *Enter*) к указанной строке добавляется «x в точке», и снова в левой части возникает курсор формульного редактора. Здесь уже нужно ввести выражение t, определяющее точку разложения. По окончании ввода (нажатие клавиши *Enter*) t перерисовывается в конце строки, задающей целевую установку, и эта установка завершается двоеточием — далее можно вводить выражение, которое следует разложить в ряд. Такого рода диалоги используются в различных описываемых далее целевых установках.

Цели задачи на разложение в степенной ряд — «упростить», «рядтейлора *x t*», «одз».

8. «Получить заданное число членов разложения в ряд Тейлора». Аналогично предыдущему, но требуется получить заданное конечное число *n* членов ряда. Диалог ввода целевой установки происходит по тому же сценарию, что и в предыдущем пункте. Однако после ввода выражения *t* для точки разложения к строке текста целевой установки добавляются слова «до членов степени» и далее вводится формульным редактором константа *n*. Цели получаемой задачи — «упростить», «формулатейлора *x t n*» «одз».

9. «Разложить в ряд Фурье». Вводится задача на разложение заданного выражения по заданной переменной *x* в ряд Фурье на отрезке [*a*, *b*]. Вначале диалога ввода целевой установки прорисовывается строка «Разложить в ряд Фурье по переменной», после чего формульным редактором вводится переменная разложения *x*. Далее (после *x*) к строке добавляются слова «на отрезке», и формульным редактором вводится выражение [*a*, *b*] (для ввода числового промежутка используется сочетание клавиш *Ctrl* + «квадратная скобка»). Цели получаемой задачи — «упростить», «рядфурье *x a b*», «одз».

10. «Получить явное описание класса». Это целевая установка задач, в которых множество, изначально заданное с помощью описателя «класс», нужно переформулировать в явном виде, не использующем описателей «класс» и «отображение». К числу таких задач относятся, например, задачи на определение геометрического места точек. Возникающая здесь целевая установка состоит из целей «упростить», «класс», «одз».

11. «Получить асимптотическую оценку». Это целевая установка задачи на преобразование, в которой требуется получить асимптотическую оценку исходного выражения для предельного поведения переменных, заданного посылками «стремится(…)». Установка состоит из целей «асимптоценка», «упростить», «одз».

Подраздел «Вычисление значения константного выражения» раздела «Преобразовать выражение» содержит следующие пункты.

1. «Найти точное целочисленное значение». В этом случае условие задачи на преобразование построено при помощи целочисленных операций (сложение, вычитание, умножение, степень с натуральным показателем, факториал, модуль и т. п.) из целочисленных констант, и требуется найти его точное численное значение. Число десятичных знаков ограничивается лишь возможностями интерпретатора (свыше 600) и ограничителями в приемах, которые легко ослабить либо вовсе убрать. Задача имеет цели «упростить» и «число».

2. «Вычислить с заданной точностью». Здесь определяется приближенное значение константного выражения, в котором гарантированы первые *n* знаков после запятой. После выбора данной целевой установки происходит возвращение в набор текста зада-

чи, где прорисовывается строка «Вычислить с точностью до» и под ней — курсор формульного редактора. После набора формульным редактором числа n нажимается *Enter*; тогда n переносится в конец указанной выше строки, и после него добавляется «знаков после запятой:». Задача имеет цели «упростить», «числоценка n».

3. «Найти приближенное значение». Здесь определяется приближенное значение константного выражения без каких-либо гарантий относительно числа точных знаков. Для вычислений используется математический сопроцессор; числа представляются в формате с плавающей запятой и с двойной точностью. Задача имеет цели «числзначение» и «выч».

Подраздел «Найти значения неизвестных» содержит следующие пункты (все они относятся к задачам на описание).

1. «Получить полное явное описание значений неизвестных». Это целевая установка для обычных задач «с неизвестными», например для решения систем уравнений и неравенств из элементарной алгебры. Для геометрических задач на вычисление и задач на решение дифференциальных уравнений используются другие целевые установки (см. ниже). В целевую установку входят цели «полный», «явное», «прямойответ», «одз», «упростить», «неизвестные $x_1 \ldots x_n$». После выбора целевой установки на экране прорисовывается строка «Найти», под которой формульным редактором вводятся неизвестные задачи — переменные, отделенные друг от друга запятыми. Затем нажимается клавиша *Enter*.

2. «Найти пример значений неизвестных». Это задачи на нахождение конкретного примера значений неизвестных, при которых истинны условия. Они имеют цели «полный», «пример», «прямойответ», «одз», «упростить», «неизвестные $x_1 \ldots x_n$». После выбора целевой установки на экране прорисовывается строка «Найти пример для». Далее формульным редактором перечисляются неизвестные и нажимается клавиша *Enter*.

3. «Выразить значения неизвестных через заданные параметры». Это задачи на вычисление, в которых требуется выразить значения неизвестных (уже явно выраженных через некоторые вспомогательные переменные посылок) через заданные известные величины. Такие задачи на описание наиболее часто встречающиеся в элементарной и аналитической геометрии, в физике и т. п. Они имеют целевую установку «полный», «явное», «прямойответ», «одз», «упростить», «известно $a_1 \ldots a_m$», «неизвестные $x_1 \ldots x_n$». Здесь x_1, …, x_n — неизвестные, a_1, …, a_m — известные параметры. Возможен случай $m = 0$ — если искомые значения неизвестных суть константы; в этом случае соответствующая цель состоит из логического символа «известно». После выбора целевой установки на экране прорисовывается строка «Выразить», и под ней формульным редактором набираются неизвестные (через запятую). После этого неизвестные

автоматически переносятся в конец указанной строки, к ней добавляется слово «через» и формульным редактором под строкой перечисляются известные параметры. После нажатия клавиши *Enter* (которое происходит сразу же в случае $m = 0$) известные параметры перерисовываются в конце строки. Если этих параметров не было, то вся строка перерисовывается виде «Найти значения $x_1, \ldots, x_n$».

Полное описание всех подразделов можно найти в [8, п. 3.4.3].

Кроме режима ввода целевой установки с помощью оглавления типов целей можно набирать эту установку непосредственно текстовым редактором. Для входа в режим набора установки текстовым редактором служит сочетание клавиш *Ctrl* + «ц». Сначала набирается тип задачи («преобразовать», «описать») и далее перечисляются в скобочной записи цели. Между целями оставляются пробелы; запятые не используются. Если цель имела вид набора $(fa_1 \ldots a_n)$, то она вводится как $f(a_1 \ldots a_n)$.

Чтобы изменить ранее введенную целевую установку, используются те же клавиши «ц» и *Ctrl* + «ц», что и для ее изначального создания. При этом следует помнить, что если задача, для которой изменяется целевая установка, не является последней в текущем списке задач, то предварительно ее следует выделить.

Можно просматривать оглавление типов целевых установок независимо от процесса ввода либо изменения задачи. Для входа в это оглавление достаточно нажать (находясь в просмотре списка задач) клавишу «Ц». Обычно данная возможность применяется для редактирования оглавления типов целевых установок.

После ввода целевой установки задачи выполняется ввод ее условия либо (в случае задачи на описание) нескольких условий. На этом процесс создания новой задачи завершается.

Если нужно сохранить текстовую версию формулировки задачи, нажимается клавиша «н», и далее эта версия вводится с помощью текст-формульного редактора. Подробнее об интерфейсе текст-формульного редактора см. [8, гл. 6].

Формульный редактор. Приведем краткое описание формульного редактора, с помощью которого происходит ввод условий и посылок задач. При входе в формульный редактор появляется прямоугольник курсора (коричневатого цвета). Этот курсор, в отличие от курсора текстового редактора, нельзя перемещать клавишами перемещений курсора — его положение зафиксировано и соответствует текущему вводимому символу формулы. Ситуация при использовании формульного редактора «линейная» — можно либо ввести очередной символ либо отменить последний введенный символ нажатием клавиши *Backspace*.

В процессе ввода формулы происходят автоматические масштабирование, «центровка» и перенесение на новую строку элементов изображения. Для завершения ввода формулы нажимается клавиша

Enter. Эта же клавиша служит для указания на завершение набора отдельных фрагментов формулы — дробей, радикалов, интегралов и т. п. Применение ее в таких случаях должно быть аккуратным, так как повторное нажатие приведет к преждевременному обрыву набора формулы (впрочем, для возвращения в редактирование терма задачи достаточно выделить этот терм и нажать «ф»). Чтобы отменить начатое редактирование формулы и выйти из формульного редактора, нажимается клавиша *Esc*.

Заметим, что если формула заведомо не набрана до конца либо набрана с принципиальными ошибками, то формульный редактор игнорирует завершающие нажатия клавиши *Enter*. В этом случае следует либо довести набор формулы до конца, либо вернуться (*Backspace*) к ошибке и исправить ее, либо вообще повторно набрать всю формулу целиком.

При входе в формульный редактор автоматически осуществляется переход в латинский режим ввода символов с клавиатуры; при выходе — автоматическое возвращение в кириллицу. Несмотря на латинский режим, приводимые далее коды клавиш и сочетаний клавиш, используемые в формульном редакторе, часто даются через кириллические обозначения клавиш (никакого переключения режима клавиатуры при их использовании не предполагается — они приводятся в таком виде просто для удобства запоминания). В тех случаях, когда возможна путаница, явно указывается на использование кириллицы (в скобках ставится пометка «рус.»).

Заметим, что в любых режимах логической системы для ручного перехода в латинский режим клавиатуры достаточно нажать клавишу *F12*, а для перехода в режим кириллицы — клавишу *F11*. Однако использовать эти возможности рекомендуется только при работе в текстовом редакторе. Следует помнить, что все управляющие клавиши и сочетания клавиш в логической системе ориентированы на кириллический режим ввода, так что переход к латинскому режиму будет означать просто отключение соответствующих управляющих воздействий.

Переменные в формульном редакторе набираются либо без индексов (малые и большие латинские буквы), либо с числовыми индексами. Для набора индекса после ввода переменной нажимается клавиша «курсор вниз», и далее вводится индекс. По окончании ввода индекса нажимается клавиша *Enter*.

Некоторые логические символы обозначаются в «стандартной» математической записи теми же словами и словосочетаниями, что и во внутренней скобочной записи. Это относится в первую очередь к понятиям, для которых в математике не предусмотрено специальной символики. Разумеется, по мере обучения решателя доля таких понятий постоянно увеличивается. В особых случаях, для наиболее часто встречающихся понятий, вводятся специальные клавиши

либо комбинации клавиш (обычно две последовательно нажимаемые клавиши). В остальных случаях для ввода логического символа, изображаемого словом либо словосочетанием, нажимается сочетание клавиш *Ctrl* + *Enter*, переводящее в режим побуквенного набора данного символа текстовым редактором. По окончании набора нажимается клавиша *Enter*, возвращающая в режим формульного редактора.

Переход на новую строку выполняется формульным редактором автоматически. Однако при использовании указанного выше способа ввода логического символа через *Ctrl* + *Enter* может оказаться, что для набора этого символа текстовым редактором на текущей строке места недостаточно. Тогда перед набором следует обеспечить переход к новой строке вручную, нажав сочетание клавиш *Ctrl* + «курсор вниз».

Подробное перечисление поддерживаемых формульным редактором логических символов и способов их ввода приведено в справочнике по логической системе. Этот справочник доступен, например, из главного меню (по клавише *F1*); в процессе набора формулы также можно перейти к нему, нажав клавишу *F1*. После нажатия возникает оглавление справочника. В этом оглавлении следует найти корневой раздел и в нем выбрать пункт «Формульный редактор». Для возвращения в набор формулы из просмотра справочника нажимается клавиша *End* (из оглавления справочника — однократно, из концевого текста справочника — дважды). Здесь мы приводим достаточно полное перечисление возможностей формульного редактора, опуская только те, которые используются крайне редко.

1. *Общелогические символы.* Отрицание $\neg A$ утверждения A вводится путем последовательного нажатия клавиш «n», «o» и последующего набора A. Символы «и», «или» вводятся, соответственно, как «&» и *Ctrl* + «v». Символ «если-то», встречающийся только под квантором общности, изображается стрелкой и вводится нажатием клавиши «курсор вправо». Символ «эквивалентно» обозначается двусторонней стрелкой и вводится нажатием клавиши «курсор влево».

Для ввода кванторной записи «$\forall x_1 \ldots x_k(A_1 \& \ldots \& A_n \to A_0)$» сначала дважды нажимается большая латинская буква «A», и появляется знак квантора общности. Затем перечисляются (без каких-либо пробелов) переменные $x_1, \ldots, x_k$ кванторной приставки; нажимается *Enter* (здесь появляется открывающая скобка), после чего вводится конъюнкция $A_1 \& \ldots \& A_n$. Далее нажимается «курсор вправо» (появляется стрелка для «если-то»), вводится утверждение A_0 и в конце ставится закрывающая скобка.

Для ввода кванторной записи «$\exists x_1 \ldots x_k(A)$» сначала дважды нажимается большая латинская буква «E» и появляется знак квантора существования. Затем набирается кванторная приставка $x_1 \ldots x_k$;

нажимается *Enter* (появляется открывающая скобка); вводится утверждение *A* и ставится закрывающая скобка.

Возможно использование квантора «существует и единственно». После знака квантора существования здесь идет восклицательный знак. Для появления такого обозначения следует вместо двукратного нажатия «E» нажать последовательно клавиши «E» и «T».

Равенство вводится обычным образом. Логические константы «истина» и «ложь» прорисовываются как «truth» и «false». Они вводятся, соответственно, двукратным нажатием клавиши «t» либо «f».

Условное выражение (*A* при *P*, иначе *B*), принимающее значение *A*, если истинно условие *P*, и значение *B* в противном случае, набирается следующим образом. Ставится открывающая скобка; вводится выражение *A*; нажимается *Ctrl* + «e» («e» — рус., от слова «если»); вводится *P*; нажимается *Ctrl* + «и» («и» — от слова «иначе»); вводится *B* и ставится закрывающая скобка.

2. *Арифметические операции и отношения*. Числовые константы набираются обычным образом; в случае десятичных дробей используется точка. Операции «плюс» (произвольное число слагаемых, большее или равное двух) и «минус» вводятся клавишами «+» и «-».

Умножение вводится нажатием клавиши «звездочка», причем в случае однократного нажатия этой клавиши (перед очередным сомножителем) фактической прорисовки чего-либо на экране не происходит. Если нужно убедиться в том, что нажатие клавиши и ввод операции умножения состоялись, то клавиша «звездочка» должна быть нажата еще дважды — в этом случае для умножения будет прорисована точка. Следует особенно аккуратно вводить произведения, часть сомножителей которых заключена в скобки. Если пропустить нажатие клавиши «звездочка» после *A*, то произведение $A(B + C)$ будет воспринято как значение функции *A* в точке $B + C$.

Дробное выражение A/B вводится следующим образом. Сначала нажимается клавиша «двоеточие» — после этого прорисовывается красным цветом горизонтальная черта дроби, над которой размещается курсор. Затем вводится числитель *A*. В процессе ввода горизонтальная черта дроби удлиняется автоматически; если вводится многоэтажное выражение, то для обеспечения необходимого места происходит автоматическая коррекция размеров и размещения всей ранее введенной части формулы. После ввода числителя нажимается клавиша *Enter* — тогда курсор перемещается в начало знаменателя — и осуществляется ввод знаменателя *B*. Наконец, после ввода знаменателя нажимается клавиша *Enter*, что завершает ввод дроби. При этом горизонтальная черта дроби из красной становится черной и происходит автоматическая центровка числителя и знаменателя для получения симметричной записи.

Степенное выражение A^B вводится следующим образом. Сначала набирается выражение A (если оно само является результатом применения более чем одноместной операции, то обязательно заключается в скобки). Затем нажимается клавиша «курсор вверх» — курсор поднимается вверх для прорисовки показателя степени. После ввода выражения B (его необязательно заключать в скобки) нажимается *Enter* — курсор опускается обратно, и ввод степени считается законченным.

Отношения «меньше», «больше» вводятся с помощью клавиш «<», «>»; отношения «меньшеилиравно», «большеилиравно» — с помощью клавиш «[», «]». Для ввода утверждения «A — число» сначала набирается выражение A, затем нажимаются клавиши «/» и «ч».

Конечная сумма $\sum_{i=a}^{b} f(i)$ вводится следующим образом. Сначала нажимается сочетание клавиш *Ctrl* + «s» — появляется большая буква «сигма» малинового цвета, причем курсор находится под этой буквой. Здесь набирается $i = a$ и нажимается клавиша *Enter*. Тогда курсор переводится в положение над буквой «сигма», где набирается выражение b и снова нажимается клавиша *Enter*. После этого курсор оказывается справа от буквы «сигма», где набирается выражение $f(i)$. По окончании набора нажимается клавиша *Enter*, в результате чего буква «сигма» перекрашивается в черный цвет — набор суммы на этом закончен.

Если требуется задать множество допустимых значений индекса суммирования косвенным образом, то используется запись $\sum_{i,P(i)} f(i)$. Ввод ее происходит аналогично предыдущему, но в положении курсора над «сигмой» сразу нажимается клавиша *Enter*.

Конечные произведения $\prod_{i=a}^{b} f(i)$, $\prod_{i,P(i)} f(i)$ вводятся совершенно так же, как конечные суммы, но вместо сочетания клавиш *Ctrl* + «s» нажимается сочетание *Ctrl* + «p».

3. *Элементарные функции*. Для ввода квадратного корня $\sqrt{A}$ последовательно нажимаются клавиши «s», «r» (появляется радикал малинового цвета), вводится выражение A и нажимается *Enter* — радикал перекрашивается в черный цвет. Если нужно ввести корень $\sqrt[n]{A}$, где n — целое от 3 до 9, то последовательно нажимаются клавиши «r», «t» (появляется малиновый радикал, над которым расположен курсор), «n» (курсор переводится под радикал) и далее — как для квадратного корня.

Для ввода максимума либо минимума выражений A_1, …, A_n последовательно нажимаются, соответственно, латинские клавиши «m», «a» либо «m», «i» и далее в скобках перечисляются указанные выражения. Для ввода модуля выражения A сначала нажимается *Ctrl* + «m», затем вводится A, затем снова нажимается *Ctrl* + «m».

Выражение $\log_a b$ набирается следующим образом. Сначала последовательно нажимаются клавиши «l», «o». Затем набирается основание логарифма a. Далее нажимается *Enter* — курсор переводится в позицию справа от логарифма, и набирается выражение под логарифмом b. Если оно представляет собой многоместную операцию, то его обязательно следует заключить в скобки — иначе под логарифмом окажется лишь первый операнд этой операции. Это же замечание о скобках относится и ко всем приводимым ниже элементарным функциям. В случае натурального логарифма достаточно нажать клавиши «l», «n» и ввести выражение под логарифмом.

Экспонента $\exp(A)$ вводится последовательным нажатием клавиш «e», «x». Синус $\sin(A)$ вводится последовательным нажатием клавиш «s», «i». Заметим, что степень синуса вводится не совсем стандартным образом: необходимо сначала набрать все выражение $\sin(A)$ (для большей наглядности, хотя и необязательно, заключенное в скобки) и лишь затем нажать «курсор вверх», переходя к вводу показателя степени. Помещать показатель степени сразу же после «sin» нельзя. Это же замечание относится и к другим элементарным функциям.

Оставшиеся элементарные функции вводятся однотипным с синусом образом — последовательным нажатием двух либо трех латинских клавиш (в ряде случаев, как указано ниже, требуется вводить не малую, а большую букву). Мы просто перечислим для них эти пары либо тройки клавиш: косинус — «c», «o»; тангенс — «t», «g»; котангенс — «c», «t»; секанс — «s», «e»; косеканс — «c», «s»; арксинус — «a», «s»; арккосинус — «a», «c», «o»; арктангенс — «a», «t»; арккотангенс «a», «c», «t»; сигнум (минус единица для отрицательных, не определен в нуле, единица для положительных) — «s», «g»; Сигнум (0 для отрицательных, 1 для неотрицательных) — «S», «g»; гиперболический синус — «s», «h»; гиперболический косинус — «c», «h»; гиперболический тангенс — «t», «h»; гиперболический котангенс — «c», «T».

4. *Символьные константы*. Константа e («e») вводится двойным нажатием латинской клавиши «e» (важно выполнять эту операцию аккуратно, чтобы не получить вместо константы переменную e; по внешнему виду они несколько отличаются друг от друга). Константа π («пи») вводится двойным нажатием латинской клавиши «p». Константа $+\infty$ («плюсбеск») вводится двойным нажатием клавиши «i». Для получения константы $-\infty$ («минусбеск») перед константой «плюсбеск» помещается знак «минус» (хотя во внутреннем представлении «минусбеск» представлена отдельным логическим символом). Мнимая единица вводится двукратным нажатием клавиши «c».

5. *Операции и отношения для целых чисел*. Утверждения «A — целое», «A — натуральное», «A — рациональное», «A — четное» вводятся следующим образом: сначала набирается выражение A, затем

нажимается «/» и далее, соответственно, клавиша «ц» (целое) либо «н» (натуральное), либо «q» (рациональное), либо «е» (четное; лат.). Выражения «числитель(A)» и «знаменатель(A)» вводятся последовательным нажатием клавиш «ч», «и» либо, соответственно, «з», «н» (далее набирается A, при необходимости — в скобках).

Утверждение «$A|B$» (A делит B) вводится в следующей последовательности: сначала набирается A, затем нажимается *Ctrl* + «д», затем B. Выражения «нод(A, B)» и «нок(A, B)» (наибольший общий делитель и наименьшее общее кратное) вводятся, соответственно, последовательным нажатием клавиш «н», «д» либо «н», «к» и дальнейшим набором (в скобках и через запятую выражений A, B. Аналогично вводятся утверждения «простое(A)» (двойное нажатие «п») и «взаимнопросты(A, B)» (клавиши «в», «п»).

Целая часть $[A]$ выражения A набирается следующим образом: сначала нажимается сочетание клавиш *Ctrl* + «е», что приводит к появлению левой квадратной скобки. Затем вводится A и снова нажимается *Ctrl* + «е» — для правой квадратной скобки.

Для вычета $A \pmod B$ сначала набирается выражение A, затем открывающая скобка, затем нажимаются клавиши «м», «д», затем набирается B и в конце — закрывающая скобка.

Число сочетаний из n по k вводится следующим образом. Сначала нажимается *Ctrl* + «с», что приводит к появлению большой буквы «C» малинового цвета; курсор находится справа в нижней части этой буквы. Затем набирается n и нажимается *Enter* — курсор перемещается в верхнюю часть справа от буквы «C». Далее набирается k и снова нажимается *Enter* — буква перекрашивается в черный цвет и ввод числа сочетаний завершается. При необходимости в процессе ввода вертикальные размеры буквы «C» автоматически увеличиваются.

Факториал $n!$ вводится последовательным набором n (при необходимости — в скобках) и нажатием *Ctrl* + «f».

6. *Алгебра множеств*. Утверждение «A — множество» вводится последовательным набором A, нажатием «/» и «s». Символ $\varnothing$ пустого множества вводится последовательным нажатием клавиш «e», «m». Символ $\cup$ объединения множеств вводится последовательным нажатием клавиш *Space*, «u», «s»; символ $\cap$ пересечения множеств — нажатием клавиш *Space*, «i», «s». Для ввода символа $\setminus$ разности множеств нажимается клавиша «\». Символ $\in$ принадлежности множеству вводится последовательным нажатием клавиш *Space*, «b», «e»; символ $\subset$ включения множеств — нажатием клавиш *Space*, «s», «u». Символ $\times$ прямого произведения множеств вводится нажатием клавиш *Space*, «p», «s».

Конечное множество $a_1, \ldots, a_n$, заданное перечислением своих элементов, вводится путем последовательного набора этих элементов (через запятую) внутри фигурных скобок.

Конечное объединение $\bigcup_{i=a}^{b} f(i)$ вводится следующим образом. Сначала нажимается *Ctrl* + «u» — появляется большой знак «объединение» малинового цвета, причем курсор находится под ним. Здесь набирается $i = a$ и нажимается *Enter*. Тогда курсор переводится в положение над «объединением», где набирается выражение b и снова нажимается *Enter*. После этого курсор оказывается справа от «объединения», где набирается выражение $f(i)$. По окончании набора нажимается *Enter*, в результате чего знак «объединение» перекрашивается в черный цвет — его набор закончен. Если требуется задать множество допустимых значений индекса объединения косвенным образом, то используется запись $\bigcup_{i,P(i)} f(i)$. Ввод ее происходит аналогично предыдущему, но в положении курсора над «объединением» сразу нажимается *Enter*.

Конечные пересечения $\bigcap_{i=a}^{b} f(i)$, $\bigcap_{i,P(i)} f(i)$ вводятся совершенно так же, как конечные объединения, но вместо сочетания *Ctrl* + «u» нажимается сочетание клавиш *Ctrl* + «x».

Класс $set_x P(x)$ всех объектов (если x — список переменных, то наборов) x, удовлетворяющих условию $P(x)$, вводится следующим образом. Сначала дважды нажимается клавиша «S» — появляется «set». Затем вводится переменная либо список переменных x; нажимается *Enter* и набирается условие $P(x)$.

Мощность $card(A)$ множества A вводится нажатием клавиш «c», «a» и набором выражения A. Заголовки утверждений «конечное(A)», «непересек(A, B) «семействомножеств(A)», «разбиение(A, B)» вводятся, соответственно, последовательными нажатиями пар клавиш: «к», «о»; «н», «п»; «С», «м»; «р», «а» (все — рус.). Заголовок выражения «подмножества(A)» вводится последовательным нажатием клавиш «П», «м».

Мощности счетного множества и континуума вводятся, соответственно, нажатиями клавиш «с», «ч» и «к», «м». Символ пустого слова вводится нажатием клавиши *Ctrl* + «щ».

Утверждения «убывмножества(A)», «возрастмножества(A)» о том, что последовательность множеств A является убывающей либо возрастающей по включению, вводятся последовательными нажатиями клавиш «У», «М» и «В», «М» (рус.).

7. *Числовые множества*. Для ввода промежутка числовой оси с концами A, B (они могут являться символами бесконечности) сначала нажимается *Ctrl* + «(» (если конец A не относится к промежутку) либо *Ctrl* + «[» (если этот конец относится к промежутку). Затем набираются отделенные запятой выражения A, B. Если конец B не относится к промежутку, то далее нажимается *Ctrl* + «)», иначе — *Ctrl* + «]».

Множество вещественных чисел вводится двойным нажатием клавиши «R»; множество рациональных чисел — двойным нажатием клавиши «*Q*»; множество целых чисел — двойным нажатием клавиши «*Z*»; множество натуральных чисел — двойным нажатием «N»; множество целых неотрицательных чисел — последовательным нажатием «Ц», «Н» (рус.).

Конечный отрезок {*A*, …, *B*} натурального ряда, начинающийся с *A* и кончающийся *B*, набирается следующим образом: левая фигурная скобка, выражение *A*, запятая, *Ctrl* + «.», запятая, выражение *B*, правая фигурная скобка

Точная нижняя грань inf *A* множества *A* вводится с помощью нажатия клавиш «I», «n»; точная верхняя грань sup *A* — с помощью «S», «u».

Утверждения «нижняягрань(*x*, *A*)» (*x* — нестрогая нижняя грань числового множества *A*), «Нижняягрань(*x*, *A*)» (строгая нижняя грань), «верхняягрань (*x*, *A*)» (нестрогая верхняя грань), «Верхняягрань(*x*, *A*)» (строгая верхняя грань), «наибольший(*x*, *A*)» (*x* — наибольший элемент числового множества *A*), «наименьший(*x*, *A*)» вводятся, соответственно, с помощью последовательных нажатий пар клавиш: «н», «г»; «Н», «г»; «в», «г»; «В», «г»; «Н», «О»; «Н», «Е» (все — рус.).

Утверждения «огрснизу(*A*)», «огрсверху(*A*)» об ограниченности числового множества *A* снизу либо сверху вводятся с помощью нажатий пар клавиш «г», «н» и «г», «в».

Выражения «внутренность(*A*)», «граница(*A*)», «замыкание(*A*)» вводятся, соответственно, при помощи нажатий пар клавиш «в», «н»; «Г», «р» (рус.); «З», «З».

8. *Простейшие обозначения, связанные с функциями*. Утверждение «*A* — функция» вводится последовательным набором *A*, нажатием «/» и «ф». Область определения Dom(*f*) функции *f* вводится с помощью последовательного нажатия клавиш «d», «o». Множество значений Val(*f*) функции *f* вводится с помощью последовательного нажатия клавиш «v», «a». Для ввода значения *f*(*a*) функции *f* в точке *a* сначала набирается обозначение функции *f* (если оно не односимвольное, то заключается в скобки), затем левая скобка, *a* и правая скобка. Выражения «образ(*f*, *A*)» (образ множества *A*), «прообраз(*f*, *A*)» (прообраз множества *A*) и «слой(*f*, *a*)» (полный прообраз элемента *a*) набираются при помощи последовательных нажатий пар клавиш «о», «м»; «п», «м»; «с», «л» (все — рус.). Утверждение «взаимнооднозначно(*f*)» (*f* инъективно) вводится при помощи последовательного нажатия клавиш «в», «о» (рус.). Множество корней числовой функции *f* на множестве *A* обозначается roots(*f*, *A*) и вводится при помощи последовательного нажатия клавиш «r», «o».

Функция, значения которой определяются выражением *t*, а условие на принадлежность аргумента *x* (этот аргумент может быть

как единственной переменной, так и набором переменных) области определения есть $P(x)$, обозначается $\lambda_x(t, P(x))$. Для ввода ее сначала дважды нажимается клавиша «L» — прорисовывается «лямбда», причем курсор располагается справа от нее и чуть ниже. Затем вводится x (переменная либо группа переменных, без запятых), нажимается *Enter* (курсор перемещается вверх на уровень «лямбды») и набираются в скобках t, $P(x)$.

Утверждение «Отображение(f, A, B)» (f есть отображение A в B) вводится при помощи двукратного нажатия клавиши «о» (рус.).

Функция «конст(A, b)», определенная на множестве A и принимающая во всех его точках значение b, вводится при помощи двукратного нажатия «к» (рус.). Функция «$a \to b$», определенная на одноэлементном множестве, состоящем из элемента a, и принимающая на этом элементе значение b, вводится следующим образом: сначала набирается a, затем *Ctrl* + «курсор вправо», затем b. Тождественная функция «тождфунк(A)» с областью определения A вводится при помощи нажатия клавиш «т», «ф».

Посредством выражения «таблица($A_1 \ldots A_n$)» обозначается функция, график которой есть объединение графиков функций $A_1, \ldots, A_n$, принимающих на пересечениях своих областей определения одинаковые значения. Символ «таблица» вводится последовательным нажатием клавиш «т», «а» (рус.).

Выражения «сужение(f, A)» (сужение функции f на множество A) и «доопределение(f, A, b)» (доопределение функции f на тех точках множества A, где она еще не определена, значением b) вводятся при помощи последовательных нажатий клавиш «с», «у» и «д», «о» (все — рус.). Выражение «обрфункция(f)» (функция, обратная к взаимно однозначной функции f) вводится при помощи последовательного нажатия «о» (рус.), «ф».

Выражение «числопеременных(f)» вводится при помощи клавиш «ч», «п»; выражение «подфункция(f, A, B)» (функция, полученная из f подстановкой элементов набора B вместо переменных, номера которых образуют набор A; нумерация начинается с единицы) вводится при помощи клавиш «п», «Ф». Утверждение «существперем(i, f)» (i-я переменная функции f является существенной) вводится при помощи клавиш «С», «Y».

9. *Планиметрия*. В планиметрии фигуры и их числовые характеристики обозначаются только через соответствующие опорные точки («прямая(AB)», «треугольник (ABC)» и т. п.); использование вспомогательных переменных для обозначения самих фигур («прямая A»; «треугольник B» и т. п.) в приемах решателя не предусмотрено. В стереометрии, наоборот, тела обозначаются не с помощью их опорных точек, а с помощью вспомогательных переменных («куб(A)», «пирамида (B)» и т. п.).

Утверждение «*A* — точка» вводится нажатием, после набора *A*, клавиш «\» и «р». Обычно такие утверждения вообще не вводятся вручную, а создаются решателем самостоятельно в начале решения задачи.

Выражения «прямая(*AB*)», «отрезок(*AB*)» «интервал(*AB*)», «луч(*AB*)», «обратныйлуч(*AB*)» (луч с началом в точке *A*, противоположный лучу *AB*) вводятся последовательными нажатиями пар клавиш «п», «р»; «о», «т»; «и», «н»; «л», «у»; «о», «л» (все — рус.). Заметим, что между *A* и *B* здесь не ставится запятая, в отличие от обычных многоместных операций и отношений формульного редактора. Такие исключительные случаи перечисления операндов без запятой используются только в планиметрии, причем лишь для некоторых (явно указываемых далее) логических символов. В качестве *A*, *B* могут быть использованы либо буквенные переменные, либо переменные с индексом. Если имеется хотя бы одна переменная с индексом, то между операндами необходим разделитель — клавиша «*» («звездочка», при ее нажатии на экране ничего не меняется, но становится возможен ввод следующего операнда).

Расстояние *l*(*AB*) между точками *A*, *B* вводится при помощи двукратного нажатия клавиши «l». Величина ∠*ABC* угла *ABC* вводится при помощи нажатия клавиш «у», «г». В обоих случаях запятые между операндами не ставятся. Если нужно обозначить не величину угла, а множество точек плоскости, лежащих внутри него (включая граничные точки), то используется обозначение «Угол(*ABC*)», вводимое клавишами «У», «г».

Выражения «расстдопрямой(*A*, *B*)» (расстояние от точки *A* до прямой *B*), «расстмежду(*A*, *B*)» (расстояние между двумя множествами точек *A*, *B*) вводятся при помощи клавиш «Р», «П»; «Р», «Ы» (все — рус.). Здесь уже нужна запятая между операндами.

Утверждение «биссектриса(*ABCD*)» (луч *BD* есть биссектриса угла *ABC*) вводится двойным нажатием клавиши «и». Запятая между операндами не ставится.

Утверждения «*a* || *b*» (прямые либо плоскости *a*, *b* параллельны) и «*a* ⊥ *b*» (прямые либо плоскости *a*, *b* перпендикулярны) вводятся, соответственно, при помощи клавиш *Ctrl* + «=» и *Ctrl* + «т».

Утверждения «однасторона(*A*, *B*, *a*)» (точки *A*, *B* лежат по одну сторону от прямой либо плоскости *a*, возможно, попадая на *a*), «разныестороны(*A*, *B*, *a*)» (здесь тоже допускается попадание точек на *a*), «симметричны(*A*, *B*, *a*)» (точки *A*, *B* расположены симметрично относительно точки, прямой либо плоскости *a*) вводятся при помощи клавиш «О», «С»; «Р», «С»; «с», «и» (все — рус.). Выражения «полоса(*A*, *B*)» (полоса между параллельными прямыми *A*, *B*), «полуплоскость(*A*, *B*)» (полуплоскость, ограниченная прямой *A* и содержащая точку *B*, не лежащую на *A*), «обрполуплоскость(*A*, *B*)»

(полуплоскость, ограниченная прямой *A* и не содержащая точку *B*, не лежащую на прямой *A*) вводятся при помощи нажатий клавиш «п», «с»; «п», «П»; «о», «П» (все — рус.).

Для уточнения того, что вводимая задача относится к планиметрии (все рассматриваемые в ней точки лежат в общей плоскости), используется вспомогательная посылка «планиметрия». Обычно она вводится решателем автоматически после набора задачи (перед ее решением); вручную ее можно ввести последовательным нажатием клавиш «п», «и».

Утверждения «△(*ABC*)» (точки *A*, *B*, *C* образуют вершины треугольника), «параллелограмм(*ABCD*)» (точки *A*, *B*, *C*, *D*, взятые в данной последовательности, образуют вершины параллелограмма); «ромб(*ABCD*)»; «прямоугольник(*ABCD*), «квадрат(*ABCD*)», «четырехугольник(*ABCD*)» (точки *A*, *B*, *C*, *D* образуют вершины выпуклого четырехугольника) вводятся последовательными нажатиями пар клавиш «т», «р»; «п», «а»; «р», «о»; «п», «р»; «к», «в»; «ч», «е» (все — рус.). Запятые между операндами не ставятся.

Утверждение «трапеция(*ABCD*)» (точки *A*, *B*, *C*, *D* образуют вершины трапеции, углы при большем основании которой — не тупые, причем точки *A*, *D* лежат на большем основании) вводится при помощи клавиш «т», «п». Утверждение «Трапеция(*ABCD*)» (то же, но углы при большем основании — любые, а точки *A*, *D* лежат на основании, которое не обязательно большее) вводится при помощи клавиш «Т», «п».

Утверждения «многоугольник(*A*)» (*A* — набор вершин простой замкнутой ломаной), «правмногоугольник(*A*)» (*A* — набор вершин правильного многоугольника) вводятся последовательными нажатиями пар клавиш «м», «н» и «п», «й». Множество точек внутри многоугольника (включая границу), определяемого набором вершин *A*, обозначается «фигура(*A*)». Точки набора *A* во всех перечисленных случаях отделяются друг от друга запятыми. Выражение «Фигура {*A*, *B*, …, *C*}» (множество точек, ограниченное составной границей с частями *A*, *B*, …, *C*) вводится при помощи клавиш «ϑ», «и».

Утверждения «Медиана(*ABCD*)», «Высота(*ABCD*)», «Биссектреуг(*BACD*)» (точка *D* является основанием, соответственно, медианы, высоты либо биссектрисы треугольника *ABC*, проведенной из вершины *A*) вводятся при помощи пар клавиш «М», «Е»; «В», «Ы»; «И», «Т» (все — рус.).

Выражения «окружность(*AB*)», «круг(*AB*)» (окружность и круг с центром в точке *A* и точкой на окружности *B*) вводятся нажатием пар клавиш «о», «к» и «к», «р» (все — рус.). Выражения «дуга(*ABC*)» (меньшая из дуг окружности с центром в точке *A* и концами *B*, *C*); «большаядуга (*ABC*)» (большая из указанных дуг); «дугаугла(*ABC*)» (дуга, на которую опирается вписанный угол *ABC*); «сектор(*ABC*)» (сектор окружности с центром в точке *A* и концами дуги *B*, *C*; берет-

ся меньшая из дуг); «сегмент(*ABC*)» вводятся при помощи нажатий пар клавиш «д», «у»; «д», «У»; «д», «в»; «С», «е»; «с», «г» (все — рус.).

Выражение «кольцо(*ABC*)» (кольцо с центром в *A*, внутренняя окружность которого проходит через *B*, а внешняя — через *C*) вводится при помощи «к», «О» (все — рус.).

Площадь *S*(*A*) плоской фигуры *A* (например, *A* может иметь вид «фигура(*B*)», «круг(*B C*)», «сектор(*BCD*)» и т. п.) вводится при помощи двукратного нажатия клавиши «S». Выражение «периметр(*A*)» (*A* обычно имеет вид «фигура(*B*)») вводится при помощи последовательного нажатия клавиш «п», «е» (рус.). Выражение «длина(*A*)» (длина кривой *A*) вводится при помощи «д», «л».

Утверждение «центр(*A*, *B*)» (точка *A* является центром фигуры *B*) вводится при помощи «ц», «е» (рус.).

Утверждение «*A* — касательная к *B*» вводится набором *A*, нажатием *Ctrl* + «q» и набором *B*. Утверждения «внешкасательная(*A*, *B*, *C*)» (прямая *A* является внешней касательной к окружностям *B*, *C*); «внутркасательная(*A*, *B*, *C*)» вводятся нажатиями «Ш», «к» и «У», «к» соответственно.

Утверждения «окружность(*AB*) вписана в фигура($C_1 \ldots C_n$)»; «окружность(*AB*) описана около фигура($C_1 \ldots C_n$)» вводятся нажатием сочетаний клавиш *Ctrl* + «э» и *Ctrl* + «ф» соответственно (предварительно вводится «окружность(*AB*)»).

Утверждения «внешкасаются(*A*, *B*)» (внешнее касание окружностей) и «внутркасаются(*A*, *B*)» вводятся последовательными нажатиями пар клавиш «ш», «к» и «у», «к».

Утверждения «выпукло(*A*)», «осьсимметрии(*A*, *B*)» вводятся при помощи клавиш «в», «ы»; «С», «и» (все — рус.).

Утверждения «$A \sim B$» (подобие фигур) и «$A \equiv B$» (конгруэнтность фигур) вводятся при помощи сочетаний клавиш *Ctrl* + «п» и *Ctrl* + «к».

Текстовый редактор. В решателе используется интерфейс текстового редактора, несколько отличающийся от стандартных версий. Этот редактор используется главным образом для работы с программами приемов, однако он может понадобиться и при вводе элементов задачи. Приведем краткое описание основных его функций.

Текстовый редактор позволяет работать только с текстами, целиком помещающимися на экране. Для работы с большими текстами использующие текстовый редактор процедуры имеют режим «перелистывания», определенный в описании их интерфейса.

При входе в текстовый редактор появляется прямоугольный курсор светло-коричневого цвета и автоматически устанавливается режим кириллицы; для перехода в режим латиницы нажимается клавиша *F12*, для возвращения в режим кириллицы — клавиша *F11*. Для ускорения перемещений курсора вводится режим увели-

ченного шага (5-кратное смещение по вертикали либо 8-кратное по горизонтали). Переключение между одношаговым режимом и ускоренным осуществляется нажатием клавиши *Tab*. Завершение редактирования происходит при нажатии клавиши *Enter*; отмена редактирования — при нажатии *Esc*.

При всех операциях с текстом соблюдается принцип: позиция, следующая за крайней правой позицией строки, есть начальная позиция следующей строки. При достижении правого края окна редактирования курсор автоматически переходит в начало следующей строки этого окна.

Для вставки пустой строки начиная с текущей позиции (все, что было правее этой позиции, смещается на одну строку вниз) используется клавиша *F7*. Для исключения одной строки начиная с текущей позиции (все, что было на следующей строке правее этой позиции, сдвигается вверх на одну строку) нажимается клавиша *F6*. Переход в начало либо в конец строки осуществляется, как обычно, с помощью клавиш *Home* и *End*.

Для вставки текста перед заданной позицией курсор перемещается к этой позиции и нажимается клавиша *Insert*. Курсор после этого становится голубым — включается режим вставки. В этом режиме каждый вводимый с клавиатуры символ вставляется непосредственно перед курсором (который сдвигается так, что расположен все время над исходным символом). По завершении вставки снова нажимается *Insert* — курсор перекрашивается в светло-коричневый цвет. Если во время вставки нажать *Backspace* — для отмены последнего введенного символа, то также происходит выход из режима вставки. В режиме вставки курсор не реагирует на нажатия клавиш сдвига курсора.

Для удаления фрагмента текста курсор сначала подводится к началу этого фрагмента (первый удаляемый символ) и нажимается клавиша *Delete*. Курсор перекрашивается в красный цвет. Фактически это лишь маркер начала удаляемой части, наложенный поверх курсора, а курсор (светло-коричневого цвета) обычным образом реагирует на клавиши сдвига. Далее курсор подводится к концу удаляемого фрагмента (первый сохраняемый символ) и снова нажимается *Delete* — выбранный фрагмент при этом удаляется. Если после выбора начала удаляемой части нажать *Backspace*, то удаление отменяется и красный маркер исчезает.

Возможна операция перенесения выбранной части текста на новое место. Для ее выполнения сначала курсор подводится к началу этой части и нажимается клавиша *F2*. Курсор перекрашивается в светло-зеленый цвет. Фактически это лишь наложенный поверх курсора маркер начала переносимой части. Затем курсор подводится к концу переносимой части (последний относящийся к ней символ) и снова нажимается *F2*. Появляется наложенный поверх

курсора светло-зеленый маркер конца. Наконец, курсор подводится к тому месту, начиная с которого должен быть расположен выбранный фрагмент, и снова нажимается *F2* — при этом происходит перенесение фрагмента на данное место. На каждом этапе нажатие *Backspace* удаляет последний введенный маркер.

Для получения подсказки о клавиатуре текстового редактора (если уже началось его использование) нажимается *F3*. После этого нажатие любой клавиши возвращает в прерванное редактирование. Для получения справочной информации, связанной с заданным логическим символом, нажимается клавиша *F4*. Возникает диалоговое окно, в котором вводится название нужного логического символа. По окончании ввода нажимается *Enter*, и появляется необходимая справочная информация. Выход из ее просмотра — по нажатию любой клавиши, кроме клавиш, используемых при просмотре такой информации (например, при нажатии клавиши *Space*). При этом восстанавливается прерванное редактирование.

Предусмотрен буфер для сохранения фрагмента текста. Практически во всех случаях содержимое этого буфера сохраняется при смене различных интерфейсов системы, и его можно использовать при любом повторном обращении к текстовому редактору. Для занесения фрагмента текста в буфер сначала курсор подводится к началу этого фрагмента и нажимается клавиша *PageDown*. Появляется синий маркер начала фрагмента, наложенный поверх курсора. Затем курсор подводится к концу фрагмента и снова нажимается *PageDown* — маркер начала исчезает, и буфер сохраняет выбранный фрагмент. Если до второго нажатия *PageDown* нажать *Backspace*, то маркер исчезает и операция отменяется. Для извлечения содержащегося в буфере фрагмента курсор подводится на ту позицию, начиная с которой должен быть вставлен текст, затем нажимается *PageUp*. Содержимое буфера при этом сохраняется.

Возможно изменение цвета фона (*Ctrl* + *F4*) и символов (*Ctrl* + *F5*). Эти цвета меняются циклическим образом (длина цикла — 16). Если внешняя процедура, обратившаяся к текстовому редактору, игнорирует цвета, то они при сохранении в файле будут утеряны.

Ряд специальных символов вводится с помощью сочетания буквенных клавиш и клавиши *Ctrl*. Так, символ ¬ вводится нажатием *Ctrl* + «n»; символ ∃ — нажатием *Ctrl* + «e» (лат.); символ ∀ — нажатием *Ctrl* + «a» (лат.). Заметим, что перечисленные символы используются только при наборе текстовым редактором обычных текстов; при наборе термов вместо них должны использоваться слова «не» , «существует», «длялюбого».

Если был набран некоторый текст, то при сохранении в файле автоматически отбрасывается его пустой конец. Если такой конец все же нужно сохранить в файле, то для обозначения его завершающей позиции используется символ ⊥, вводимый клавишей *F9*.

Каждому коду клавиатуры в языке программирования ЛОС сопоставлен некоторый логический символ. Если при программировании процедуры интерфейса нужно определить этот символ, то курсор текстового редактора подводится на ту позицию, начиная с которой должно быть записано название символа; нажимается клавиша *F8* и сразу же за этим нажимается та клавиша (либо сочетание клавиш), для которой нужно узнать кодирующий ее логический символ.

Термы вводятся текстовым редактором только в скобочной записи $F(A_1 \ldots A_m)$. Логический символ F вводится в виде слитно написанного слова; между операндами A_i необходимы разделители, причем разделителем служит пробел либо скобка. Запятые в качестве разделителей не допускаются (запятая сама является логическим символом). Числа вводятся как последовательности расположенных подряд цифр. В случае десятичных дробей используется запятая, а не точка, как в формульном редакторе. Знак «минус», как и прочие операции, применяется в сочетании со скобкой: «минус(*A*)». Для удобства работы со скобками предусмотрена операция перехода от скобки к парной ей скобке. Курсор помещается на одну из скобок, и нажимается клавиша *F1*, переводящая его на парную скобку.

Геометрический редактор. Геометрический редактор позволяет создавать простые чертежи, состоящие из некоторой системы обозначенных буквами точек, между которыми проведены линии (отрезки прямых либо окружности). Чертежи используются в задачах по геометрии и в описании геометрических приемов. Какой-либо функциональной нагрузки в работе решателя они пока не имеют, а используются лишь для обеспечения необходимой наглядности. Тем не менее при выполнении по ходу решения задачи дополнительных построений решатель сам пополняет чертеж новыми точками и линиями. Все, что нужно для этого сделать, — ввести в описании задачи чертеж, обозначения точек на котором согласованы с используемыми в условии задачи.

Если вводится новая геометрическая задача, для которой нужен чертеж, то лучше всего начинать ее набор прямо с чертежа. Для этого достаточно ввести бланк задачи (прорисована верхняя горизонтальная линия, обозначающая начало задачи) и нажать клавишу «ч». Если уже была набрана часть текста задачи, то вход в геометрический редактор осуществляется тем же самым нажатием клавиши «ч». В обоих случаях вся отведенная для задачи часть экрана занимается прямоугольником геометрического редактора, а набранная ранее часть текста задачи пропадает с экрана. Она будет восстановлена по окончании (либо отмене) создания чертежа и размещена под чертежом.

Можно попробовать создать чертеж с помощью автоматической процедуры, имеющейся в решателе. Для этого нажимается клави-

ша «Ч». Если до этого была введена ручная версия чертежа, то она будет заменена автоматической; восстановление предыдущей версии чертежа при этом достигается путем выделения чертежа (левой клавишей мыши) и нажатия *Ctrl + Del*. Если нужно вручную изменить ранее введенный чертеж задачи (в том числе автоматически введенный), то, как и при первоначальном вводе чертежа, нажимается «ч».

Заметим, что по окончании редактирования чертежа либо при отмене начатого редактирования автоматически выполняется прокрутка списка задач, так что в верхней части экрана оказывается начало обрабатываемой задачи. Указанным выше способом можно входить в редактирование чертежа только для последней задачи просматриваемого списка задач. Если же задача не последняя, то ее следует сначала выделить (нажатие правой клавиши мыши в любой точке поля задачи), и лишь затем нажать «ч».

Окно геометрического редактора представляет собой прямоугольник, ограниченный линией синего цвета. В правом верхнем углу отделена небольшая область, в которой появляется указатель текущего режима редактирования. Управление сменой режимов осуществляется с помощью меню геометрического редактора, появляющегося в верхней части экрана. Предусмотрены следующие режимы работы.

1. *Нейтральный режим*. Этот режим устанавливается при входе в геометрический редактор. У него прямоугольник указателя режима пустой. Переход из любого другого режим в этот режим обеспечивается нажатием клавиши *Space*. В нейтральном режиме можно выполнять следующие операции:

а) перемещение точки либо сопровождающего ее буквенного обозначения. Для этого точка выделяется нажатием левой клавиши мыши после подведения к ней курсора мыши. Затем можно сдвигать точку нажатием клавиш курсора (постоянное удержание клавиши нажатой приводит к медленному движению точки в выбранном направлении). Аналогично можно сдвигать обозначение выделенной точки нажатием сочетания клавиши *Ctrl* и клавиш курсоров. При движении точки корректируются все ведущие к ней отрезки прямых;

б) изменение обозначения точки. Сначала нажимается буквенная клавиша (быть может, сопровождаемая несколькими цифровыми при указании индекса) для нового обозначения, затем выделяется нужная точка (левой клавишей мыши) и нажимается «=». Как и в случае добавления новой точки (см. ниже), для ввода большой буквы на клавиатуре нажимается малая буква и наоборот. Старое обозначение заносится в накопитель обозначений точек и будет использовано при последующих вводе либо коррекции обозначения (если не сбросить накопитель с помощью клавиши *Delete*).

Эти операции можно применять и в других режимах (но не в режиме ввода новых точек, так как тогда каждая попытка выделить точку будет приводить к созданию новой точки).

2. *Режим ввода новых точек*. Вход в режим ввода новых точек осуществляется при нажатии клавиши *Insert* либо выборе мышью пункта «Точка» меню в верхней части экрана. В прямоугольнике указателя режима появляется слово «точка». Для ввода новой точки следует выбрать курсором мыши ее позицию и нажать левую клавишу мыши. В качестве обозначения точки каждый раз выбирается первая не использованная большая буква. Однако можно явно указывать обозначения точек, предварительно нажав последовательно некоторые буквенные клавиши. Тогда последующие нажатия левой клавиши мыши на выбранных позициях приведут к использованию для точек обозначений из данной последовательности. Так как обычно точки обозначаются большими буквами, предусмотрено автоматическое преобразование вводимых с клавиатуры малых букв в большие, а больших — в малые. Для сброса накопителя обозначений новых точек нажимается клавиша *Delete*.

Обозначения точек можно сопровождать индексами, для этого после нажатия буквенной клавиши нажимаются одна или несколько цифровых клавиш, образующих индекс.

Если нужно удалить ранее введенную точку, то к ней подводится курсор мыши и нажимается правая клавиша мыши. При этом обозначение удаленной точки сохраняется в накопителе обозначений и будет использовано (если не сбросить этот накопитель) при вводе новой точки.

3. *Режим сдвига точек*. Переход в этот режим осуществляется нажатием сочетания клавиш *Ctrl* + «с» (рус.) либо выбором мышью пункта «Сдвиг» меню в верхней части экрана. Указатель режима прорисовывает слово «сдвиг». Далее можно изменить положение любой из точек, не прибегая к сравнительно медленному перемещению ее клавишами курсора. Для этого сначала точка выделяется мышью (нажатие левой клавиши мыши после подведения к точке курсора мыши), затем курсор мыши подводится к новой позиции точки и снова нажимается левая клавиша мыши.

4. *Режим проведения отрезка*. Переход в режим осуществляется при нажатии сочетания клавиш *Ctrl* + «о» (рус.) либо выборе мышью пункта «Отрезок» меню в верхней части экрана. Указатель режима прорисовывает слово «отрезок». Для проведения отрезка между двумя точками сначала выделяется первая из них, затем курсор мыши подводится ко второй и нажимается левая клавиша мыши. Если нужно не ввести, а, наоборот, удалить ранее введенный отрезок, то после выделения первой точки отрезка на второй точке нажимается правая клавиша мыши.

5. *Режим проведения прямой*. Переход в режим осуществляется при нажатии сочетания клавиш *Ctrl* + «п» либо выборе мышью пункта «Прямая» меню в верхней части экрана. Указатель режима прорисовывает слово «прямая». Режим аналогичен режиму проведения отрезков, однако здесь отрезок продолжается вплоть до границ рамки. Режим практически не используется.

6. *Режим проведения перпендикуляра*. Переход в режим осуществляется при нажатии сочетания клавиш *Ctrl* + «т» либо выборе мышью пункта «Перпендикуляр» меню в верхней части экрана. Указатель режима прорисовывает слово «перпендикуляр». Возможны две операции: восставить перпендикуляр к прямой в заданной точке этой прямой либо опустить из заданной точки перпендикуляр на заданную прямую.

В первом случае левой клавишей мыши выбираются две различные точки на прямой (вторая из них — та, через которую пройдет перпендикуляр). Чтобы доопределить направление перпендикуляра и ввести на нем еще одну точку, курсором мыши выбирается какая-либо из уже введенных ранее точек и на ней нажимается правая клавиша мыши. Тогда перпендикуляр к прямой, проходящей через первые две точки, будет проведен из второй точки в направлении третьей точки, причем на нем будет введена ближайшая к третьей точке новая точка.

Во втором случае, как и в первом, сначала левой клавишей мыши выбираются две точки на прямой, к которой требуется провести перпендикуляр, и затем на третьей точке, из которой должен быть опущен перпендикуляр, нажимается левая клавиша мыши. При проведении перпендикуляра вводится новая точка — основание этого перпендикуляра.

7. *Режим проведения окружности*. Переход в режим осуществляется при нажатии сочетания клавиш *Ctrl* + «к» либо выборе мышью пункта «Окружность» меню в верхней части экрана. Указатель режима прорисовывает слово «окружность». Для проведения новой окружности сначала левой клавишей мыши выбирается ее центр, затем берется какая-либо точка, через которую должна пройти окружность, и снова нажимается левая клавиша мыши. Чтобы удалить введенную ранее окружность, сначала левой клавишей мыши выбирается ее центр, затем на той точке окружности, с помощью которой она вводилась, нажимается правая клавиша мыши. Последняя операция возможна лишь в том случае, если ранее введенная окружность не перемещалась по экрану (такое перемещение происходит при перемещении ее центра); иначе для удаления окружности нужно удалить ее центр).

8. *Режим параллельного перемещения всего изображения*. Переход в режим осуществляется при нажатии клавиши *Home* либо выборе мышью пункта «Перемещение» меню в верхней части экрана.

Указатель режима прорисовывает слово «перемещение». Для задания вектора перемещения сначала на некоторой позиции нажимается клавиша мыши, затем курсор мыши переносится на новую позицию и снова нажимается клавиша мыши.

9. *Режим изменения размеров рамки чертежа*. Переход в режим осуществляется при выборе мышью пункта «Рамка» меню в верхней части экрана. Возникает подменю, в котором перечисляются названия четырех сторон рамки («правыйкрай», «левыйкрай», «верхнийкрай» и «нижнийкрай»). После выбора в нем нужной стороны указатель режима повторяет название выбранной стороны. Для выбора нового положения этой стороны курсор мыши перемещается на требуемую позицию и нажимается левая клавиша мыши. Режим обычно используется для увеличения зоны чертежа, под которым размещены другие изображения. Это увеличение относится только к периоду редактирования; по завершении работы геометрического редактора нижняя граница чертежа выбирается автоматически по его крайним снизу точкам.

3.4. Общая схема функционирования решателя

3.4.1. Сканирование задачи

При обычном программировании для решения задач заданного типа разрабатывается один общий алгоритм, который гарантирует получение за конечное число шагов ответа в случае любой конкретной задачи этого типа. Такой алгоритм представляется в виде нескольких взаимодействующих между собой блоков, обеспечивающих циклический процесс постепенного упрощения параметров решаемой задачи (в частности, уменьшения оставшегося объема перебора для процедур переборного типа) — вплоть до получения окончательного ответа. Каждый из блоков несет здесь вполне определенную функциональную нагрузку, как правило, основанную на том или ином теоретическом утверждении, связанном с обрабатываемыми алгоритмом объектами. Его можно рассматривать как прием для достижения определенной текущей подцели. В свою очередь, отдельный блок сам определяется схемой своих подблоков и т. д. Здесь возникает иерархия уровней, в которой на каждом уровне несколько приемов (как правило, не более одного-двух десятков) оказываются связаны в вычислительный цикл, так что после выполнения очередного приема известно, к какому приему следует переходить дальше.

К сожалению, такого рода четкую программу действий удается составить лишь для достаточно узких классов задач. Хорошо известны результаты об алгоритмической неразрешимости различных общих проблем, и даже в случае наличия алгоритма он часто

оказывается практически неприемлемым по своей трудоемкости. Для преодоления возникающих здесь трудностей приходится резко увеличивать количество используемых приемов, чтобы охватить алгоритмическими возможностями если не все многообразие задач предметной области, то хотя бы возможно больший класс задач, встречающихся в реальных ситуациях. Эти приемы создаются уже не путем «теоретического» проектирования алгоритма, а извлекаются из последовательности конкретных обучающих задач и складываются в одну общую базу приемов.

Количество приемов, используемых для решения задач в предметной области, обычно существенно больше, чем в алгоритмической процедуре указанного выше «блочного» типа, — сотни и тысячи вместо десятков. Последовательность их работы заранее не фиксирована, и после срабатывания очередного приема необходим цикл поиска следующего применяемого приема. Разумеется, при увеличении числа приемов, работающих в общем процессе, существенно усложняется регулировка взаимодействия между ними, и возникает необходимость в длительной эмпирической оптимизации их решающих правил.

Для организации цикла поиска очередного применяемого приема в решателе используется *процедура сканирования задачи*, которую можно представлять как своего рода модель внутреннего логического зрения. Для большинства приемов активизация их при рассмотрении задачи начинается с обнаружения в логических структурах данных некоторого логического символа, заранее выбранного для этого приема. В качестве такого «ключевого» логического символа берется обычно некоторый логический символ, появление которого необходимо для возможности применения рассматриваемого приема, причем при нескольких возможных выборах предпочтение отдается наиболее редко встречающемуся логическому символу (для уменьшения потерь времени при попытках применения приема).

Указанное закрепление за приемами логических символов предопределяет организацию всей базы приемов решателя по принципу энциклопедии: она распадается на группы приемов, «принадлежащих» соответствующим символам, причем каждая группа реализуется в виде отдельной алгоритмической процедуры A_f — программы логического символа f. В особых случаях прием не удается связать с каким-либо конкретным логическим символом, появление которого является необходимым для срабатывания. Такие приемы (их совсем немного) распределены по четырем логическим символам — названиям «доказать», «описать», «преобразовать», «исследовать» типов задач, при решении которых возможно их применение.

Текущая ситуация, возникающая в процессе работы решателя, описывается последовательностью задач $Z_1, Z_2, \ldots, Z_N$, где Z_{i+1} — вспомогательная задача, введенная при решении задачи Z_i, $i = 1,$

…, $N - 1$). В этой последовательности (далее называем ее цепью задач) задача Z_1 является фиктивной — она создается автоматически при запуске логической системы и используется для организации интерфейса. Эта задача (далее называемая исходной задачей) имеет тип «исследовать» и единственную однобуквенную посылку «вход». При просмотре посылок данной задачи решатель обращается к программе логического символа «вход», которая и является программой интерфейса логической системы. С помощью интерфейса можно ввести некоторую задачу Z_2, которая далее и решается, порождая вспомогательные задачи Z_3, …, Z_N. Будем называть задачу Z_N текущей задачей, Z_2 — корневой задачей. Исходная задача, кроме организации интерфейса, выполняет функции «доски объявлений» — различные процедуры решателя могут обмениваться между собой сообщениями, размещая их в списке комментариев к посылкам этой задачи.

Каждая задача Z_i, $i = 1, \ldots, N$, характеризуется натуральным числом M_i, называемым ее максимальным уровнем и определяющим уровень средств, отведенных для ее решения (по исчерпании этих средств выдается отказ), целым неотрицательным числом m_i, $m_i \leq M_i$, называемым текущим уровнем этой задачи и определяющим уровень средств, среди которых в текущий момент ведется поиск очередного преобразования задачи Z_i, а также вспомогательной информацией, необходимой для возобновления прерванной процедуры решения задачи Z_{i-1} по окончании решения задачи Z_i.

Текущий уровень задачи является одним из входных параметров, получаемых приемами; он учитывается решающими правилами приемов и позволяет организовать необходимые приоритеты в их применении: при меньших значениях этого уровня срабатывают приемы с бóльшим приоритетом. В процессе обучения решающие правила приемов корректируются таким образом, чтобы на каждом шаге выбирался прием, наиболее целесообразный с точки зрения обучающего систему эксперта. При первоначальном обращении к задаче Z_i ее текущий уровень — 0.

Изменение текущей ситуации происходит в следующем рабочем цикле решателя.

1. Происходит обращение к программе логического символа f — типа задачи Z_N. Если при этом не срабатывает ни один из приемов либо сработавшие приемы изменили лишь комментарии задач и ни один из них не указал явно на необходимость повторного рассмотрения задачи (в таких случаях говорим, что процедура не внесла существенных изменений в текущую ситуацию), то переход к пункту 3 , иначе — к пункту 2.

2. Если в результате срабатывания приема определился ответ на задачу Z_N либо был выдан отказ на нее, то возобновляется прерванный ранее прием решения задачи Z_{N-1}, в процессе реализации

которого возникла задача Z_N. При $N = 2$ в этом случае выдается ответ либо отказ на решаемую задачу и возвращение в программу интерфейса решателя; при $N = 1$ происходит выход из логической системы. При отсутствии ответа либо отказа на задачу Z_N текущий уровень этой задачи заменяется на 0, и переход к пункту 1. Это означает, что при наличии существенных изменений, внесенных приемом, решатель повторяет цикл анализа текущей ситуации с самого начала.

3. Осуществляется последовательный просмотр всех условий и посылок F задачи Z_N, вес ν которых равен текущему уровню m_N этой задачи либо $m_N + 1$. Сначала просматриваются условия, затем посылки; порядок просмотра условий (посылок) — слева направо по соответствующим спискам задачи Z_N. Если $\nu = m_N$, то происходит однократный просмотр всех вхождений логических символов φ в F (слева направо); если же $\nu = m_N + 1$, то происходит серия таких просмотров, в процессе которых значение текущего уровня задачи Z_N полагается последовательно равным 0, 1, …, m_N. Для каждого рассматриваемого вхождения логического символа φ в F осуществляется обращение к программе логического символа φ (исходные данные этой программы содержат полную информацию о координатах вхождения символа φ в задачу Z_N). Если процедура не внесла существенных изменений в текущую ситуацию, то переход к рассмотрению очередного вхождения логического символа, иначе — к пункту 2. Если просмотр терма F закончился безрезультатно, то вес его увеличивается на единицу (новые либо видоизмененные посылки и условия задачи получают вес 0). Если просмотр всех условий и посылок задачи Z_N закончился безрезультатно, то текущий уровень задачи Z_N увеличивается на единицу. Если в результате он становится больше, чем максимальный уровень M_N, то на задачу Z_N выдается отказ, иначе — переход к пункту 1.

Использование весов посылок и условий позволяет сузить область просмотра при поиске очередного приема, исключая из нее посылки и условия, имеющие большой вес (они уже были достаточно хорошо рассмотрены ранее, и срабатывание связанного с ними приема маловероятно). В результате происходит локализация рассмотрения задачи, направляемого в первую очередь на новые либо видоизмененные посылки и условия; рассмотрение же всей задачи в целом имеет место, как правило, лишь на начальном этапе ее решения.

По мере повышения текущего уровня m_N задачи Z_N в просмотр вовлекаются ранее отложенные посылки и условия F, вес ν которых больше m_N; это происходит при $\nu = m_N + 1$ (см. пункт 3), причем предварительно осуществляется поиск приема, срабатывающего при рассмотрении F для меньших, чем m_N, значений текущего уровня. Переключение внимания при рассмотрении задачи может происходить также в результате срабатывания приемов, уменьшающих веса тех или иных условий и посылок.

3.4.2. Запуск решения задачи и его пошаговый просмотр

Запуск решения задачи происходит из просмотра списка задач некоторого концевого раздела задачника. Войдя в этот список и создав в нем новую задачу либо выбрав для решения одну из ранее имевшихся задач, следует прежде всего добиться, чтобы верхняя горизонтальная линия данной задачи была прорисована на экране, а верхняя линия предыдущей задачи не была видна на экране. Альтернативный способ — выделение нужной задачи нажатием правой кнопки мыши на ее поле (чтобы задача оказалась выделена, ее верхняя горизонтальная линия опять же должна быть видна на экране), после чего можно произвольно перемещаться по списку задач и выполнять запуск выделенной задачи из любой его точки.

Если требуется получить ответ задачи без отображения процесса решения, то нажимается клавиша «о» (рус.). Если на задачу будет получен ответ, то он будет прорисован в верхней части экрана, с указанием времени решения (в секундах либо минутах), а также с указанием трудоемкости, измеряемой в числе шагов работы интерпретатора языка ЛОС. Ответ и трудоемкость сохраняются в файлах задачника и впоследствии будут прорисовываться непосредственно под условием задачи (время решения не сохраняется). Для исключения их из файлов следует выделить задачу (правой кнопкой мыши) и нажать *Ctrl* + *F4*. Если ответ на задачу не получен, то происходит перерисовка ее текста с указанием под ним времени, затраченного на решение.

Если нужно отображать процесс решения по шагам, то нажимается клавиша «р» (рус.). Каждый последующий шаг обеспечивается нажатием клавиши *Enter*. Если в некоторый момент нужно оборвать пошаговое отображение решения и далее решать задачу до получения ответа, то нажимается клавиша «0» (нуль). Если нужно вообще прервать решение, то нажимается клавиша *Esc*, возвращающая к тексту задачи в задачнике.

При показе очередного шага решения в верхней части экрана прорисовывается описание текущего действия. Над этим описанием отображается вся цепь задач (кроме фиктивной исходной задачи) — сначала идет выбранная из задачника задача (возможно, измененная в процессе решения по сравнению с ее первоначальной версией, оставшейся в задачнике), затем вспомогательная задача, к которой произошло обращение от первой задачи, и т. д. Под последней задачей этой цепи задач и размещается описание текущего действия. При просмотре всех этих записей применяется та же прокрутка, что и при просмотре списка задач в задачнике.

Отображаемые на экране задачи приводятся с теми же сокращениями, что и в задачнике (для включения либо выключения полного просмотра нажимается клавиша «ы»).

Кроме того, в просматриваемой цепи задач могут встречаться «замаскированные» под задачи обращения к вспомогательным процедурам (пакетным операторам, см. описание языка ГЕНОЛОГ), не являющиеся задачами. Такие вспомогательные операторы введены из соображений оптимизации: каждый из них содержит в себе сравнительно небольшое число приемов, и поиск нужного приема в нем ускоряется на порядок по сравнению с поиском по всей базе приемов. Типы этих операторов аналогичны типам задач: проверочный оператор является аналогом задачи на доказательство, нормализатор — аналогом задачи на преобразование, синтезатор — аналогом задачи на описание и анализатор — аналогом задачи на исследование.

Под кадром, содержащим описание текущего действия, могут быть размещены несколько кадров со вспомогательными задачами — эти задачи решались в процессе выполнения данного действия. Можно повторно запустить решение любой из них — выделив ее либо разместив ее верхнюю линию в верхней части экрана и нажав клавишу *Enter*. Эту процедуру можно применять любое число раз и внутри пошагового просмотра решения вспомогательных задач — таким образом создается подобие гипертекста решения. Для возвращения на предыдущий уровень данного гипертекста нажимается клавиша *End*.

Комментарии к очередному действию решателя размещаются в скобках после описания этого действия. Вспомогательные задачи, сопровождающие текущее действие, также снабжаются комментарием, размещаемым в скобках перед описанием задачи. Иногда эти комментарии специально подготовлены для объяснения примененного приема, иногда они просто представляют собой подзаголовок того раздела оглавления базы приемов, в котором размещен примененный прием.

Если комментарий недостаточен для понимания выполненного действия или вообще непонятен (такое может случиться, если комментарий просто является подзаголовком концевого пункта оглавления приемов — некоторые из этих подзаголовков понятны только в контексте заголовков внешних разделов), то можно перейти к просмотру описания сработавшего приема. Конечно, здесь понадобится знакомство с языками, на которых задаются приемы решателя — ЛОС и ГЕНОЛОГ. Для этого нажимается клавиша «б», переводящая в просмотр приема (если прием реализован на языке ГЕНОЛОГ) либо (если прием реализован на языке ЛОС) переводящая в тот концевой пункт оглавления программ, который связан с последней пройденной перед реализацией приема контрольной точкой. В обоих случаях можно также просмотреть ЛОС-программу примененного приема, нажав клавишу «ф» — она переводит в отладчик языка ЛОС. Для возвращения в просмотр текущего действия

решателя из просмотра описания приема на языке ГЕНОЛОГ следует нажать клавишу *End*. Для возвращения из просмотра ЛОС-программы приема следует нажать клавишу «з».

При просмотре текущего шага решения можно посмотреть исходную задачу в задачнике, не прерывая процесса решения. Для этого достаточно нажать клавишу *Home*; возвращение к просмотру текущего шага решения из задачника — по нажатии клавиши *End*.

Если при пошаговом просмотре возник промежуточный результат, представляющий самостоятельный интерес (даже в случае, когда ответ на решаемую задачу так и не будет получен), то можно выделить этот результат (целую формулу или ее часть, одну или несколько) — так же, как это делалось в задачнике, перейти в задачник (через *Home*) и зарегистрировать выделенную формулу в любой из задач или введя новый бланк задачи. Как и обычно, для этого следует войти в формульный редактор и нажать *Insert* + *N*, где *N* — номер выделенного элемента среди всех выделенных элементов, $1 \le N \le 9$. Далее можно нажать *End* и продолжить просмотр шагов решения.

Если возникла необходимость прервать процесс пошаговой трассировки решения некоторой задачи из цепи задач и перейти к очередному шагу для ее надзадачи, то следует выделить (правой кнопкой мыши) данную надзадачу и нажать клавишу *Enter*. Тогда следующий отображаемый на экране шаг будет относиться уже к выбранной надзадаче.

Если требуется прервать затянувшийся шаг решения задачи, то нажимается клавиша *Break*. В результате появится текст текущего выполняемого фрагмента ЛОС-программы, в котором текущий (еще не выполненный) оператор выделен малиновым цветом. Далее возможен анализ текущей ситуации с помощью средств отладчика языка ЛОС либо возвращение в главное меню при нажатии клавиши *Esc*. Можно также просмотреть текущую цепь задач (нажатием клавиши «з») либо (до нажатия «з» либо вернувшись из просмотра цепи задач в ЛОС-программу нажатием «ф») запустить процесс пошаговой трассировки на уровне той задачи, которая при прерывании оказалась текущей. Это делается нажатием клавиш *Space* и *Enter*.

Иногда пошаговый просмотр решения оказывается неудобен из-за того, что на некотором этапе начинается решение трудоемкой вспомогательной задачи, обращение к которой не было вынесено в самостоятельный шаг. Разумеется, по завершении этого решения и отображении срабатывания приема, в рамках которого оно происходило, решение вспомогательной задачи можно повторить для ознакомления с подробностями. Однако длительная пауза из-за неотображаемого процесса решения может оказаться нежелательной. В таких случаях можно повторно запустить пошаговый просмотр решения, изменив его режим непосредственно перед появлением паузы. Для выбора нужного режима трассировки следует нажать

клавишу «т» (это делается либо до запуска решения, из задачника, либо уже в начатом процессе трассировки). После нажатия появляется диалог установки режима.

Для ввода либо отмены условия на трассировку, определяемого в одном из пунктов диалога, следует переместить курсор мыши внутрь прямоугольника этого пункта и нажать левую клавишу мыши (наличие знака «плюс» справа от прямоугольника означает, что условие включено, иначе — выключено). После выбора необходимой комбинации условий курсор мыши перемещается в прямоугольник «Ввести» и снова нажимается левая клавиша мыши.

Для преодоления указанных выше пауз можно воспользоваться пунктом «ручной выбор входа в подпроцесс». Если этот пункт активирован, то каждая попытка обращения решателя к вспомогательной задаче (включая наиболее крупные пакетные операторы) будет выводиться на экран. Чтобы продолжить решение без входа в пошаговую трассировку этой вспомогательной задачи, нажимается *Enter*; чтобы войти в нее, нажимается *Ctrl* + *Enter*.

Заметим, что в этом режиме текущее действие решателя не сопровождается выдачей списка вспомогательных задач, решенных для его выполнения, — сообщения об обращениях к этим задачам уже выдавались на экран до осуществления действия, и была возможность просмотреть их решение по шагам. Недостатком режима с ручным входом в подпроцессы является чрезмерное количество выдаваемых на экран сообщений о попытках решения вспомогательных задач, большая часть которых оказывается неудачной. Действительно ценные шаги тонут в этом потоке сообщений. Поэтому данный режим является лишь техническим, в обычных ситуациях отключенным.

Отладочные режимы трассировки описаны в документации по отладчику языка ЛОС. С помощью этих режимов можно, в частности, обеспечить прерывание процесса решения при попытке применения заданного приема, что часто используется при оптимизации его решающих правил.

Возможен серийный запуск решения задач из задачника. Такой запуск бывает необходим для контроля за изменениями в поведении решателя при его обучении. Простейшая форма этого запуска — выбор в оглавлении задачника нужного подраздела и нажатие сочетания клавиши *Ctrl* + «з» либо *Ctrl* + «э». В первом случае из цикла решения будут исключены все задачи, которые при предыдущем запуске решателем не были решены; во втором случае будут решаться все задачи подряд.

При серийном решении последовательно решаются сначала все задачи из текущего пункта просматриваемого меню (выделенного в момент запуска) оглавления задачника, затем — все задачи из следующего пункта этого меню и т. д. до конца подраздела. На экране при этом последовательно сменяются формулировки решаемых за-

дач. Если решатель слишком долго решает какую-либо задачу (возможно, «залипает» на ней из-за плохих решающих правил), то последовательное нажатие клавиш *Break* и *Esc* переводит в решение следующей задачи. Если до конца решения всех задач подраздела произойдет «зависание» программы и понадобится ее перезапуск, то после перезапуска автоматически восстанавливается прерванный цикл решения задач вплоть до достижения конца меню.

Для обрыва серийного режима следует нажать клавишу *Break* (на экране появится фрагмент программы на языке ЛОС, в котором произошло прерывание, и установится пошаговый режим отладочной трассировки) и далее нажать сочетание клавиш *Ctrl* + «з». Лишь после этого нажатие клавиши *Esc* выведет в главное меню.

По окончании серийного запуска обновляется статистика о результатах решения задач подраздела. Такая статистика позволяет находить «особые точки» в задачнике (например, выявлять задачи, которые перестали решаться или решение которых замедлилось после внесенных в приемы изменений). Для просмотра статистики следует войти в меню нужного подраздела и нажать клавишу «з». После небольшой паузы, необходимой для просмотра всех задач подраздела, на экране возникает таблица, в которой указываются следующие сведения.

1. Пять наибольших величин замедления в решении задач по сравнению с предыдущим запуском. Эти величины измеряются в числе шагов интерпретатора языка ЛОС, как и величины трудоемкости решения задач, сохраняемые в задачнике, — отсюда легко извлекается процент замедления. Если нужно просмотреть имеющие самые большие значения замедления задачи, то нажимается клавиша «з», переводящая в просмотр первой из этих задач. Переход к очередной задаче (отобранные для просмотра задачи упорядочены по убыванию замедлений) осуществляется нажатием клавиши «ш». Для просмотра в таком режиме отбираются не более 40 задач подраздела (это же ограничение распространяется на другие приводимые ниже случаи отбора серий задач).
2. Число нерешенных задач раздела. Для просмотра этих задач нажимается клавиша «н». Чтобы перейти к просмотру очередной нерешенной задачи, следует нажать «ш».
3. Число утерянных решений задач подраздела (только по сравнению с предпоследним циклом решения). Для просмотра серии таких задач сначала нажимается «у» и далее — через нажатия «ш».
4. Пять наибольших величин ускорения в решении задач. Просмотр задач с ускорившимся решением — через нажатие клавиши «с» (рус.) и далее к каждой следующей задаче — через нажатие «ш».
5. Суммарное замедление в решении задач подраздела (отрицательная его величина означает суммарное ускорение).
5. Число изменившихся по сравнению с предпоследним циклом решения ответов на задачи подраздела. Для просмотра серии соот-

ветствующих задач сначала нажимается клавиша «и» и далее — через нажатия «ш».

6. Число сомнительных ответов. Для просмотра серии соответствующих задач — нажатие «о» (рус.) и далее — через «ш».

7. Число задач, замедлившихся более чем на 10 000 шагов интерпретатора языка ЛОС, и число задач, замедлившихся более чем на 100 000.

8. Можно просмотреть список всех ответов на задачи подраздела. Для этого нажимается клавиша «О» (рус.). Если на выделенном ответе нажать клавишу «курсор вправо», то происходит переход к просмотру условия соответствующей задачи.

9. Пять наибольших величин трудоемкости решения задач подраздела (в числе шагов интерпретатора). Для просмотра задач в порядке убывания трудоемкости (не более 40 задач) — нажатие «т» и далее — через «ш».

Серийный запуск позволяет косвенно оценивать «холостой ход» приема — суммарную трудоемкость попыток его применения, не приводивших к срабатыванию приема. Фактически здесь оценивается относительная трудоемкость попыток применения приема на одно его срабатывание. Чтобы получить такую статистику, требуется запустить серийное решение не через *Ctrl* + «з», а через *Ctrl* + «х» (рус.). Тогда в указанной выше таблице статистики будут отображены данные о пяти наихудших случаях такой относительной трудоемкости («холостого хода»). Эти данные суть пары (суммарная трудоемкость попыток применить прием — число срабатываний приема), для которых отношение первого элемента ко второму (если второй равен нулю, то вместо него берется единица) наибольшее. Возможен просмотр серии наихудших приемов (по убыванию указанной характеристики) — через нажатие клавиши «х» (рус.). Переход к каждому очередному приему — через «ш». При просмотре приема повторный вызов на экран указанной пары чисел — через «Х» (рус.); пара (A_1, A_2) прорисовывается в виде двух термов — «пассив(A_1)» и «актив(A_2)».

3.5. Практика работы с решателем

Рассмотрим несколько примеров действий по вводу и решению задач.

3.5.1. Элементарная алгебра

Пример 3.1

Упростить выражение:

$$\sqrt{\frac{4}{x}+\frac{1}{4x^{-1}}-2}+\sqrt{\frac{1}{4x^{-1}}+\frac{2^{-2}}{x}+\frac{1}{2}}.$$

Решение. Находясь в главном меню, выбрать пункт «Оглавление задачника» (клавиша «з» либо левая кнопка мыши в прямоугольнике указанного пункта). Используя клавиши курсора, найти в корневом меню оглавления раздел «Элементарная алгебра» (возможно, сначала понадобится несколько раз нажать «курсор влево»). Войти в этот раздел, далее войти в подраздел «Упрощение выражений», затем в любой из подразделов «Упрощение иррациональных выражений — 1, 2, 3».

В действительности выбор раздела несущественен; решатель будет работать вне зависимости от того, в каком разделе находится задача, и классификация задач по разделам нужна лишь для упрощения поиска нужной задачи вручную.

Используя клавишу *PageDown* либо «курсор вниз», вывести на экран последний пункт выбранного концевого раздела (все его пункты — номера с тремя штрихами). Затем нажать клавишу «к» для создания бланка новой задачи. Далее нажимается «ц» и в оглавлении типов целевых установок выбираются разделы «Преобразовать выражение» — «Упростить выражение в области допустимых значений», причем после выбора последнего пункта нажимается «курсор вправо». Экран расчищается, и в верхней его части возникает текст «Упростить в о.д.з. выражение:». Далее нажимается *Enter*, и формульным редактором вводится упрощаемое выражение.

В процессе набора можно получать справки о клавиатуре формульного редактора, нажимая *F1*. Точнее, нажатие *F1* переводит в общее оглавление справочника по системе, и из его корневого меню нужно перейти в раздел «Формульный редактор». Далее полезно прочитать раздел «Общие сведения»; для получения информации о вводе конкретных символов нужно переходить в соответствующие подразделы. Чтобы вернуться в набор формулы, достаточно нажать *End*.

По завершении набора упрощаемого выражения нажимается *Enter*. В данном примере задача оказывается полностью введенной, и для решения ее без пошаговой трассировки нажимается «о» (рус.), а для входа в пошаговую трассировку (она отображает лишь верхний уровень процесса решения — срабатывания приемов; такую трассировку, в отличие от отладочной трассировки, называем далее семантической) нажимается «р» (рус.). Каждый очередной шаг трассировки — нажатие *Enter*.

Если в некоторый момент под кадром, поясняющим текущее действие, оказываются расположены другие кадры, то эти последние суть кадры обращений к вспомогательным задачам, решавшимся для выполнения текущего действия. Можно выбрать любой из них для детального просмотра решения вспомогательной задачи.

При этом верхняя отделяющая линия кадра должна быть в точности верхней границей экрана; такое выравнивание обеспечивается сочетанием клавиши *Ctrl* и «курсор вверх» либо «курсор вниз». Вход в решение вспомогательной задачи — нажатие *Enter*. По завершении трассировки решения вспомогательной задачи нажатие *Enter* возвращает на предыдущий уровень (такое возвращение можно обеспечить в процессе трассировки нажатием *End*).

Пример 3.2

Решить уравнение:

$$\frac{x^2}{x+2}+1=\frac{4}{x+2}.$$

Решение. Действия аналогичны предыдущим, но выбирается раздел «Элементарная алгебра» — «Решение уравнений» — «Рациональные уравнения — 1, 2». Нажимаются «к», «ц» и выбирается раздел оглавления целевых установок «Найти значения неизвестных» — «Получить полное явное описание значений неизвестных». На экране появляется текст «Найти», под которым размещен курсор формульного редактора. Этим редактором вводятся неизвестные задачи (разделенные запятой) — в данном примере единственная переменная *x*. После нажатия *Enter* ввод целевой установки завершается. Для ввода уравнения снова нажимается *Enter*, и далее происходит набор уравнения.

3.5.2. Планиметрия

Пример 3.3

Основание равнобедренного треугольника равно *a*, угол при вершине равен *b*. Найдите биссектрису, проведенную к боковой стороне.

Решение. Вычислительные задачи по геометрии вводятся в следующем порядке: сначала набираются посылки задачи, перечисляющие известные свойства чертежа; затем вводится целевая установка задачи; затем указываются неизвестные величины, которые должны быть вычислены. Перед набором посылок можно ввести чертеж, однако чертеж может быть создан системой и автоматически (по окончании набора задачи нажимается *Ctrl* + «ч»).

При наборе посылок следует обязательно обозначить все точки, участвующие в задаче, буквами (обычно большими; можно использовать натуральные индексы). Ссылаться на плоские фигуры (треугольники, многоугольники, окружности, прямые и т. д.) можно только через их базисные точки. В нашем случае обозначим вершины треугольника буквами *A*, *B*, *C* и введем первую посылку «△(*ABC*)». Для ввода посылки нажимается *Enter*, затем последовательно нажимаются клавиши «т», «р» (обычно используются первые две буквы вводимого понятия). Это приводит к прорисовке символа △ с расположенной после него открывающей скобкой. Далее последовательно нажимаются *A*, *B*, *C* и закрывающая скобка, после чего — завершающее набор формулы нажатие *Enter*. Никакие разделители между буквами не ставятся. Заметим, что если в обозначении треугольника либо другой фигуры используется буква с индексом, то после такой буквы, перед набором следующей, обязательно нужно нажать на клавишу «звездочка».

После ввода первой посылки (она указывает только на то, что три точки *A*, *B*, *C* образуют вершины некоторого треугольника, т. е. не лежат на одной прямой) нужно ввести посылку, определяющую, что треуголь-

ник равнобедренный. Выберем в качестве основания треугольника вершины A, C и введем посылку $l(AB) = l(BC)$. Для обозначения расстояния дважды нажимается клавиша «l», что приводит к появлению рукописной латинской буквы l с идущей после нее открывающей скобкой. Затем (как в обозначении треугольника) подряд вводятся буквы для точек, между которыми рассматривается расстояние, и ставится закрывающая скобка.

Далее вводятся: посылка $l(AC) = a$, обозначающая длину основания треугольника через a, и посылка $\angle(ABC) = b$ — для величины угла при основании. Чтобы получить обозначение угла, последовательно нажимаются клавиши «у», «г». Далее — как для обозначения треугольника. Как и обычно, вершина угла размещается в обозначении угла посередине.

Для ввода основания D биссектрисы треугольника ABC, проведенной из угла BAC, можно использовать посылку «Биссектреуг(*BACD*)». Другой способ (более громоздкий, но не использующий специального обозначения «Биссектреуг») — пара посылок «$D \in$ прямая (BC)», «биссектриса(*BACD*)». В первом случае нажимаем «И», «Т» (рус.) и далее — как в случае обозначения треугольника, но для четырех букв. Во втором случае сначала вводим D, затем нажимаем *Space*, «b», «е» (лат.); появляется символ $\in$. Далее нажимаем «п», «р» (рус.); появляется слово «прямая» с открывающей скобкой. Вводим буквы B, C и закрывающую скобку. Для ввода «биссектриса(…)» нажимаем клавиши «и», «т».

На этом ввод посылок завершен. Для ввода целевой установки нажимаем клавишу «ц», переводящую в оглавление типов целевых установок. Из корневого меню этого оглавления переходим к пункту «Найти значения неизвестных» — «Выразить значения неизвестных через заданные параметры».

Заметим, что применявшийся в элементарной алгебре пункт «Получить полное явное описание значений неизвестных» в планиметрических задачах на вычисление *не следует использовать*. Это объясняется принципиальным различием понятий «известная» и «неизвестная» в задачах из двух этих разделов. В элементарной алгебре все переменные из списка посылок по умолчанию считались известными и могли входить в ответ задачи. В геометрической задаче на вычисление посылки содержат обозначения точек A, B, C, …, которые не должны появляться в ответе. Соответственно различаются и типы выбираемых целевых установок.

После выбора указанного типа целевой установки нажимается «курсор вправо». Далее сначала вводится буква (или несколько отделенных запятыми букв) для неизвестной (неизвестных) и нажимается *Enter*. Затем вводятся разделенные запятыми буквы для известных числовых параметров, через которые должны быть выражены неизвестные (если таких параметров вообще нет, сразу нажимается *Enter*, иначе оно нажимается после ввода параметров). В нашем примере обозначим неизвестную длину биссектрисы через x; параметры суть a, b.

После ввода целевой установки вводим равенства, связывающие неизвестные (переменные) с теми выражениями, которые они обозначают и которые нужно вычислить. В ответ войдет сама неизвестная, а не обозначаемое ею выражение.

Как уже говорилось выше, после ввода задачи по планиметрии можно ввести чертеж: либо попробовать сделать это автоматически (нажатием *Ctrl* + «ч»), либо ввести чертеж вручную (нажать «ч» и далее воспользо-

ваться геометрическим редактором; чертеж появится перед списком посылок, в начале задачи). Можно также скорректировать вручную чертеж, созданный автоматически (вход в редактирование уже созданного чертежа — снова через «ч»; удаление чертежа — выделить его левой кнопкой мыши и нажать *Ctrl* + *Del*).

Пример 3.4

В трапеции *ABCD* с основаниями *AD* и *BC* имеем *AD* = 3, *BC* = 1. Точка *P* лежит на стороне *AB*, а точка *Q* — на стороне *CD*, причем отрезок *PQ* параллелен основаниям и проходит через точку пересечения диагоналей трапеции. Найти длину отрезка *PQ*.

Решение. Напомним, что в решателе предусмотрены два варианта обозначения трапеции: «трапеция(*ABCD*)» и «Трапеция(*ABCD*)» — первый из них для случая трапеции с бóльшим основанием *AD* и острыми углами при основании; второй — для случая, когда углы при основании не обязательно острые. В обоих случаях указанные записи представляют собой даже не обозначения трапеции, а лишь утверждения о том, что точки *A*, *B*, *C*, *D* являются вершинами соответствующей трапеции. Сама трапеция (как и любой другой четырехугольник) обозначается посредством выражения «фигура(*ABCD*)».

В нашем примере годится любой из указанных способов записи. Например, введем посылку «трапеция(*ABCD*)». Затем вводятся посылки *l*(*AD*) = 3, *l*(*BC*) = 1. Принадлежность точки *P* стороне *AB*, а точки *Q* — стороне *CD* записывается в виде «*P* ∈ отрезок(*AB*), *Q* ∈ отрезок(*CD*)». Параллельность отрезка *PQ* основаниям трапеции записывается как «прямая(*PQ*) || прямая(*AD*)». Чтобы сформулировать условие о том, что отрезок *PQ* проходит через точку пересечения диагоналей трапеции, нужно ввести обозначение для этой точки. Например, обозначим ее через *M*. То, что *M* является точкой пересечения диагоналей, записываем как «*M* ∈ прямая(*AC*), *M* ∈ прямая(*BD*)». Далее добавляем посылку «*M* ∈ отрезок(*PQ*)». На этом ввод посылок завершается. Как и в предыдущей задаче, выбираем целевую установку и вводим единственное условие *x* = *l*(*PQ*) для неизвестной *x*.

Упражнения

Ввести перечисленные ниже задачи в соответствующие разделы компьютерного решателя и просмотреть процесс их решения.

3.1. Решить неравенство:

$$\frac{\sqrt{x^2+x-6}+3x+13}{x+5}>1.$$

3.2. Решить систему уравнений:

$$\begin{cases}(x+y)(x^2-y^2)=16,\\(x-y)(x^2+y^2)=40.\end{cases}$$

3.3. Решить уравнение:

$$(\operatorname{tg} x)^{\cos^2 x} = (\operatorname{ctg} x)^{\sin x}.$$

3.4. Решить неравенство:

$$|x-2|^{\log_4(x+2)-\log_2 x} < 1.$$

3.5. Для всех a решить неравенство: $ax^2 + (a + 1)x + 1 > 0$.

3.6. Определить, при каких a неравенство $\sin^6 x + \cos^6 x + a \sin x \cos x \geq 0$ выполнено для всех значений x.

3.7. Найти все a, при которых из неравенства $x^2 - a(1 + a^2)x + a^4 < 0$ следует неравенство $x^2 + 4x + 3 > 0$.

3.8. Определить, при каких a система

$$\begin{cases} x^2 + y^2 = 2(1 + a), \\ (x + y)^2 = 14 \end{cases}$$

имеет ровно два решения.

3.9. Известно, что $ABCD$ — ромб и радиусы окружностей, описанных около треугольников ABC и ABD, соответственно равны R и r. Найти площадь ромба $ABCD$.

3.10. В параллелограмме $ABCD$ биссектриса угла BAD пересекает сторону CD в точке M, причем $DM/MC = 2$. Известно, что угол CAM равен a. Найти угол BAD.

3.11. Окружность проходит через вершины A и C треугольника ABC, пересекает сторону AB в точке D и сторону BC в точке E. Известно, что $AD = 5$, $AC = 2\sqrt{7}$, $BE = 4$, $BD/CE = 3/2$. Найти угол CDB.

Рекомендуемая литература

1. *Кудрявцев, В. Б.* Введение в теорию интеллектуальных систем / В. Б. Кудрявцев, Э. Э. Гасанов, А. С. Подколзин. — Москва : Изд-во ВМК МГУ, 2006.

2. *Малпас, Дж.* Реляционный язык ПРОЛОГ и его применение / Дж. Малпас. — Москва : Наука, 1990.

3. *Мендельсон, Э.* Введение в математическую логику / Э. Мендельсон. — Москва : Наука, 1971.

4. *Метакидес, Г.* Принципы логики и логического программирования / Г. Метакидес, А. Нероуд. — Москва : Факториал, 1998.

5. *Подколзин, А. С.* Компьютерное моделирование процессов решения математических задач / А. С. Подколзин. — Москва : Изд-во ЦПИ при механико-математическом факультете МГУ, 2001.

6. *Подколзин, А. С.* Компьютерный решатель математических задач / А. С. Подколзин // Доклады Академии наук. — 1994. — Т. 335. — № 4. — С. 427—429.

7. *Подколзин, А. С.* О развитии техники моделирования логических процессов / А. С. Подколзин // Интеллектуальные системы. — 2006. — Т. 10. — Вып. 1-4. — С. 169—188.

8. *Подколзин, А. С.* Компьютерное моделирование логических процессов. Т. 1 / А. С. Подколзин. — Москва : Физматлит, 2008.

9. *Хант, Э.* Искусственный интеллект / Э. Хант. — Москва : Мир, 1978.

10. *Чёрч, А.* Введение в математическую логику / А. Чёрч. — Москва, 1960.

11. *Godel, K.* Die Vollständigkeit der Axiome des logischen Funktionenkalküls / K. Godel // Monatshefte fur Math, und Phys. — 1930. — Bd 37. — P. 349—360.

12. *Herbrand, I.* Recherches sur la theorie de la demonstration / I. Herbrand // Travaux de la Societe des Sciences et des Letters de Varsovie. Classe III. Sciences Mathematiques et Physiques. — 1930. — № 33.

Новинки по дисциплине «Компьютерное моделирование логических процессов» и смежным дисциплинам

1. *Бессмертный, И. А.* Интеллектуальные системы : учебник и практикум для вузов / И. А. Бессмертный, А. Б. Нугуманова, А. В. Платонов. — Москва : Издательство Юрайт, 2023. — 243 с. — (Высшее образование).
2. *Боев, В. Д.* Имитационное моделирование систем : учебное пособие для вузов / В. Д. Боев. — Москва : Издательство Юрайт, 2023. — 253 с. — (Высшее образование).
3. *Галиаскаров, Э. Г.* Анализ и проектирование систем с использованием UML : учебное пособие для вузов / Э. Г. Галиаскаров, А. С. Воробьев. — Москва : Издательство Юрайт, 2023. — 125 с. — (Высшее образование).
4. *Долганова, О. И.* Моделирование бизнес-процессов : учебник и практикум для вузов / О. И. Долганова, Е. В. Виноградова, А. М. Лобанова ; под редакцией О. И. Долгановой. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2023. — 322 с. — (Высшее образование).
5. *Древс, Ю. Г.* Имитационное моделирование : учебное пособие для вузов / Ю. Г. Древс, В. В. Золотарёв. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2023. — 142 с. — (Высшее образование).
6. *Кудрявцев, В. Б.* Распознавание образов : учебное пособие для вузов / В. Б. Кудрявцев, Э. Э. Гасанов, А. С. Подколзин. — 2-е изд. — Москва : Издательство Юрайт, 2023. — 107 с. — (Высшее образование).
7. *Кудрявцев, В. Б.* Теория автоматов : учебник для вузов / В. Б. Кудрявцев, Э. Э. Гасанов, А. С. Подколзин. — 2-е изд. — Москва : Издательство Юрайт, 2023. — 204 с. — (Высшее образование).
8. *Маликов, Р. Ф.* Компьютерное моделирование динамических систем в среде rand model designer : учебное пособие для вузов / Р. Ф. Маликов. — Москва : Издательство Юрайт, 2023. — 223 с. — (Высшее образование).
9. Моделирование процессов и систем : учебник и практикум для вузов / Е. В. Стельмашонок, В. Л. Стельмашонок, Л. А. Еникеева,

С. А. Соколовская ; под редакцией Е. В. Стельмашонок. — Москва : Издательство Юрайт, 2023. — 289 с. — (Высшее образование).

10. Моделирование систем и процессов : учебник для вузов / В. Н. Волкова [и др.] ; под редакцией В. Н. Волковой, В. Н. Козлова. — Москва : Издательство Юрайт, 2023. — 450 с. — (Высшее образование).

11. Моделирование систем и процессов. Практикум : учебное пособие для вузов / В. Н. Волкова [и др.] ; под редакцией В. Н. Волковой. — Москва : Издательство Юрайт, 2023. — 295 с. — (Высшее образование).

12. *Перевалов, В. П.* Математическое моделирование химико-технологических процессов : учебное пособие для вузов / В. П. Перевалов, Г. И. Колдобский. — Москва : Издательство Юрайт, 2023. — 53 с. — (Высшее образование).

13. *Рейзлин, В. И.* Математическое моделирование : учебное пособие для вузов / В. И. Рейзлин. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2022. — 126 с. — (Высшее образование).

14. *Советов, Б. Я.* Базы данных : учебник для вузов / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2023. — 420 с. — (Высшее образование).

15. *Советов, Б. Я.* Моделирование систем. Практикум : учебное пособие для бакалавров / Б. Я. Советов, С. А. Яковлев. — 4-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2022. — 295 с. — (Бакалавр. Академический курс).

16. *Стружкин, Н. П.* Базы данных: проектирование : учебник для вузов / Н. П. Стружкин, В. В. Годин. — Москва : Издательство Юрайт, 2023. — 477 с. — (Высшее образование).

17. Теоретические основы моделирования : учебник для вузов / Е. В. Стельмашонок, В. Л. Стельмашонок, Л. А. Еникеева, С. А. Соколовская ; под редакцией Е. В. Стельмашонок. — Москва : Издательство Юрайт, 2023. — 65 с. — (Высшее образование).

18. *Щепетов, А. Г.* Основы проектирования приборов и систем. Задачи и упражнения. Mathcad для приборостроения : учебное пособие для вузов / А. Г. Щепетов. — 2-е изд., стер. — Москва : Издательство Юрайт, 2023. — 270 с. — (Высшее образование).

Наши книги можно приобрести:

Учебным заведениям и библиотекам:
в отделе по работе с вузами
тел.: (495) 744-00-12, e-mail: vuz@urait.ru

Частным лицам:
список магазинов смотрите на сайте urait.ru
в разделе «Частным лицам»

Магазинам и корпоративным клиентам:
в отделе продаж
тел.: (495) 744-00-12, e-mail: sales@urait.ru

Отзывы об издании присылайте в редакцию
e-mail: gred@urait.ru

**Новые издания и дополнительные материалы доступны
на образовательной платформе «Юрайт» urait.ru,
а также в мобильном приложении «Юрайт.Библиотека»**

Учебное издание

**Кудрявцев Валерий Борисович,
Гасанов Эльяр Эльдарович,
Подколзин Александр Сергеевич**

КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ ЛОГИЧЕСКИХ ПРОЦЕССОВ

Учебник для вузов

Формат $70\times100\,{}^{1}\!/_{16}$.
Гарнитура «Charter». Печать цифровая.
Усл. печ. л. 11,09.

ООО «Издательство Юрайт»
111123, г. Москва, ул. Плеханова, д. 4а.
Тел.: (495) 744-00-12. E-mail: izdat@urait.ru, www.urait.ru