

ЛОГИЧЕСКИЙ ПОДХОД К ИСКУССТВЕННОМУ ИНТЕЛЛЕКТУ

*От классической логики к логическому
программированию*

Перевод с французского
П. П. Пермякова
под редакцией
Г. П. Гаврилова



Москва «Мир» 1990

**APPROCHE LOGIQUE
DE L'INTELLIGENCE
ARTIFICIELLE
1 DE LA LOGIQUE CLASSIQUE
À LA PROGRAMMATION LOGIQUE**

par
**André Thayse, Pascal Gribomont,
Georges Louis, Dominique Snyers. Pierre Wodon**
Philips Research Laboratory, Bruxelles

Paul Gochet
Université de Liège
Eric Grégoire
Université de Louvain, Louvain-la-Neuve
Eduardo Sanchez
Ecole Polytechnique Fédérale de Lausanne
avec la collaboration de
Philippe Delsarte
Philips Research Laboratory, Bruxelles

**Dunod
informatique**

ББК 32.973

Л69

УДК 681.3

Авторы: А. Тей, П. Грибомон, Ж. Луи, Д. Снийерс, П. Водон, П. Гоше, Э. Грегуар, Э. Санчес, Ф. Дельсарт

Л69 **Логический** подход к искусственному интеллекту: от классической логики к логическому программированию: Пер. с франц./Тейз А., Грибомон П., Луи Ж. и др. — М.: Мир, 1990. — 432 с., ил.

ISBN 5-03-001636-8 (русск.)

Монография специалистов из Бельгии и Швейцарии, излагающая проблемы и методы искусственного интеллекта с точки зрения математической логики. Она состоит из шести глав: логика, аксиоматические системы, представление знаний и рассуждений, логика и модифицируемые рассуждения, формальные грамматики и логическое программирование, Пролог и логическое программирование. Книга построена так, что для понимания материала от читателя требуется только знание основ информатики.

Для всех изучающих и использующих методы искусственного интеллекта и логического программирования.

Л $\frac{1602110000-423}{041(01)-90}$ 22—90

ББК 32.973

Редакция литературы по математическим наукам

ISBN 5-03-001636-8 (русск.)
ISBN 2-04-018658-1 (франц.)

© Bordas, Paris, 1988
© перевод на русский язык,
П. П. Пермиков, 1990

Предисловие редактора перевода

Многообразие научных и технических исследований, называемое искусственным интеллектом, уже давно использует различные логические средства — язык, понятия и приемы логических исчислений. В искусственном интеллекте есть целая область, существенно опирающаяся на логические представления и конструкции, основная ее задача состоит в разработке способов доказательства теорем.

Вполне естественным поэтому представляется стремление авторов данной книги рассказать и показать, где и как работает логика в искусственном интеллекте; причем они попытались сделать это так, чтобы изложение было доступно даже читателю, не имеющему специальной подготовки ни по искусственному интеллекту, ни по логике.

Отметим, однако, что решить эту сложную задачу в полной мере авторам, как нам кажется, пока не удалось. Но все же «развеять немного туманную завесу» над довольно обширной «логической панорамой» искусственного интеллекта они смогли.

Читателю, не являющемуся специалистом по логике, книга, бесспорно, сослужит добрую службу, хотя и потребует от него систематического ее прочтения и привлечения хорошего руководства по математической логике. Для читателя, имеющего традиционную логическую подготовку, т. е. изучавшего теорию высказываний (логику и исчисление) и теорию предикатов первого порядка (логику и исчисление), интерес могут представить гл. 3 и 4, материал которых на русском языке в достаточно последовательном и полном виде пока еще не появлялся. Та часть книги, в которой описывается ряд аспектов логического программирования, просто представляет читателю некоторые взаимосвязи, существующие между логическими исчислениями и языками логического программирования. Она, естественно, не может служить руководством по Прологу.

В предисловии авторы говорят о своем намерении осветить в последующих томах и другие важные приложения логики в искусственном интеллекте. Как стало известно, в настоящее время появился второй том, имеющий подзаголовок «От модальной логики к логике баз данных».

В заключение отметим, что, по нашему мнению, книга будет полезна научным и инженерно-техническим работникам, а также студентам старших курсов вузов и всем читателям, интересующимся приложениями математической логики.

Г. П. Гаврилов

Предисловие

Цель этой книги — представить понятия и методы искусственного интеллекта (ИИ), используя в качестве определяющего логический подход. Мы стремились добиться достаточно автономного и дидактически выдержанного изложения. Оно ориентировано на читателей (студентов или исследователей), имеющих хорошую культуру в области математики и информатики. Однако особых знаний ни в логике, ни в ИИ не предполагается. Мы пока что запланировали выпустить два тома. Содержание данного тома, первого из них, можно описать следующим образом.

В первой главе собраны математические и логические сведения, которые будут использоваться в дальнейшем. Излагаются основные понятия классической логики. Сперва рассмотрено исчисление высказываний, а затем — исчисление предикатов. Особое внимание уделено методу резолюций и связанным с ним понятиям, что обусловлено их ролью в приложениях.

Вторая глава представляет аксиоматический подход к логике и служит введением в теории первого порядка. В ней показано, как исчисление предикатов превращается в основу теории для изучения специфических математических структур. В этом контексте изложены некоторые фундаментальные вопросы логики, естественным образом продолжающиеся в теоретическую информатику. В частности, это касается алгоритмических языков Тьюринга и Гёделя, тезиса Чёрча, класса вычислимых функций и понятия разрешимости.

В третьей главе показано, как классическая логика (особенно логика предикатов) может использоваться для представления знаний и автоматических рассуждений, относящихся к ним. Изложены методы, позволяющие преобразовать логическое представление в сетевое и объектное. Затем обсуж-

даются сравнительные достоинства этих различных представлений. Классическая логика связана с формализацией корректных рассуждений. Однако, как это бывает в ИИ, моделирование рассуждений не ограничивается областью абсолютно корректных рассуждений. Основанные на неполной, неточной или изменчивой информации, наши рассуждения часто гипотетичны, лишь в той или иной степени правдоподобны и предполагают осуществление систематических пересмотров (модификаций).

Четвертая глава служит введением в логики, которые предназначены для формализации модифицируемых рассуждений: логики умолчаний, модальные логики знания и веры, немонотонные логики, автоэпистемические логики.

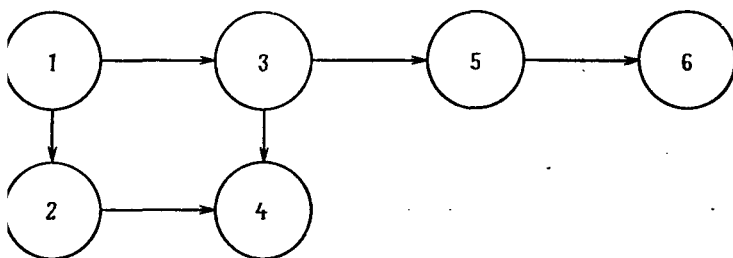
Пятая глава — вспомогательная. В ней показывается, как логическая интерпретация формальных грамматик и их правил вывода приводит к языкам логического программирования, наиболее известным из которых является Пролог. Грамматики классифицированы по иерархии Хомского. Каждая из грамматик этой иерархии описывается соответствующей машиной или автоматом. Автомат является основной моделью в теории грамматик и языков. Эта модель послужила источником сетевого формализма. Показана связь между такими сетями и языками функционального программирования, среди которых особенно типичен Лисп.

В шестой главе представлен язык программирования Пролог. Его создание навеяно формальной логикой и формальными грамматиками. В Прологе некоторые логические формулы (хорновские дизъюнкты) становятся инструкциями, допускающими исполнение на ЭВМ. Пролог можно считать языком ИИ, который хорошо приспособлен для автоматизации некоторой формы логических рассуждений. В этом смысле он представляет собой итог изучения понятий и методов ИИ, основанных на логике.

Итак, главная цель авторов — изложить согласованно и стройно набор дисциплин, включающий: классические логики, представление знаний и корректные рассуждения, неклассические логики, моди-

фицируемые рассуждения, формальные грамматики, теорию автоматов, логическое программирование (особенно язык Пролог). Это интеграция различных дисциплин такого типа для решения сложных проблем, составляющих предмет того, что обычно принято называть ИИ. Среди проблем, особенно часто исследуемых свойственными ИИ методами, назовем распознавание и понимание речи и изображений, создание экспертных систем, имитацию рассуждений (например, в функционировании сознания или медицинских дисциплинах), робототехнику.

Зависимость между главами первого тома этой книги можно схематически изобразить так.



Планируется выпуск второго тома, который будет посвящен четырем следующим темам:

- временная логика и ее приложения в ИИ и информатике,
- углубленное изучение представления знаний и рассуждений,
- логические грамматики и их применение к моделированию естественного языка и пониманию речи,
- логика баз данных и баз знаний.

Авторы хотели представить в этих двух томах основы ИИ, руководствуясь логикой. Важные аспекты ИИ были либо сильно сокращены (как, например, логическое программирование), либо полностью обойдены молчанием. В частности, последнее относится к:

- функциональному программированию и объектно-ориентированному программированию,

- доказательству теорем и верификации программ,
- эвристикам поиска и стратегиям,
- видению, пониманию речи, обучению,
- построению экспертных систем.

Эти темы могли бы быть изложены в последующих томах.

Мы благодарны всем, кто помог нам в работе:

Даниэль Брикс, Мари-Франс Деклерфей, Даниэль Дзиржовски, Пьер-Ив Шоббен и Мишель Зинцофф согласились прочесть и прокомментировать части этой книги.

Анна-Мари Де Сёсте и Эдит Моэ участвовали в редактировании текста.

Один из авторов (Эрик Грегуар) благодарит Бельгийский институт научных исследований в промышленности и сельском хозяйстве за оказанную поддержку (Конвенция 4856: Общее направление исследований в ИИ).

1. Логика

1.1. Исчисление высказываний

1.1.1. Введение

Исчисление высказываний — одна из самых простых теорий, однако оно основательно используется в весьма различных областях. Логикам, информатикам и математикам необходима полная ясность.

Исчисление высказываний изучает предложения, которые могут быть либо истинными, либо ложными. Рассмотрим три следующих истинных предложения (утверждения):

- За четверть часа до своей смерти он был еще жив.
- Если верно, что когда идет дождь, то дорога мокрая, то справедливо также и следующее утверждение: если дорога сухая, то дождя нет.
- Земля вертится.

Чтобы убедиться в правильности первого предложения, достаточно понимать смысл слов: это предложение является *истиной языка*. Чтобы принять второе утверждение, достаточно понимать смысл некоторых слов (если ... то, нет), а также знать, что куски фразы «идет дождь» и «дорога мокрая» являются *высказываниями*, т. е. предложениями, которые могут быть истинными или ложными. Второе предложение останется истинным, если заменить эти два высказывания другими. Такие истины языка называются *логическими истинами*. Напротив, третье предложение не является истиной языка, так как оно выражает некоторый факт (в данном случае — из физики и астрономии). Таким образом, это предложение — *фактическая истина*.

1.1.2. Словарь

Пропозициональный логический словарь традиционно состоит из бесконечного счетного множества

высказываний, обозначаемых строчными буквами (иногда с индексами), и пяти связок (точнее, логических или пропозициональных, связок), описанных и таблице на рис. 1.1.

название	обычное обозначение	тип	другие обозначения
отрицание	\neg	унарный	\neg , \sim , not, не
конъюнкция	\wedge	бинарный	$\&$, \cdot , and, и
дизъюнкция	\vee	бинарный	$ $, or, или
импликация	\supset	бинарный	\Rightarrow , \rightarrow
эквивалентность	\equiv	бинарный	\Leftrightarrow , \leftrightarrow , \sim

Рис. 1.1. Логические связки.

Высказывания представляют собой предложения, которые могут быть истинными или ложными. Логическая истинность, упомянутая в § 1.1.1, может быть записана как

$$(p \supset q) \supset (\neg q \supset \neg p).$$

Ясно, что эта фраза остается логически истинной при любых предложениях, представленных высказываниями p и q . Естественно потребовать, чтобы одинаковые высказывания представляли одинаковые предложения. Такой тип замены называется *одновременной подстановкой*.

Здесь почти не рассматривается соответствие между фразами (и рассуждениями) естественного языка, с одной стороны, и формулами исчисления высказываний, с другой стороны [86]. К этому вопросу мы еще вернемся в § 1.1.5.

1.1.3. Синтаксис исчисления высказываний

Теперь займемся логикой высказываний как таковой, т. е. безотносительно к ее связям с фразами естественного языка. В таком контексте логику высказываний часто называют *исчислением высказываний*.

Словарь исчисления высказываний дает возможность строить сложные, или составные, высказывания из исходных (простых, элементарных), соединяя последние связками. *Правила построения* описывают те выражения, которые являются объектами языка. Такие высказывания называют *формулами*. Аналогия с естественными языками очевидна: фраза — это составное высказывание, построенное по определенным правилам.

Совокупность правил построения выглядит так:

- **Базис:** всякое высказывание является формулой.
- **Индукционный шаг:** если X и Y — формулы, то $\neg X$, $(X \wedge Y)$, $(X \vee Y)$, $(X \supset Y)$ и $(X \equiv Y)$ — формулы.
- **Ограничение:** формула однозначно получается с помощью правил, описанных в базисе и индукционном шаге.

Эта *грамматика* требует некоторых пояснений. Прежде всего, появились два новых типа символов: X , Y и $(,)$. Круглые *скобки* позволяют указать порядок, в котором применялись правила построения. Например, в формуле

$$(p \wedge (q \vee r)) \quad (1.1)$$

индукционный шаг применялся дважды: первый раз — при построении формулы $(q \vee r)$ из формул q и r , а второй — при построении заключительной формулы из формул p и $(q \vee r)$. Скобки здесь использовались так же, как и в арифметике. Естественность употребления скобок является следствием древовидности структуры логических формул и арифметических выражений. Описанное в базисе правило сопоставляет высказывания висячим узлам дерева, а индуктивное правило, сформулированное в индукционном шаге, порождает дерево, растущее из некоторого узла (связки) и построенное на основе одного из двух ранее сформированных деревьев (рис. 1.2). Корневой узел (корень) дерева высказываний, соответствующего некоторой формуле, является связкой. Ее называют *главной связкой* формулы. Формула, у которой главная связка — импликация (эквивалентность), иногда называется *условной* (*биусловной*).

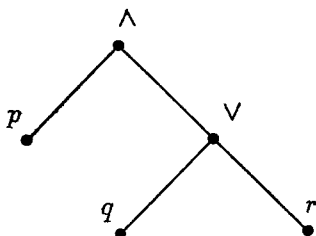


Рис. 1.2. Древовидная форма формулы (1.1).

Естественным образом вводятся и прописные буквы X и Y . Это *метасимволы*, т. е. не принадлежащие языку обозначения, позволяющие вводить понятия и свойства этого языка. Буквы p, q, r представляют определенные высказывания. Метасимволы X и Y служат для обозначения формул вообще.

Если при построении формулы X формула Y используется как элемент в индуктивном правиле, то Y называется *подформулой* формулы X . Множество подформул формулы (1.1) таково:

$$\{p, q, r, (q \vee r)\}.$$

Множество формул является подмножеством множества всех слов (соединений), построенных из исходных символов. Это множество слов *счетное* (можно установить взаимно однозначное соответствие между ним и множеством натуральных чисел). Следовательно, множество формул счетное. Оно является также *рекурсивным* (или *разрешимым*), то есть можно достоверно выяснить, является ли данное слово (соединение) формулой или нет. Скобки в изображении формул служат также для установления различия формул, например

$(p \wedge (q \vee r))$ и $((p \wedge q) \vee r)$ — разные формулы.

Аналогичная ситуация имеет место и в арифметике. Так, например, формулы $a * (b + c)$ и $(a * b) + c$ — различные.

По причинам удобочитаемости некоторые пары круглых скобок иногда заменяют парами фигурных или квадратных. Кроме того, когда позволяет контекст, некоторые пары скобок вообще опускают.

1.1.4. Семантика исчисления высказываний

Объектами изучения естественных и формальных языков являются, в частности, *синтаксис* (который позволяет распознавать фразы среди наборов слов) и *семантика* (которая придает определенное значение фразам). В равной мере это относится к исчислению высказываний.

Уже отмечалось, что высказывание либо истинно, либо ложно. Поэтому введем *семантическую область* **{И, Л}**. *Интерпретировать* формулу — значит приписать ей одно из двух *значений истинности* **И** или **Л** (часто используют обозначения 1 для **И** и 0 для **Л**).

Семантика (то есть набор правил интерпретации формул) должна быть *композиционной*: значение формулы должно быть функцией значений ее составляющих. Точнее, значение истинности формулы зависит только от структуры этой формулы и от значений истинности составляющих ее высказываний.

Таким образом, связи исчисления высказываний представляют *функции истинности*¹⁾: например, значение истинности формулы $(X \wedge Y)$ будет известно, если известны значения истинности X и Y . Рис. 1.3 описывает семантику отрицания. Семантика бинарных связок дана на рис. 1.4.

X	$\neg X$
и	л
л	и

Рис. 1.3. Таблица истинности для отрицания.

X	Y	$X \wedge Y$	$X \vee Y$	$X \supset Y$	$X \equiv Y$
и	и	и	и	и	и
и	л	л	и	л	л
л	и	л	и	и	л
л	л	л	л	и	и

Рис. 1.4. Таблица истинности для бинарных связок.

¹⁾ Называемые также *булевыми* (или *булевскими*) *функциями* или *функциями алгебры логики*. — Прим. ред.

p	q	r	$q \supset r$	$p \supset (q \supset r)$	$p \wedge q$	$(p \wedge q) \supset r$	(1.2)
и	и	и	и	и	и	и	и
и	и	л	л	л	и	л	и
и	л	и	и	и	л	и	и
и	л	л	и	и	л	и	и
л	и	и	и	и	л	и	и
л	и	л	л	и	л	и	и
л	л	и	и	и	л	и	и
л	л	л	и	и	л	и	и

Рис. 1.5. Пример.

Интерпретация — это отображение i , сопоставляющее каждому элементарному высказыванию p некоторое значение истинности. Интерпретацию i , заданную на множестве элементарных высказываний, можно распространить (продолжить) на множество формул (высказываний) посредством таблиц истинности. Соответствующее продолжение (расширение) I тоже называется *интерпретацией*. Интерпретация, при которой истинное значение формулы есть И, называется *моделью* этой формулы.

Литера¹⁾ — это элементарное высказывание или его отрицание. Литеры p и $\neg p$ *противоположны*. Интерпретация определяет разбиение множества L литер на два подмножества $L_{\text{и}}$ и $L_{\text{л}}$, каждое из которых содержит по одному элементу из каждой пары противоположных литер.

Для иллюстрации метода таблиц истинности докажем, что формула

$$(p \supset (q \supset r)) \equiv ((p \wedge q) \supset r) \quad (1.2)$$

истинна при любых интерпретациях. Соответствующая таблица с 8 возможными интерпретациями приведена на рис. 1.5.

1.1.5. Исчисление высказываний и естественный язык

В исчислении высказываний имеется пять связок, использование и смысл которых были формально пред-

¹⁾ Или *литерал*. — Прим. перев.

ставлены в предыдущих параграфах. С одной стороны, ясно, что для этих связок имеются «эквиваленты» в естественном языке. В этом параграфе мы очертим границы сходства между логическими связками и связками естественного языка. С другой стороны, естественный язык был бы сильно обеднен сокращением числа его связок до пяти. Далее в данном параграфе мы покажем, что, напротив, исчисление высказываний не станет богаче, если ввести дополнительные связки.

В естественном языке связки «не», «и», «или», «если ..., то» на первый взгляд вполне однозначно описываются приведенными выше функциями истинности. Например, фразы

Хорошая погода или идет дождь

и

Идет дождь или хорошая погода

кажутся синонимами. Однако иначе обстоит дело с фразами

Ему стало страшно и он убил чужака

и

Он убил чужака и ему стало страшно,

так как слово «и» подчеркивает здесь определенный временной и причинный нюанс. Наконец, многочисленные связки естественного языка ни в коей мере не соответствуют функциям истинности. Связка «потому что» служит хорошим тому примером. Чтобы убедиться в этом, предположим, что идет дождь (высказывание p) и что земля мокрая (высказывание q). Легко принимается как истинное высказывание

q потому что p ,

но нельзя считать истинным высказывание

p потому что q .

Логическая дизъюнкция является *неразделительной*¹⁾: она истинна, если истинным является хотя бы один

¹⁾ Или, иначе, *соединительной*. — Прим. перев.

из двух ее операндов. В естественном языке союз «или» иногда является исключаящим (разделительным). В предложении «целое число четно или нечетно» одна ветвь альтернативы истинна, а другая ложна.

Импликация — очень важная связка; она отражает структуру рассуждений, в частности математических. Первый ее операнд называется *посылкой* (или *антецедентом*), а второй — *заключением* (или *консеквентом*). Ясно, что если посылка истинна, то импликация принимает значение истинности заключения. Однако может вызвать удивление, что импликация бывает истинна и тогда, когда ее посылка ложна. Тем не менее легко убедиться, что импликация (как мы ее определили) является единственной связкой, удовлетворяющей следующим требованиям:

- Если первый операнд истинный, то значение истинности совпадает со значением второго операнда.
- Значение истинности зависит от двух операндов.
- Связка некоммутативна.

В той мере, в какой эти три критерия удачно очерчивают естественную импликацию, принятое нами определение служит наилучшим возможным приближением. Чтобы подчеркнуть этот нюанс, смоделированную связкой \supset импликацию иногда называют *материальной импликацией*. Материальная импликация — основная связка в математических рассуждениях. Она соединяет условие H и утверждение T в теореме. Однако было бы незаконно писать $H \supset T$, так как H и T — не логические формулы, а какие-то достаточно произвольные высказывания: неформальные или принадлежащие некоторому нелогическому формализму. В таких случаях лучше писать $H \Rightarrow T$. Аналогично, в таком контексте лучше писать $C_1 \Leftrightarrow C_2$, а не $C_1 \equiv C_2$.

Исчисление высказываний выражает только чисто функционально-истинностные связи между высказываниями. Это очень хорошо, однако желательно иметь возможность выражать все многообразие взаимосвя-

зей. Кроме того, нет особых причин ограничиваться унарными и бинарными связками.

Связка (функционально-истинностная) задается своей таблицей истинности. Таблица для связки с n операндами имеет 2^n строк. С каждой из этих строк связано свое значение истинности. Таким образом, можно задать 2^{2^n} n -арных (то есть с n операндами) связок. Однако на самом деле двух связок \neg и \supset достаточно для порождения всех этих многочисленных связок при любых значениях n .

Три другие традиционные связки могут быть «определены» с помощью отрицания и импликации¹⁾ посредством следующих всегда истинных формул:

$$\begin{aligned}(X \vee Y) &\equiv (\neg X \supset Y), \\(X \wedge Y) &\equiv \neg (X \supset \neg Y), \\(X \equiv Y) &\equiv ((X \supset Y) \wedge (Y \supset X)).\end{aligned}\tag{1.3}$$

(Здесь центральная связка \equiv понимается как символ, обозначающий «равенство по определению».)

С другой стороны, легко показать, что каждую n -арную связку можно определить (выразить, описать) через пять традиционных связок. Приведем пару расхожих примеров (как на естественном языке, так и на математическом). Формула

$$(X \wedge \neg Y) \vee (\neg X \wedge Y)$$

задает *исключающую дизъюнкцию*²⁾ X и Y . Далее, высказывание «если X , то Y , иначе Z » описывается формулой

$$(X \supset Y) \wedge (\neg X \supset Z),$$

соответствующей часто употребляемой тернарной связке. Отметим еще, что связками с нулевым числом операндов являются только **И** и **Л**. Их можно также считать n -арными связками для всех натуральных значений n .

¹⁾ Можно еще выбирать: отрицание и конъюнкцию, отрицание и дизъюнкцию. Напротив, используя только отрицание и эквивалентность, нельзя получить все другие связки.

²⁾ Или, иначе, *сложение по модулю 2*, или *разделительную дизъюнкцию*. — Прим. ред.

Заметим наконец, что бинарная связка «не-и»¹⁾, обозначаемая \uparrow , позволяет определить все другие связки. По определению $(X \uparrow Y)$ ложно, только если X и Y истинны. В частности, $(X \uparrow X)$ эквивалентно $\neg X$. Это свойство порождать все другие связки (см. [86]) используется в цифровой электронике. Имея логические схемы, реализующие «не-и», можно построить любую логическую схему, моделирующую заданную функцию истинности (и соответствующую ей связку). Связка «не-или»²⁾, обозначаемая \downarrow , является еще одной бинарной связкой, порождающей все другие связки. По определению $(X \downarrow Y)$ истинно тогда и только тогда, когда X и Y ложны.

1.1.6. Выполнимые и общезначимые формулы

В этом параграфе мы рассмотрим формулы, которые истинны при всех интерпретациях, а также формулы, которые при всех интерпретациях ложны. В дальнейшем это подведет нас к определению логического следствия.

Формула *семантически выполнима* или просто *выполнима*, если она допускает некоторую модель, т. е. ее можно интерпретировать со значением И. Например, формулы $(p \wedge q)$ и $(p \vee q)$ выполнимы: они истинны, в частности, если p и q истинны. Другие примеры: все высказывания выполнимы, отрицания высказываний тоже выполнимы. Формула, не являющаяся выполнимой, называется *невыполнимой*. Формула $(p \wedge \neg p)$ невыполнима.

Формула *общезначима*, если она истинна, независимо от истинностных значений, приписанных составляющим ее высказываниям. Формула (1.2) общезначима так же, как и формулы $(p \vee \neg p)$, $(p \supset p)$ и $(\neg \neg p \equiv p)$. Формулу, которая не является общезначимой, иногда называют *необщезначимой*.

Сразу же заметим, что отрицание общезначимой формулы — невыполнимая формула. Точно так же

¹⁾ Иные названия: дизъюнкция отрицаний или штрих Шеффера. — Прим. перев.

²⁾ Конъюнкция отрицаний или стрелка Пирса. — Прим. перев.

отрицание выполнимой формулы — необщезначимая формула. Иногда *нейтральной* называют формулу, которая не является ни общезначимой, ни невыполнимой.

Формула, содержащая k различных высказываний, допускает 2^k интерпретаций. Рассмотрев все эти 2^k интерпретаций, можно тотчас определить, с какой формулой мы имеем дело — общезначимой, нейтральной или невыполнимой. Из рис. 1.5 (§ 1.1.4) следует, что формула (1.2) общезначима, а, например, формулы $p \wedge q$ и $p \supset (q \supset r)$ нейтральны.

Таким образом, множество формул естественно делится на три рекурсивных подмножества (§ 1.1.3). Эти три подмножества бесконечны.

Общезначимые формулы исчисления высказываний часто называют *тавтологиями*. Если A — формула, то запись

$$\models A$$

означает, что A есть тавтология. Более общо: если E — множество формул, то запись

$$E \models A$$

означает, что при всех интерпретациях, при которых истинны все формулы из E , истинна также формула A . Формула A называется *логическим следствием* из E . Таким образом, тавтология — логическое следствие из пустого множества. Если E содержит единственный элемент B , то пишут $B \models A$ вместо $\{B\} \models A$. Легко устанавливается следующее утверждение: A является логическим следствием из B тогда и только тогда, когда материальная импликация $(B \supset A)$ есть тавтология (это остается верным, в частности, если B невыполнима или A общезначима). Данный результат можно записать так:

$$B \models A \Leftrightarrow \models (B \supset A).$$

В более общем виде имеем

$$\{H_1, \dots, H_n\} \models C \Leftrightarrow \models (H_1 \wedge \dots \wedge H_n) \supset C.$$

В этом контексте формулы H_i называют *гипотезами*, а формулу C — *заключением*. Фундаментальная

проблема логики, называемая *проблемой дедукции*, состоит в следующем: определить, является ли формула C логическим следствием множества формул E . Мы только что видели, что, когда множество E конечно, проблема дедукции сводится к определению некоторой условной общезначимости. Определение, которое мы сейчас введем, позволит заняться проблемой дедукции эффективнее.

Множество формул E *семантически выполнимо* (или просто *выполнимо*), если все его элементы допускают общую модель; в противном случае оно *невыполнимо*. Таким образом, множество формул уподобляется (с точки зрения семантики) конъюнкции своих элементов.

Формула C является логическим следствием конечного множества E тогда и только тогда, когда $E \cup \{\neg C\}$ невыполнимо. В этом состоит *принцип дедукции*. Заметим также, что множество E невыполнимо тогда и только тогда, когда \perp — его логическое следствие. Или еще можно сказать, что любая формула — его логическое следствие. Символически принцип дедукции записывается следующим образом:

$$\{H_1, \dots, H_n\} \models C \Leftrightarrow \{H_1, \dots, H_n, \neg C\} \models \perp.$$

Различные формулировки этого принципа будут даны в § 1.1.9. Практический метод применения принципа дедукции представлен в § 1.1.13. Случай дедукции с бесконечным множеством формул рассматривается в § 1.1.18.

Если формулы A и B — логические следствия друг друга, то они называются *логически эквивалентными*. Такая ситуация имеет место тогда и только тогда, когда формула $(A \equiv B)$ является тавтологией.

Правило *одновременной подстановки*, сформулированное чуть ниже, представляет собой очень простое средство порождения тавтологий.

Если X — тавтология, p — высказывание и A — формула, то выражение, получаемое из X в результате одновременной замены всех вхождений p на A в X , является тавтологией.

1.1.7. Алгоритмическая точка зрения

В предыдущем параграфе мы определили понятия общезначимой формулы и выполнимой формулы. В данном и следующем параграфах будут описаны алгоритмы, позволяющие эффективно распознавать выполнимость и общезначимость формул.

Задача выяснения того, является ли некоторое выражение формулой, достаточно проста. Напротив, выявление выполнимости или общезначимости формулы может оказаться довольно нудной процедурой. В § 1.1.6 мы видели, что формула, содержащая k различных высказываний, допускает 2^k интерпретаций. Заманчиво иметь более эффективный алгоритм проверки, чем последовательный просмотр всех интерпретаций. В последующих параграфах будут представлены основные результаты, полученные в этой области.

Явно или неявно различные алгоритмы используют понятие *семантического дерева*. Если дано конечное или счетное множество высказываний $P = \{p_1, \dots, p_n, \dots\}$, то семантическое дерево — это бинарное корневое дерево, удовлетворяющее следующим условиям.

- Каждая дуга помечена позитивной или негативной литерой, взятой из P .
- Литеры, которыми помечены две дуги, выходящие из одного узла, противоположны.
- Никакая ветвь не содержит более одного вхождения каждой литеры.
- Никакая ветвь не содержит пару противоположных литер.

Пример семантического дерева изображен на рис. 1.6. Если P конечно, то все соответствующие ему семантические деревья обязательно конечны. Если P бесконечно, то существуют конечные и бесконечные семантические деревья. Напомним, что бесконечное бинарное дерево непременно содержит хотя бы одну бесконечную ветвь.

Каждому узлу N семантического дерева соответствует *частичная интерпретация*, т. е. функция I_N ,

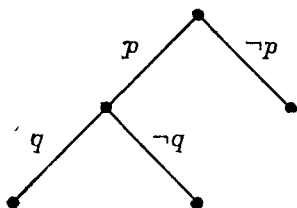


Рис. 1.6. Неполное семантическое дерево.

которая сопоставляет истинностное значение некоторым элементам из P . Частичная интерпретация I_N сопоставляет значение **И** (соответственно **Л**) высказыванию p , если некоторая дуга пути, соединяющая N с корнем, помечена литерой p (соответственно $\neg p$). Частичная интерпретация не определена для p , если ни одна из литер p и $\neg p$ не встречается на этом пути.

Конечное семантическое дерево *полно*, если каждый его *лист*¹⁾ соответствует некоторой тотальной, т. е. всюду определенной, интерпретации. Бесконечное семантическое дерево *полно*, если каждая ветвь, выходящая из корня, определяет тотальную интерпретацию. Полное семантическое дерево, соответствующее P , является конечным тогда и только тогда, когда P тоже конечно. В противном случае все его ветви бесконечны.

Для того чтобы определить, выполняма ли формула A , тривиальный алгоритм требует просмотра некоторого полного семантического дерева, соответствующего (конечному) множеству высказываний, встречающихся в A . Для каждого листа этого дерева формула A оценивается согласно соответствующей интерпретации. Формула выполняма, если по крайней мере для одного из листьев получается значение **И**. Этот алгоритм крайне неэффективен: если формула A содержит n различных высказываний, то нужно рассматривать 2^n интерпретаций.

Алгоритм Куайна [86] — довольно непосредственное усовершенствование тривиального алгоритма.

¹⁾ То есть *концевой узел*. — Прим. ред.

Если при всех возможных расширениях некоторой частичной интерпретации исследуемая формула принимает одно и то же истинностное значение, то бесполезно строить поддерево, исходящее из узла, соответствующего этой частичной интерпретации. Например, если исследуемая формула есть $(p \vee q \vee r)$, то любой путь с положительной литерой соответствует такой интерпретации, при которой эта формула истинна.

Рассмотрим применение этого алгоритма для проверки общезначимости формулы

$$[((p \wedge q) \supset r) \wedge (p \supset q)] \supset (p \supset r).$$

Начинаем с произвольного упорядочивания множества $\{p, q, r\}$ высказываний, содержащихся в формуле. Например, пусть порядок будет (p, q, r) . Это эквивалентно тому, что дуги уровня 1 (исходящие из корня) помечены литерами p или $\neg p$, дуги уровня 2 — q или $\neg q$ и дуги уровня 3 — r или $\neg r$.

Сначала рассмотрим те интерпретации, при которых p принимает значение **И**. При такой частичной интерпретации наша формула сводится к

$$[(q \supset r) \wedge q] \supset r.$$

Если q интерпретировать как **И**, то получим $(r \supset r)$, что общезначимо. Если q интерпретировать как **Л**, то имеем $(\text{Л} \supset r)$, что тоже общезначимо.

Далее рассмотрим те интерпретации, при которых p ложно. Наша формула сводится к

$$[(\text{Л} \supset r) \wedge \text{И}] \supset \text{И},$$

что всегда истинно.

Из проведенного анализа заключаем, что исследуемая формула всегда истинна, т. е. общезначима. Соответствующее семантическое дерево изображено на рис. 1.6.

Слово «истинно» и символ **И** часто употребляются вперемежку. Обычно это не причиняет неудобств и облегчает рассуждения. Разрешается также вводить семантические символы **И** и **Л** прямо в сами формулы. Это не опасно. В таком контексте **И** и **Л** можно

считать сокращениями или разновидностями записей соответственно $(p \vee \neg p)$ и $(p \wedge \neg p)$. Впрочем, некоторые авторы начинают вводить исчисление высказываний с константы **Л** (оператора без операндов) и связки \supset . Тогда $\neg A$ определяется как $(A \supset \mathbf{Л})$.

1.1.8. Алгоритм редукции

Алгоритм редукции, о котором сейчас пойдет речь, позволяет доказывать общезначимость формул с помощью приведения к абсурду. Это особенно удобно, когда формула содержит много импликаций. Для иллюстрации проверим общезначимость формулы

$$[(p \wedge q) \supset r] \supset [p \supset (q \supset r)].$$

Предположим, что при некоторой интерпретации I эта формула принимает значение **Л**. Согласно таблице истинности импликации, условное предложение ложно тогда и только тогда, когда его заключение ложно, а посылка истинна. Из этого правила получаем

$$I(p \supset (q \supset r)) = \mathbf{Л},$$

$$I((p \wedge q) \supset r) = \mathbf{И}.$$

То же самое правило, примененное к первой строке этого результата, позволяет определить I . Получаем

$$I(p) = \mathbf{И}, \quad I(q) = \mathbf{И}, \quad I(r) = \mathbf{Л},$$

что противоречит соотношению $I((p \wedge q) \supset r) = \mathbf{И}$. Это противоречие означает общезначимость исходной формулы.

Легко построить примеры, для которых алгоритмы Куайна и редукции ненамного эффективнее, чем тривиальный алгоритм. Точнее, установлено, что проблема выявления общезначимости (или выполнимости) формулы исчисления высказываний *NP-полна*. Не вдаваясь в рассуждения, выходящие за рамки этой книги, отметим, что указанный факт означает ничтожность вероятности существования эффективного общего алгоритма для проверки выполнимости или общезначимости формул исчисления высказываний.

Тем не менее в некоторых частных случаях возможно эффективное решение этой проблемы. *Метод резолюций*, который будет описан ниже, позволяет распознавать общезначимость и невыполнимость формул. В большинстве случаев этот алгоритм достаточно эффективен. К затронутому сейчас вопросу мы еще вернемся в § 1.1.12.

1.1.9. Алгебраический подход

Семантика произвольной формулы исчисления высказываний полностью определяется ее таблицей истинности. Отметим, что многие формулы могут иметь одну и ту же семантику. Например, таковы формулы $p \supset (q \supset r)$ и $(p \wedge q) \supset r$ (рис. 1.5, § 1.1.4).

Такое же явление типично и в элементарной алгебре. Например, алгебраические выражения $x(x - y)(x + y)$ и $x^3 - xy^2$ имеют «одинаковую семантику»: они представляют одинаковые полиномы (и одинаковые функции).

Для логических формул, как и для формул алгебры полиномов, важно уметь обнаруживать эквивалентность двух различно представленных объектов. Методы, используемые для этой цели, подразумевают прежде всего изучение «замечательных тождеств», которые задают различные способы представления объекта. А затем удобно бывает задать одну или пару канонических форм представления объектов. В случае с полиномами обычно используют две такие формы: произведение сомножителей минимальной степени и сумму одночленов. Замечательные тождества позволяют более или менее легко перейти от одного представления к другому.

В контексте исчисления высказываний понятие логической эквивалентности берется вместо понятия тождества. Связка \approx соответствует логической эквивалентности: $A \approx B$ означает $\models (A \equiv B)$.

Приведем первую группу основных логических эквивалентностей. Они относятся к связкам \wedge и \vee , рассматриваемым как отдельно, так и в их сочетании со связкой \supset .

Эти и последующие свойства доказываются методом таблиц истинности.

$$\begin{aligned}
 (X \wedge X) \approx X \approx (X \vee X) & \quad (\text{идемпотентность}), \\
 \left. \begin{aligned} (X \wedge Y) \approx (Y \wedge X) \\ (X \vee Y) \approx (Y \vee X) \end{aligned} \right\} & \quad (\text{коммутативность}), \\
 \left. \begin{aligned} ((X \wedge Y) \wedge Z) \approx (X \wedge (Y \wedge Z)) \\ ((X \vee Y) \vee Z) \approx (X \vee (Y \vee Z)) \end{aligned} \right\} & \quad (\text{ассоциативность}).
 \end{aligned} \tag{1.4}$$

Зададим бинарное отношение \leq на множестве формул. Выражение $A \leq B$ означает $\models (A \supset B)$. Это отношение является *отношением порядка*, так как имеют место четыре следующих свойства (они избыточны):

$$\begin{aligned}
 X &\leq X && (\text{рефлексивность}), \\
 X &\leq Y \text{ и } Y \leq X \Rightarrow X \approx Y && (\text{антисимметричность}), \\
 X &\leq Y \text{ и } Y \leq Z \Rightarrow X \leq Z && (\text{транзитивность}), \\
 \left. \begin{aligned} X &\leq Y \Leftrightarrow ((X \wedge Y) \approx X) \\ X &\leq Y \Leftrightarrow ((X \vee Y) \approx Y) \end{aligned} \right\} && (\text{элиминация}).
 \end{aligned} \tag{1.5}$$

В этом контексте наибольшая нижняя граница двух формул есть их конъюнкция, а наименьшая верхняя граница — их дизъюнкция. На обычном алгебраическом языке структура, удовлетворяющая условиям (1.4) и (1.5), называется *решеткой*. Точнее, *фактор-множество* формул относительно эквивалентности \approx является решеткой, называемой *решеткой Линденбаума* [5].

Третья группа логических эквивалентностей описывает отношения между связками конъюнкции и дизъюнкции:

$$\left. \begin{aligned} ((X \wedge Y) \vee Z) &\approx ((X \vee Z) \wedge (Y \vee Z)) \\ ((X \vee Y) \wedge Z) &\approx ((X \wedge Z) \vee (Y \wedge Z)) \end{aligned} \right\} \quad \begin{array}{l} \text{(дистрибу-} \\ \text{тивность)}. \end{array} \tag{1.6}$$

Связка «отрицание» описывается четвертой группой формул:

$$\left. \begin{aligned} (X \vee \neg X) &\approx \mathbf{И} \\ (X \wedge \neg X) &\approx \mathbf{Л} \end{aligned} \right\} \text{ (дополнительность), } \quad (1.7)$$

$$\neg \neg X \approx X \quad \text{(инволюция).}$$

Снабженная свойствами (1.6) и (1.7), решетка Линденбаума становится *булевой алгеброй*.

Если в некоторой логической эквивалентности, не содержащей связку \supset , поменять местами, с одной стороны, \wedge и \vee , а с другой стороны, **И** и **Л**, то опять получим логическую эквивалентность. В этом состоит принцип *двойственности*. Впрочем, вхождение связки \supset легко исключить, пользуясь логической эквивалентностью

$$(X \supset Y) \approx (\neg X \vee Y).$$

Укажем еще *законы де Моргана*:

$$\neg (X \wedge Y) \approx (\neg X \vee \neg Y),$$

$$\neg (X \vee Y) \approx (\neg X \wedge \neg Y).$$

Двойственность встречается иногда в другой форме: от одной формулы переходят к двойственной ей, меняя местами, с одной стороны, \wedge и \vee , с другой стороны, — **Л** и **И** и, кроме того, заменяя каждое высказывание его отрицанием. Этот второй тип двойственности используется в рамках принципа дедукции (§ 1.1.6). Необходимое и достаточное условие того, что C — логическое следствие гипотез H_1, \dots, H_n , имеет вид

$$H_1 \wedge \dots \wedge H_n \wedge \neg C \approx \mathbf{Л} \quad \text{(прямая дедукция),}$$

$$\neg H_1 \vee \dots \vee \neg H_n \vee C \approx \mathbf{И} \quad \text{(обратная дедукция).}$$

Переход от одного выражения к другому осуществляется с помощью двойственности (второго типа).

Перечисленные в этом параграфе логические эквивалентности можно задать в некоторой канонической форме представления формул, которая будет описана в следующем параграфе. Сразу же заметим, что каждая логическая формула может изучаться алгебраиче-

чески. В частности, формула A общезначима тогда и только тогда, когда $A \approx \mathbf{I}$.

Рассмотрим множество P , состоящее из конечного числа (скажем, α) высказываний. Исходя из P , можно породить бесконечно много формул. Однако существует только 2^{2^α} логически различных формул. Таким образом, алгебраический подход позволяет заменить при изучении бесконечное множество на конечное.

1.1.10. Дизъюнкты и нормальные формы

Как будет продемонстрировано в нашем изложении, бывает полезно преобразовать данную формулу в эквивалентную ей, имеющую вид «нормальной», или «канонической», формы. В этом отношении особый интерес представляют понятия «дизъюнкт» и «конъюнктивная нормальная форма».

*Дизъюнктом*¹⁾ называется дизъюнкция конечного числа литер, то есть формула вида

$$(l_1 \vee l_2 \vee \dots \vee l_n).$$

Ее часто записывают как $\vee \{l_i | i = 1, \dots, n\}$ или просто как множество литер: $\{l_i | i = 1, \dots, n\}$.

Такое представление является исключением из правила, упомянутого в конце § 1.1.6: дизъюнкт эквивалентен *дизъюнкции* (а не конъюнкции) своих элементов. По этой причине нужно каждый раз явно оговаривать, что множество литер представляет именно дизъюнкт.

Пустой дизъюнкт — единственный невыполнимый дизъюнкт. Естественно его обозначить через \mathbf{I} (иногда в литературе встречается символ \square).

Конъюнктивной нормальной формой (КНФ) называется конъюнкция конечного числа дизъюнктов.

¹⁾ В оригинале — *clause* (клауза, т.е. «статья»). В отечественной литературе широко используется термин «дизъюнкт». Его мы и применяем в данном переводе, хотя различные производные от слова *clause* мы переводим с сохранением «оригинального написания», например «неклаузальное правило резолюций» (см. § 1.1.15). — *Прим. перев.*

Теорема 1.1. *Любая формула имеет логически эквивалентную ей КНФ.*

Доказательство. Ниже приведен алгоритм нормализации.

- Сначала заменяем $(X \equiv Y)$ на $(X \supset Y) \wedge (Y \supset X)$, а затем $(U \supset V)$ — на $(\neg U \vee V)$. Это делается для исключения связок \equiv и \supset .
- Необходимое число раз применяются правила преобразования, выведенные из законов де Моргана:

$$\neg(X \wedge Y) \rightarrow (\neg X \vee \neg Y),$$

$$\neg(X \vee Y) \rightarrow (\neg X \wedge \neg Y).$$

Символ \Rightarrow означает «заменяется на». В настоящем контексте правило $(a \rightarrow b)$ означает только $(a \approx b)$. Двойные отрицания «гасятся» по закону

$$\neg \neg X \rightarrow X.$$

На этой стадии связка «отрицание» остается только непосредственно перед высказываниями.

- Наконец, необходимое число раз применяются правила преобразования, выведенные из законов дистрибутивности:

$$X \vee (Y \wedge Z) \rightarrow (X \vee Y) \wedge (X \vee Z),$$

$$(X \wedge Y) \vee Z \rightarrow (X \vee Z) \wedge (Y \vee Z).$$

Формула, полученная в конце третьего этапа, это и есть нормальная *форма*, эквивалентная исходной формуле. \square

Дизъюнкты, содержащие противоположные литеры (то есть высказывание и его отрицание), общезначимы и могут быть опущены. Можно также опускать повторения литер в пределах одного и того же дизъюнкта. Полученная таким способом нормальная форма называется *приведенной*.

Алгоритм нормализации формулы очень прост, но, вообще говоря, неэффективен.

Если в некоторой нормальной форме дизъюнкт c_i содержится в дизъюнкте c_j , то c_j можно опустить. В частности, содержащую пустой дизъюнкт нормаль-

ную форму можно заменить этим единственным дизъюнктом.

Пустая нормальная форма эквивалентна **И**, в то время как нормальная форма, содержащая только пустой дизъюнкт, эквивалентна **Л**.

Дизъюнкт общезначим тогда и только тогда, когда он содержит пару противоположных литер. Единственным невыполнимым дизъюнктом является пустой дизъюнкт **Л**.

Нормальная форма общезначима тогда и только тогда, когда все ее дизъюнкты общезначимы.

В качестве примера приведем к КНФ формулу

$$(p \supset (q \supset r)) \supset ((p \wedge s) \supset r).$$

Каждая из нижеприведенных формул эквивалентна исходной.

Первый этап (исключение импликации):

$$(p \supset (q \supset r)) \supset ((p \wedge s) \supset r),$$

$$\neg (p \supset (q \supset r)) \vee ((p \wedge s) \supset r),$$

$$\neg (\neg p \vee (q \supset r)) \vee (\neg (p \wedge s) \vee r),$$

$$\neg (\neg p \vee (\neg q \vee r)) \vee (\neg (p \wedge s) \vee r).$$

Второй этап (перенос и снятие отрицания):

$$(\neg \neg p \wedge \neg (\neg q \vee r)) \vee ((\neg p \vee \neg s) \vee r),$$

$$(\neg \neg p \wedge (\neg \neg q \wedge \neg r)) \vee ((\neg p \vee \neg s) \vee r),$$

$$(p \wedge (q \wedge \neg r)) \vee ((\neg p \vee \neg s) \vee r).$$

Третий этап (применение дистрибутивности и упрощение):

$$(p \vee ((\neg p \vee \neg s) \vee r)) \wedge ((q \wedge \neg r) \vee$$

$$\vee ((\neg p \vee \neg s) \vee r)),$$

$$(p \vee (\neg p \vee \neg s) \vee r) \wedge (q \vee (\neg p \vee \neg s) \vee r) \wedge$$

$$\wedge (\neg r \vee (\neg p \vee \neg s) \vee r),$$

$$(p \vee \neg p \vee \neg s \vee r) \wedge (q \vee \neg p \vee \neg s \vee r) \wedge$$

$$\wedge (\neg r \vee \neg p \vee \neg s \vee r),$$

$$\mathbf{И} \wedge (q \vee \neg p \vee \neg s \vee r) \wedge \mathbf{И},$$

$$(q \vee \neg p \vee \neg s \vee r).$$

Нормальная форма содержит единственный дизъюнкт $(q \vee \neg p \vee \neg s \vee r)$. Отсюда следует, что исходная формула ложна только когда p и s истинны, а q и r ложны.

Замечание. Понятие дизъюнкта важно для практики. Описание задач и алгоритмов в терминах дизъюнктов составляет основу логического программирования вообще и языка Пролог в частности (разд. 5.1 и гл. 6).

Двойственным понятием к дизъюнкту является произведение¹⁾, или конъюнкт, конечного множества литер. *Дизъюнктивная нормальная форма (ДНФ)* — дизъюнкция конечного числа конъюнктов. Можно показать, что любая формула приводится к логически эквивалентной ей ДНФ.

1.1.11. Алгоритм Девиса и Патнема

В §§ 1.1.7 и 1.1.8 были описаны алгоритмы проверки выполнимости и общезначимости логических формул. В § 1.1.10 было введено понятие эквивалентной нормальной формы и предложен (теорема 1.1) алгоритм преобразования произвольной логической формулы в эквивалентную ей нормальную форму. Цель этого параграфа состоит в описании алгоритма проверки выполнимости и общезначимости логических формул, предварительно приведенных к нормальной форме.

Алгоритм Куайна для проверки выполнимости и общезначимости формулы (§ 1.1.7) упрощается в применении к КНФ. Прежде всего, проблема общезначимости становится тривиальной: речь идет о проверке тавтологичности каждого дизъюнкта. Проблема выполнимости более интересна. Предположим для простоты, что проверяемая КНФ — приведенная. Будем применять классические теоретико-множественные обозначения.

Пусть p — некоторое высказывание. Приведенная нормальная форма S разбивается на S_p (дизъюнкты,

¹⁾ Встречаются также термины «конъюнкция», «импликант», «импликанта» и «грань». В оригинале — *sibe*. — Прим. ред.

содержащие p), $S_{\neg p}$ (дизъюнкты, содержащие $\neg p$) и $S \setminus (S_p \cup S_{\neg p})$ (остальные дизъюнкты).

Применительно к нормальным формам алгоритм Куайна сводится к следующим правилам.

Если $S = \phi$, то S выполнима.

Если $S = \{ \perp \}$, то S невыполнима.

В остальных случаях:

- выбираем высказывание p , входящее в S ,
- находим S_p , $S_{\neg p}$ и $S'' = S \setminus (S_p \cup S_{\neg p})$,
- получаем S'_c , дизъюнкты которого — дизъюнкты из S_p , лишенные литеры p ,
- получаем $S'_{\neg p}$, дизъюнкты которого — дизъюнкты из $S_{\neg p}$, лишенные литеры $\neg p$.

Множество S невыполнимо тогда и только тогда, когда невыполнимы два множества $(S'_p \cup S'')$ и $(S'_{\neg p} \cup S'')$.

Замечание. Этот алгоритм *рекурсивен*: проблема выполнимости, поставленная для множества S , сводится к такой же проблеме для двух более простых множеств, уже не содержащих высказывания p .

Следующая лемма гарантирует корректность описанного алгоритма.

Лемма 1.2. Если p истинно, то форма S эквивалентна $(S'_{\neg p} \cup S'')$. Если p ложно, то S эквивалентна $(S'_p \cup S'')$.

Доказательство. Если p истинно, то все дизъюнкты с p тоже истинны и могут быть опущены. Можно вычеркнуть ложную литеру $\neg p$ из каждого содержащего ее дизъюнкта. Эти упрощения преобразуют $S = (S_p \cup S_{\neg p} \cup S'')$ в $(S'_{\neg p} \cup S'')$. Случай, когда p — ложно, вполне аналогичен. \square

Замечание. Осуществление этого алгоритма естественно изолировать бинарным деревом, на котором проверяются следующие условия.

- Следующий (по дуге p) за узлом, помеченным S , узел помечен выражением $(S'_{\neg p} \cup S'')$.
- Следующий (по дуге $\neg p$) за узлом, помеченным S , узел помечен выражением $(S'_p \cup S'')$.

Очевидно, что это дерево — семантическое, хотя

различные высказывания не обязательно следуют в одном и том же порядке на каждой ветви.

В качестве примера применим упрощенный алгоритм Куайна для доказательства невыполнимости множества

$$S = \{p \vee q, p \vee r, \neg q \vee \neg r, \neg p\}.$$

Выбирая высказывание p , получаем

$$S_p = \{p \vee q, p \vee r\}, \quad S_{\neg p} = \{\neg p\},$$

$$S'' = \{\neg q \vee \neg r\},$$

$$S'_p = \{q, r\}, \quad S'_{\neg p} = \{\mathbf{J}\},$$

$$(S'_p \cup S'') = \{q, r, \neg q \vee \neg r\},$$

$$(S'_{\neg p} \cup S'') = \{\mathbf{J}, \neg q \vee \neg r\}.$$

Очевидно, что множество $(S'_{\neg p} \cup S'')$ невыполнимо. Таким образом, достаточно доказать невыполнимость множества $U = (S'_p \cup S'')$. Выбирая высказывание q , получаем

$$U_q = \{q\}, \quad U_{\neg q} = \{\neg q \vee \neg r\}, \quad U'' = \{r\},$$

$$U'_q = \emptyset, \quad U'_{\neg q} = \{\neg r\},$$

$$(U'_q \cup U'') = \{\mathbf{J}, r\}, \quad (U'_{\neg q} \cup U'') = \{\neg r, r\}.$$

Множества $\{\mathbf{J}, r\}$ и $\{\neg r, r\}$ невыполнимы. Проверка закончена.

Алгоритм Девиса и Патнема базируется на использовании нормальной формы исследуемой формулы — для выбора высказываний в оптимальном порядке. При этом не строятся бесполезные ветви семантического дерева [14]. Приоритетный выбор p удобен в двух следующих случаях:

- S содержит один из дизъюнктов с литерой p или $\neg p$.

- Только одна из литер p и $\neg p$ входит в S . Например, предположим, что S содержит дизъюнкт $\{p\}$. Множество $(S'_p \cup S'')$ будет содержать пустой дизъюнкт и, следовательно, будет невыполнимым. Таким образом, возможная невыполнимость для S сводится к невыполнимости для $(S'_{\neg p} \cup S'')$. С другой

стороны, если p входит в S , тогда как $\neg p$ не входит, то S'_p пусто. Возможная невыполнимость для S сводится к невыполнимости для $(S'_p \cup S'')$.

Эта простая стратегия сильно увеличивает эффективность алгоритма. Всякий раз, когда она применима, она сводит проблему невыполнимости для S к проблеме невыполнимости для единственного множества S' , более простого, чем S .

Позже мы вернемся к понятию рекурсивного алгоритма. Отметим здесь, что слово «рекурсивный» имеет много смыслов (понятие рекурсивного множества уже было затронуто в § 1.1.3).

В качестве примера интересно установить общезначимость одного расхожего метода доказательства, называемого доказательством разбором случаев. Обозначим гипотезу через h , два возможных случая — через p и q , а заключение — через c . Требуется доказать

$$\{h, h \supset (p \vee q), p \supset c, q \supset c\} \models c.$$

Это сводится к доказательству невыполнимости множества

$$\{h, h \supset (p \vee q), p \supset c, q \supset c, \neg c\}.$$

Представим элементы этого множества в виде дизъюнктов и сделаем следующие преобразования:

$$1. \{\underline{h}, \neg h \vee p \vee q, \neg p \vee c, \neg q \vee c, \neg c\}.$$

$$2. \{p \vee q, \neg p \vee c, \neg q \vee c, \underline{\neg c}\}.$$

$$3. \{p \vee q, \underline{\neg p}, \neg q\}.$$

$$4. \{\underline{q}, \neg q\}.$$

$$5. \{\perp\}.$$

Здесь каждая строка соответствует одному прохождению цикла нашего алгоритма. Выбираемая литера подчеркнута.

Замечание. Сейчас самое время подчеркнуть в связи с этим примером ограниченность формальной

логики. На самом деле мы установили общезначимость *формальной модели* принципа доказательства разбором случаев. Обоснованность же соответствия между самим принципом (философским понятием) и его моделью (понятием формальной логики) лежит за пределами формальной логики.

1.1.12. Принцип резолюций

Не существует общего, по-настоящему эффективного критерия для проверки выполнимости КНФ. Тем не менее есть достаточно удобный метод для выявления невыполнимости множества дизъюнктов. Действительно, множество дизъюнктов невыполнимо тогда и только тогда, когда пустой дизъюнкт \perp является логическим следствием из него (§ 1.1.6). Таким образом, невыполнимость множества S можно проверить, порождая логические следствия из S до тех пор, пока не получим пустой дизъюнкт.

Для порождения логических следствий будет использоваться очень простая схема рассуждений. Пусть A , B и X — формулы. Предположим, что две формулы $(A \vee X)$ и $(B \vee \neg X)$ — истинны. Если X тоже истинна, то отсюда можно заключить, что B истинна. Наоборот, если X ложна, то можно заключить, что A истинна. В обоих случаях $(A \vee B)$ истинна. В обозначениях § 1.1.6 получается правило

$$\{A \vee X, B \vee \neg X\} \models A \vee B,$$

которое можно записать также в виде

$$\{\neg X \supset A, X \supset B\} \models A \vee B.$$

В том частном случае, когда X — высказывание, а A и B — дизъюнкты, это правило называется *правилом резолюций*. Очень подробное обсуждение его содержится в [14]. Общезначимость правила резолюций выражается следующей леммой.

Лемма 1.3. Пусть s_1 и s_2 — дизъюнкты нормальной формы S , l — литера. Если $l \in s_1$ и $\neg l \in s_2$, то дизъюнкт $r = (s_1 \setminus \{l\}) \cup (s_2 \setminus \{\neg l\})$ является логическим следствием нормальной формы S .

Следствие. Нормальные формы S и $S \cup \{r\}$ эквивалентны.

Замечание. Дизъюнкт r называется *резольвентой* дизъюнктов s_1 и s_2 .

1.1.13. Доказательства невыполнимости, основанные на принципе резолюций

Доказательства невыполнимости логических формул очень важны в ИИ (гл. 5). Способы таких доказательств, основанные на принципе резолюций, выделяются среди прочих тем, что они дают возможность использовать средства автоматического доказательства, применяемые в логическом программировании (гл. 6).

При всей простоте метод резолюций (или, короче, резолюция) достаточен для обнаружения возможной невыполнимости множества приведенных дизъюнктов. Таким образом, можно попробовать доказать невыполнимость конечного множества дизъюнктов из S с помощью следующего алгоритма.

При условии, что $\Pi \notin S$,
выбираем l, s_1, s_2 , такие, что $l \in s_1$ и $\neg l \in s_2$;
вычисляем резольвенту r ;
заменяем S на $S \cup \{r\}$.

В качестве примера перепроверим невыполнимость множества

$$S = \{p \vee q, p \vee r, \neg q \vee \neg r, \neg p\},$$

уже установленную в § 1.1.11. Дизъюнкты удобно пронумеровать. Получится следующий список:

1. $p \vee q$
2. $p \vee r$
3. $\neg q \vee \neg r$
4. $\neg p$

Затем вычисляются и добавляются к S резольвенты. В списке, который приводится ниже, каждый дизъюнкт — резольвента некоторых из предшествующих

дизъюнктов. Номера их приведены в скобках права от соответствующей резольвенты.

5. $p \vee \neg r$ (1,3)
6. q (1,4)
7. $p \vee \neg q$ (2,3)
8. r (2,4)
9. p (2,5)
10. $\neg r$ (3,6)
11. $\neg q$ (3,8)
12. $\neg r$ (4,5)
13. $\neg q$ (4,7)
14. \bot (4,9)

Замечание. Алгоритм проверки невыполнимости — недетерминированный: вообще говоря, возможен не один выбор для l , s_1 и s_2 . В приведенном примере мы выбирали дизъюнкты s_1 и s_2 в лексикографическом порядке номеров. Такая стратегия неоптимальна. Некоторые резольвенты оказались ненужными, а также вычислялись в некоторых случаях более одного раза. Для сравнения продемонстрируем теперь применение этого же алгоритма с минимумом резольвент:

5. q (1,4)
6. r (2,4)
7. $\neg q$ (3,6)
8. \bot (5,7)

Ясно, что принятая стратегия может значительно влиять на процесс выполнения алгоритма. Тем не менее упомянем два важных свойства, не зависящих от используемой стратегии.

Прежде всего, если множество S не содержит ни одной пары дизъюнктов, допускающих резольвенту, то оно выполнимо (разумеется, за исключением случая, когда оно содержит пустой дизъюнкт). Интерпретация I получается заданием $I(p) = \mathbf{И}$ тогда и только тогда, когда положительная литера p принадлежит одному из дизъюнктов множества S .

Во-вторых, если выполнение этого алгоритма закончилось «нормально» после порождения пустого дизъюнкта, то установлена невыполнимость исходного

множества S . Это непосредственное следствие леммы 1.3.

Может случиться, что выполнение алгоритма не завершится, хотя число различных дизъюнктов, которые могут быть порождены с помощью резолюций, очевидно, конечно. Например, такое явление имеет место для множества $\{p, \neg p \vee q\}$. Резольвентный дизъюнкт q будет порождаться неограниченное число раз. Рассмотрим также случай невыполнимого множества $\{p, \neg p \vee q, \neg q\}$. Оптимальная стратегия установит невыполнимость посредством построения резольвент q , а затем \perp . Напротив, «неадекватная» стратегия приведет к заикливанию, которое только что было отмечено. Таким образом, можно констатировать, что стратегия влияет не только на эффективность алгоритма, но и на его завершаемость.

Представляет интерес свойство *завершаемости* метода резолюций: конечное множество S невыполнимо тогда и только тогда, когда пустой дизъюнкт *может* быть выведен из S с помощью резолюций. Из леммы 1.3 следует достаточность этого условия. Пустой дизъюнкт, будучи невыполнимым, не может быть логическим следствием из выполнимого множества дизъюнктов. Необходимость сформулированного условия отражена в следующей лемме [14].

Лемма 1.4. *Если множество дизъюнктов S невыполнимо и содержит резольвенты своих элементов, то оно обязательно содержит пустой дизъюнкт.*

Замечание. Ясно, что число различных дизъюнктов, которые можно составить на основе конечного множества S , конечно. Таким образом, лемма 1.4 дает простое общее средство решения вопроса о выполнимости или невыполнимости конечного множества дизъюнктов.

Мы не приводим формального доказательства леммы 1.4, но проиллюстрируем ее одним очень простым примером. Рассмотрим множество дизъюнктов

$$S = \{p, \neg p \vee \neg q, \neg p \vee r, q\}.$$

Первый этап состоит в построении семантического дерева и помечивании каждого листа дизъюнктом, делающим ложной интерпретацию, соответствующую этому листу (§ 1.1.7). Это возможно тогда и только тогда, когда S невыполнимо. Здесь именно такой случай и подходящее дерево изображено на рис. 1.7. Второй этап состоит в том, что каждому узлу N дерева сопоставляется некоторый дизъюнкт s . Этот дизъюнкт будет зависеть от дизъюнктов c_1 и c_2 , которые уже сопоставлены двум узлам, следующим за N . Две соответствующие дуги помечены двумя противоположными литерами, скажем l и $\neg l$. Если один из дизъюнктов, например c_1 , не содержит ни l , ни ее отрицание, то можно выбрать $s \equiv c_1$, в противном случае s будет резольвентой для c_1 и c_2 по отношению к l .

Этот результат, полученный для множества S , иллюстрируется рис. 1.8. Заметим, что каждый узел

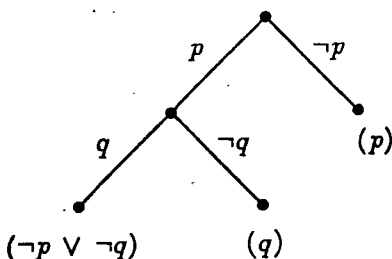


Рис. 1.7. Пример, первый этап.

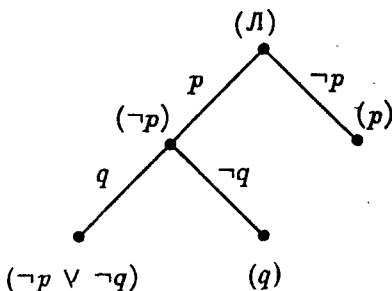


Рис. 1.8. Пример, второй этап.

помечен дизъюнктом, при котором ложна частичная интерпретация, соответствующая этому узлу (§ 1.1.7). Таким образом, корень будет помечен пустым дизъюнктом. Наконец, по построению каждый дизъюнкт является элементом S или получен из S с помощью резолюции. Число резольвент, которые надо вычислить для установления невыполнимости множества дизъюнктов S , не больше числа невисячих узлов семантического дерева. Максимальное значение этой величины равно 2^{n-1} , где n — число различных высказываний, входящих в S . Тем самым показано, что резолюция неэффективна, если применяется только что описанная простая стратегия. Отметим, что некоторые стратегии выбора приводят к лучшим реализациям [14].

Лемма 1.4 остается справедливой для бесконечного S . Отсюда следует, что бесконечное множество дизъюнктов невыполнимо тогда и только тогда, когда в нем найдется конечное невыполнимое подмножество. К этому вопросу мы еще вернемся в § 1.1.18. Приведем непосредственное следствие, отражающее свойство завершаемости принципа резолюций.

Следствие 1.5. *Если множество S формул невыполнимо, то этот факт всегда можно установить методом резолюций.*

1.1.14. Приложения и примеры использования метода резолюций

Метод резолюций применяется в качестве механизма доказательства при реализации принципа дедукции, упомянутого в § 1.1.6. Для доказательства того, что некое заключение C является логическим следствием множества гипотез H_1, \dots, H_n , нужно применить резолюцию к множеству $\{H_1, \dots, H_n, \neg C\}$. Эти гипотезы и отрицание заключения должны иметь вид дизъюнктов.

Резолюция — еще и такой механизм, который служит основой для осуществления инструкций на языке Пролог (разд. 5.1 и гл. 6).

Будет интересно вновь рассмотреть задачу о доказательстве разбором случаев, встречающуюся в конце § 1.1.11. Множество дизъюнктов, невыполнимость которого следует установить, таково:

$$\{h, \neg h \vee p \vee q, \neg p \vee c, \neg q \vee c, \neg c\}.$$

Исходя из этого множества, можно следующим образом получить пустой дизъюнкт:

- | | |
|---------------------------|-----------------------|
| 1. h | гипотеза |
| 2. $\neg h \vee p \vee q$ | гипотеза |
| 3. $\neg p \vee c$ | гипотеза |
| 4. $\neg q \vee c$ | гипотеза |
| 5. $\neg c$ | отрицание заключения |
| 6. $p \vee q$ | резольвента для 1 и 2 |
| 7. $\neg p$ | резольвента для 3 и 5 |
| 8. $\neg q$ | резольвента для 4 и 5 |
| 9. q | резольвента для 6 и 7 |
| 10. \bot | резольвента для 8 и 9 |

Всякое доказательство, формальное или неформальное, является последовательностью элементарных рассуждений, последнее из которых и есть заключение (в данном случае это \bot , потому что речь идет об опровержении). Однако взаимосвязь между различными этапами доказательства есть не что иное, как отношение частичного порядка. Таким образом, естественно, хотя менее удобно, представлять доказательство деревом, а не последовательностью. В случае доказательства с помощью метода резолюций это дерево будет семантическим. Дерево нашего примера изображено на рис. 1.9. Чтобы уменьшить громоздкость, узлы помечены номерами дизъюнктов, а не самими дизъюнктами.

Замечание. В таком контексте семантическое дерево часто изображают корнем вниз.

Правило *согласия*¹⁾ двойственно правилу резолюций. Если его применить к конъюнкциям $p \wedge c_1$ и $\neg p \wedge c_2$, то получим конъюнкцию $c_1 \wedge c_2$. Эта операция была введена в рамках булевой алгебры

¹⁾ В оригинале — *consensus* (лат.) — Прим. ред.

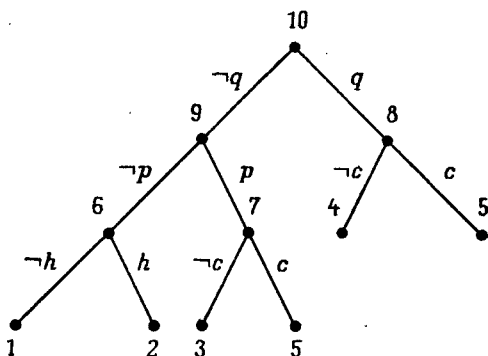


Рис. 1.9. Дерево доказательств.

применительно к синтезу логических схем [109]. Она позволяет выявлять общезначимость ДНФ.

1.1.15. Неклаузальное правило резолюций

При использовании метода резолюций для доказательства невыполнимости требуется, чтобы логические формулы, к которым применяется правило резолюций, были приведены к КНФ (§ 1.1.10). Это преобразование, предшествующее доказательству, может оказаться довольно трудоемким. Неклаузальная резолюция предназначена для распространения механизма доказательства на произвольные логические формулы. В частности, это позволяет использовать резолюцию как средство в доказательствах, проводимых по принципу дедукции, даже если гипотезы и отрицание заключения не являются дизъюнктами.

Пусть A_1 и A_2 — формулы исчисления высказываний. Введем определения

$$\top_p(A_1, A_2) =_{\text{def}} A_1^p := \perp \wedge A_2^p := \text{н},$$

$$\perp_p(A_1, A_2) =_{\text{def}} A_1^p := \perp \vee A_2^p := \text{н}.$$

Символом $A^p := q$ обозначена формула, полученная заменой всех вхождений p в A на q . Оператор \top употребляется для доказательства общезначимости, а \perp — невыполнимости (см. [107] по поводу правила согласия и [81] по поводу правила резолюций). Лег-

ко проверить, что, если A_1 и A_2 — конъюнкции (дизъюнкты), то оператор \top (\perp) — классическое правило согласия (резолуций).

Рассмотрим простой пример на применение неклаузуальной резолюции. Попробуем доказать невыполнимость множества формул

$$\{A_1 : (p \wedge r) \vee (q \wedge \neg r), \\ A_2 : (\neg p \wedge r) \vee (\neg q \wedge \neg r)\}.$$

Доказательство таково:

1. $(p \wedge r) \vee (q \wedge \neg r)$	A_1
2. $(\neg p \wedge r) \vee (\neg q \wedge \neg r)$	A_2
3. $(q \wedge \neg r) \vee (\neg q \wedge \neg r)$	\perp_p (1,2)
4. $(p \wedge r) \vee (\neg p \wedge r)$	\perp_q (1,2)
5. \perp	\perp_r (4,3)

Эти формулы либо гипотезы, либо получены с помощью правила \perp из предыдущих формул (номера которых указаны в скобках).

Заметим еще, что операторы \top и \perp содержатся в понятии *булева вывода* (дизъюнктивного и конъюнктивного), описанного в [106].

Неклаузуальное правило резолюции используется в § 3.1.20 в примере рассуждений по поводу знаний.

1.1.16. Хорновские дизъюнкты

Хотя метод резолюций в наибольшей общности неэффективен, он эффективен в некоторых частных случаях. Важнейшим из них является случай хорновских дизъюнктов, т. е. таких, которые содержат не более одной позитивной литеры¹⁾. Дизъюнкт

$$\{\neg p, \neg q, \neg r, s\}$$

эквивалентен импликации

$$(p \wedge q \wedge r) \supset s.$$

Хорновский дизъюнкт называется *точным*, если он содержит позитивную литеру. В противном случае он

¹⁾ Для некоторых авторов хорновские дизъюнкты — это те, которые содержат не более одной негативной литеры. Мы здесь придерживаемся наиболее распространенного мнения.

называется *негативным*. Точный дизъюнкт нередко выражает некоторое *правило*: негативные литеры соответствуют гипотезам (которые представлены соответствующими высказываниями), а позитивная литера представляет заключение. *Унитарный позитивный* дизъюнкт, сводящийся к одиночной позитивной литере, представляет некоторый *факт*, т. е. заключение, не зависящее ни от каких гипотез.

Часто задача состоит в том, что надо проверить какую-то формулу (называемую *целью*), логически выведенную из множества фактов и правил. Резолюция является методом доказательства от противного: исходя из фактов, правил и отрицания цели, приходим к противоречию (пустому дизъюнкту). Этот метод (наряду с понятиями факта, правила и цели) будет использоваться в рассуждениях по поводу знаний в §§ 3.1.21—3.1.23 и 3.3.8—3.3.11.

Рассмотрим множество S хорновских дизъюнктов (без тавтологий). Выполнимость S можно проверить с помощью следующего алгоритма.

При условии $\mathcal{L} \notin S$,

выбираем p и c , такие, что:

- p — унитарный позитивный дизъюнкт из S ,
- c — дизъюнкт из S , содержащий $\neg p$;

вычисляем резольвенту r ;

заменяем S на $(S \setminus \{c\}) \cup \{r\}$.

Этот алгоритм, применимый только к множеству хорновских дизъюнктов, очень похож на общий алгоритм из § 1.1.13. Принципиальное отличие состоит в том, что на каждом этапе некая литера удаляется из одного дизъюнкта. Действительно, в только что введенных обозначениях дизъюнкт r есть не что иное, как дизъюнкт c , из которого была удалена литера $\neg p$. Отсюда сразу же следует, что выполнение алгоритма всегда завершается, какая бы стратегия ни была принята при выборе p и c . Если N — число литер, первоначально присутствующих в S (с учетом повторений), то цикл будет выполняться не более N раз. Таким образом, сложность этого алгоритма порядка N^2 .

1.	$p \vee \neg r \vee \neg t$	\underline{q}	r	$t \vee \neg p \vee \neg r$	$t \vee \neg q$	$\neg p \vee \neg q \vee \neg r$
2.	$p \vee \neg r \vee \neg t$	\underline{q}	\underline{r}	$t \vee \neg p \vee \neg r$	t	$\neg p \vee \neg q \vee \neg r$
3.	$p \vee \neg t$	\underline{q}	r	$t \vee \neg p \vee \neg r$	t	$\neg p \vee \neg q \vee \neg r$
4.	$p \vee \neg t$	\underline{q}	\underline{r}	$t \vee \neg p \vee \neg r$	t	$\neg p \vee \neg r$
5.	$p \vee \neg t$	\underline{q}	r	$t \vee \neg p \vee \neg r$	\underline{t}	$\neg p$
6.	\underline{p}	\underline{q}	r	$t \vee \neg p \vee \neg r$	t	$\neg p$
7.	\underline{p}	\underline{q}	r	$t \vee \neg p \vee \neg r$	t	Π

Рис. 1.10. Проверка выполнимости.

Имеются две возможности, которыми может закончиться этот алгоритм: либо порожден пустой дизъюнкт, либо получено множество S , уже не содержащее дизъюнктов, равных p и s .

Рассмотрим первую возможность. Множества S и $S \cup \{r\}$ (в обозначениях нашего алгоритма) эквивалентны: это верно в общем случае (§§ 1.1.12—1.1.13). Однако дизъюнкт s является здесь логическим следствием резольвенты r и, значит, может быть опущен. Это показывает, что порождаются только логические следствия из S . Таким образом, появление пустого дизъюнкта означает невыполнимость S .

При второй возможности S обязательно выполнимо. Обозначим через T множество дизъюнктов, полученных в результате выполнения алгоритма. Моделью для S (и для T) является такая интерпретация I , что $I(p) = \text{И}$ тогда и только тогда, когда унитарный позитивный дизъюнкт p находится в T . Это каноническая модель для S .

Для примера применим этот алгоритм к проверке невыполнимости множества

$$S = \{p \vee \neg r \vee \neg t, q, r, t \vee \neg p \vee \neg r, \\ t \vee \neg q, \neg p \vee \neg q \vee \neg r\}.$$

Каждая строка в таблице на рис. 1.10 представляет один из выполняемых этапов. Выбираемый унитарный позитивный дизъюнкт и соответствующая ему негативная литера подчеркнуты.

Недавно был разработан алгоритм, имеющий линейную сложность относительно N [25].

Множества хорновских дизъюнктов обладают одной особенностью, которую стоит отметить. Если множество P высказываний фиксировано, то любая интерпретация I задается подмножеством P_I высказываний, которым она сопоставляет значение И. Пересечение $\bigcap (\mathcal{I})$ множества интерпретаций \mathcal{I} определяется равенством

$$P_{\bigcap (\mathcal{I})} = \bigcap_{I \in \mathcal{I}} P_I.$$

Пусть \mathcal{M} — множество моделей для конечного множества S произвольных дизъюнктов. Если интерпретация $M = \bigcap (\mathcal{M})$ — тоже модель для S , то она называется *минимальной моделью* для S . (Если \mathcal{M} пусто, то $P_M = P$.) В общем случае пересечение двух моделей не является моделью. Например, множество S , состоящее из единственного дизъюнкта $p \vee q$, допускает три модели I , J и K , такие, что $P_I = \{p, q\}$, $P_J = \{p\}$ и $P_K = \{q\}$. Однако $J \cup K$ не модель для $p \vee q$.

Заметим сразу, что каноническая модель для полного множества S хорновских дизъюнктов — это в точности минимальная модель для S . Тем самым установлен следующий результат.

Теорема 1.6. *Любое конечное выполнимое множество хорновских дизъюнктов допускает одну и только одну минимальную модель.*

Этот результат интересен по двум причинам. Прежде всего нахождение минимальной модели означает, что истинными считаются только явно сформулированные факты (высказывания) и логические следствия, получаемые из них согласно правилам. Это положение соответствует гипотезе «замкнутого мира», широко используемой в области баз данных. Кроме того, в некоторой мере это узаконивает интересное понятие отрицания, используемое в логическом программировании (гл. 6).

С другой стороны, предположим, что множество S уже было исследовано и что минимальная модель M уже найдена. Рассматривая множество S' , получаемое присоединением к S новых фактов и новых

правил, видим, что имеются две возможности. Во-первых, очевидно, что полученное множество может оказаться невыполнимым. Во-вторых, если оно останется выполнимым, то его минимальная модель M' является расширением модели M .

В общем случае, напротив, присоединение новых дизъюнктов делает бесполезной всю уже выполненную работу. Например, множество S , состоящее из единственного дизъюнкта $p \vee q \vee \neg r$, допускает пустую минимальную модель, но присоединение дизъюнкта r дает такое множество, которое, будучи выполнимым, уже не допускает минимальной модели.

1.1.17. Хорновские дизъюнкты и КС-грамматики

Здесь мы примем новую форму записи хорновских дизъюнктов. Например,

$$\{\neg p, \neg q, \neg r, s\}$$

будем записывать как

$$s :- p, q, r.$$

При такой записи появляются интересные аналогии. Прежде всего, хорновский дизъюнкт можно уподобить правилу переписывания, а множество хорновских дизъюнктов — некоторой КС-грамматике¹⁾ (§§ 5.1.2, 5.1.3). Символами будут высказывания и выделенный символ \mathbf{J} . Все они нетерминальные. Рассмотрим пример. Пусть множество дизъюнктов представлено следующей грамматикой:

1. $p :- q, r, s.$
2. $p :- r, t.$
3. $q :-.$
4. $r :-.$
5. $t :- p, r.$
6. $t :- q.$
7. $\mathbf{J} :- p, q, r.$

Это множество невыполнимо тогда и только тогда,

¹⁾ Контекстно-свободной свободно-контекстной, бесконтекстной и т. п. — Прим. перев.

когда пустая цепочка ε может быть получена с помощью правил грамматики (правил переписывания) исходя из символа **Л**. (Эти понятия детально разъяснены в разд. 5.1 и 5.2.) Здесь именно такой случай, что подтверждается следующей цепочкой переписываний:

Л (7) p, q, r (2) r, t, q, r (4) t, q, r (6) q, q, r (3) $q, r, (3) r$ (4) ε ¹⁾.

Это, очевидно, просто иная запись нижеследующего логического вывода:

- | | |
|---|-------------------------|
| 1. $\neg p \vee \neg q \vee \neg r$ | дизъюнкт цели. |
| 2. $p \vee \neg r \vee \neg t$ | правило. |
| 3. $\neg r \vee \neg t \vee \neg q \vee \neg r$ | резольвента для 1 и 2. |
| 4. r | факт. |
| 5. $\neg t \vee \neg q \vee \neg r$ | резольвента для 3 и 4. |
| 6. $t \vee \neg q$ | правило. |
| 7. $\neg q \vee \neg r$ | резольвента для 5 и 6. |
| 8. q | факт. |
| 9. $\neg r$ | резольвента для 7 и 8. |
| 10. r | факт. |
| 11. Л | резольвента для 9 и 10. |

Наличие нескольких дизъюнктов с общей позитивной литерой влечет за собой *недетерминированность* этой грамматики: одно и то же выражение может быть переписано многими способами.

Другое толкование указанного подхода состоит в том, что литеры рассматриваются как (вызываемые) процедуры. В этом состоит основа логического программирования (разд. 5.1 и гл. 6).

Для сравнения разберем тот же пример, пользуясь алгоритмом из предыдущего параграфа:

- | | |
|-------------------------------------|------------------------|
| 1. $\neg p \vee \neg q \vee \neg r$ | дизъюнкт цели. |
| 2. r | факт. |
| 3. $\neg p \vee \neg q$ | резольвента для 1 и 2. |
| 4. q | факт. |
| 5. $\neg p$ | резольвента для 3 и 4. |

¹⁾ Это читается так: перепись от **Л** по (7) дает p, q, r , перепись по (2) дает r, t, q, r и т. д.

6. $p \vee \neg r \vee \neg t$	правило.
7. r	факт.
8. $p \vee \neg t$	резольвента для 6 и 7.
9. $t \vee \neg q$	правило.
10. t	резольвента для 4 и 9.
11. p	резольвента для 8 и 10.
12. \perp	резольвента для 5 и 11.

1.1.18. Теорема компактности

Фундаментальная проблема логики — выявление невыполнимости множества дизъюнктов, или, шире, множества формул (§ 1.1.6). Мы видели, что метод резолюций решает эту проблему для конечного множества дизъюнктов или формул (§ 1.1.12—1.1.13). В данном параграфе этот результат обобщается на случай бесконечного множества.

Введем для начала два полезных понятия. Будем здесь предполагать фиксированным (конечное или бесконечное) множество P базовых (исходных, примитивных) высказываний. Пусть E — множество формул, которые можно построить из элементов множества P , S — подмножество из E . S называется *финитно выполнимым*, если все его конечные подмножества выполнимы. S называется *максимальным*, если оно финитно выполнимо и, кроме того, для любой формулы A из E оно содержит либо A , либо $\neg A$.

Существует тесная связь между интерпретациями и максимальными финитно выполнимыми множествами. Прежде всего, любая интерпретация I естественным образом задает максимальное (финитно) выполнимое множество S_I согласно правилу

$$A \in S_I \Leftrightarrow I(A) = \text{И}.$$

Обратное гораздо интересней и составляет предмет следующей леммы.

Лемма 1.7. *Любое максимальное финитно выполнимое множество допускает одну и только одну модель.*

Доказательство. Пусть S — максимальное финитно выполнимое множество. Зададим интерпретацию I

следующим образом. Для каждого элемента p из P положим $I(p) = \mathbf{И}$ тогда и только тогда, когда $p \in S$.

Прежде всего, если J — модель для S , то $J = I$. Действительно, для любого высказывания p должно выполняться $J(p) = I(p) = \mathbf{И}$, если $p \in S$. Кроме того, должно выполняться $J(p) = I(p) = \mathbf{Л}$, если $p \notin S$, так как в этом случае $\neg p \in S$.

С другой стороны, если A — формула из S , то $I(A) = \mathbf{И}$. Действительно, пусть Q^+ — множество высказываний, входящих в A , Q^- — множество соответствующих негативных литер. Пусть R^+ и R^- — соответственно пересечения этих множеств с S . Множество $(R^+ \cup R^- \cup \{A\})$ конечное и вложено в S , следовательно, оно выполнимо. По построению, его модели точно такие же, как для $(R^+ \cup R^-)$. Так как I — одна из этих моделей, то $I(A) = \mathbf{И}$ \square

Замечание. Любое максимальное финитно выполнимое множество выполнимо.

А теперь — основной результат.

Теорема 1.8. *Множество формул S выполнимо тогда и только тогда, когда все его конечные подмножества выполнимы.*

Доказательство. Необходимость этого утверждения очевидна. Для доказательства достаточности можно ограничиться построением некоторой интерпретации для S , или, что то же самое, максимального выполнимого множества, содержащего S . Рассмотрим здесь общий случай, когда рассматриваемое множество P счетное. При этом множество формул E тоже счетное. Пусть $A_1, A_2, \dots, A_n, \dots$ — некоторая нумерация E . Следующим рекуррентным образом зададим последовательность подмножеств E_n в E :

- $E_0 = S$,
- если все конечные подмножества объединения $E_n \cup \{A_n\}$ выполнимы, то задаем $E_{n+1} = E_n \cup \{A_n\}$, иначе задаем $E_{n+1} = E_n \cup \{\neg A_n\}$.

Индукцией по n легко доказать, что для любого n все конечные подмножества в E_n выполнимы.

- По условию это верно для E_0 .

Предположим, что все конечные подмножества в E_n выполнимы. Установим это и для E_{n+1} .

— По построению это очевидно для $E_{n+1} = E_n \cup \{A_n\}$.

— Если $E_{n+1} = E_n \cup \{\neg A_n\}$, то обязательно существует такое конечное подмножество E' в E_n , что $E' \cup \{A_n\}$ невыполнимо. Пусть E'' — произвольное конечное подмножество в E_{n+1} . Ввиду индуктивного предположения конечное множество $E' \cup E'' \setminus \{\neg A_n\}$ выполнимо. Пусть I — модель для этого множества. Тогда $I(A_n) = \mathbf{И}$ невозможно из-за невыполнимости $E' \cup \{A_n\}$. Следовательно, $I(A_n) = \mathbf{Л}$, I — модель для $E'' \cup E''$, а значит, и для E'' .

Объединение всех E_n максимально. Любое конечное подмножество входит в одно из E_n и, следовательно, выполнимо. Таким образом, это объединение является максимальным финитно выполнимым множеством. Его (единственная) модель будет также моделью для S . \square

Мы предположим, что наше множество высказываний, а следовательно, и множество формул являются счетными. Однако это не обязательно. Если множество формул несчетно, но *вполне упорядочено*, то наше доказательство и к этому случаю легко приспособить: индукцию по натуральным числам надо заменить на *трансфинитную* индукцию. Действительно, любое вполне упорядоченное множество изоморфно некоторому ординалу. Если мы примем аксиому выбора (например, в форме: декартово произведение непустых множеств непусто), то даже не надо будет требовать вполне упорядоченность: в таком контексте ее допускает любое множество. Эти понятия строго изложены в [3] и особенно в [5].

Замечание. Теорема 1.8 утверждает, что из любого невыполнимого множества можно выделить конечное невыполнимое подмножество. Этот результат аналогичен следующему факту из топологии: «из любого открытого покрытия можно выделить конечное подпокрытие». Данный факт характеризует

компактные топологические пространства. Аналогия глубокая, поэтому теорема 1.8 названа «теоремой компактности» [5].

1.2. Исчисление предикатов

1.2.1. Введение

Исчисление высказываний позволяет формализовать лишь малую часть множества рассуждений. Например, рассмотрим следующее рассуждение:

Все люди смертны;
Сократ — человек;
следовательно, Сократ смертен. (1.8)

Это рассуждение правильное, но оно выходит за рамки логики высказываний. В нем содержатся три высказывания:

p : Все люди смертны,
 q : Сократ — человек,
 r : Сократ смертен.

Используя соответствующие связи, можно написать формулу

$$(p \wedge q) \supset r.$$

Эта формула необщезначима. Таким образом, логика высказываний не позволяет корректно выразить рассуждение (1.8). Причина этой неудачи довольно очевидна. Логика высказываний моделирует высказывания и функционально-истинностные связки, позволяющие комбинировать высказывания. В таком контексте высказывание — неделимый объект. Однако ясно, что в естественных языках высказывание имеет внутреннюю структуру. Иначе говоря, значение высказывания, хотя бы в первом приближении, является функцией значений его компонент.

До того как будет рассмотрен собственно семантический аспект, даже простой лексический анализ обнаруживает интересное сходство между тремя выска-

званиями приведенного в примере рассуждения: некоторые слова присутствуют в этом рассуждении более одного раза. Следовательно, можно предположить, что указанное сходство должно сохраниться в формальной версии этого рассуждения.

С другой стороны, в естественном языке часто опускают легко восполняемые слова. Текст (1.8) в этом отношении не исключение. Мы уже заметили наличие связи конъюнкции, обозначенной в тексте точкой с запятой. Первое высказывание содержит также скрытую импликацию. Его можно переписать следующим образом.

Быть человеком —
значит быть смертным.

Эта переформулировка не вполне удовлетворительна: такое впечатление, что ей соответствует формула типа $(A \supset B)$. Это не так: слово «быть» устанавливает связь между посылкой A и следствием B условного наклонения. Впрочем, заметим, что слово «Сократ» устанавливает аналогичную связь между двумя последними строками в (1.8). Между тем эти случаи различны: слово «быть» означает какой-то, пока не определенный, член некоторого универсума, тогда как «Сократ» означает конкретный член этого универсума. По аналогии с языком математики говорят, что «быть» — это *переменная* (заново обозначим ее, как это принято, через x), тогда как «Сократ» — *константа*.

Текст (1.8) можно переписать так, что будет лучше видна его реальная структура, а именно следующим образом.

Для всех x , если x является человеком,
то x является смертным;
Сократ является человеком;
(следовательно) Сократ является смертным. (1.9)

Теперь можно пополнить запас различных составляющих. В дополнение к высказываниям и связкам (неявной конъюнкции, импликации вида «если ..., ..., то» и импликации вида «следовательно») у нас есть разные новые компоненты: «для всех», « x », «Сократ».

«является человеком» и «является смертным».

Забота об утонченном анализе не должна заводить слишком далеко. Достаточно ясно, что расчленение выражения «для всех» бесполезно. Напротив, и выделение слова «является» могло бы показаться интересным, но в нашем контексте это не подтверждается, так как оно может слишком легко исчезнуть.оборот речи «является альтруистом» можно заменить на «заботится о других», тогда как «является смертным» приблизительно означает «умрет».

Такие обороты речи служат примерами *предикатных констант*. Предикатная константа не меняет своего значения истинности. Она лишь связана с подходящим числом аргументов или параметров, называемых *термами*. Это число мест зависит только от рассматриваемой предикатной константы. Последняя вместе с подходящим числом термов называется *предикатной формой*.

Выражение «для всех x » служит примером *квантификации*, которая состоит из квантора (здесь «для всех») и одной переменной (здесь x). Естественные языки содержат не только огромное число связей, но и громадное число кванторов. Заметим для начала, что отрицанием выражения

для всех x , A

очевидно будет

существует (хотя бы один) x , $\neg A$.

Таким образом, нужен квантор «существует». Другие возможные кванторы:

Почти для всех...

Ровно для одного...

Существует ровно один...

Существует не более одного...

Существует много...

Существует бесконечно много...

Существует ровно пять...

Мы ограничимся квантором «для всех» и теми кванторами, которые можно из него получить с помощью пропозициональных связей и, возможно, посредством

отношения равенства, которое будет введено далее. Уместно отметить, что в естественном языке квантификация очень часто лишь подразумевается, но явно не представляется — например во фразах: «Все люди смертны», «Некоторые люди умерли».

Теперь мы можем формализовать гипотезы и заключение рассуждения (1.9). Квантор «для всех» обозначим \forall , высказывание « c — человек» представим через $H(c)$, а « d смертен» запишем как $M(d)$. Таким образом, получим полуформальное рассуждение:

$$\forall x (H(x) \supset M(x)) \text{ и } H(\text{Сократ}), \\ \text{следовательно, } M(\text{Сократ})$$

По виду формул можно понять, что M и H — *одноместные* предикатные константы, т. е. M и H допускают единственный аргумент. Заметим также, что x — переменная (только одна переменная может следовать за квантором). Напротив, ниоткуда не следует, что «Сократ» — константа.

Теперь займемся формализацией *логики предикатов* или *исчисления предикатов*. Для начала точно опишем язык этой логики.

1.2.2. Словарь

Основными символами языка являются *переменные*, *индивидуальные константы*, *предикатные константы*, *связки* (\neg , \wedge , \vee , \supset , \equiv), *кванторы общности* \forall (для всех) и *существования* \exists (существует)¹⁾. В языке предикатов содержится язык высказываний, так как высказывание — не что иное, как предикатная константа без аргументов или, точнее, предикатная константа с нулевым числом мест. Удобно заменить понятие индивидуальной константы более общим понятием *функциональной константы*. Функциональная константа с определенным числом мест — в точности то же, что предикатная константа. Индивидуальная константа — просто нульместная функциональная константа.

¹⁾ Соответственно *универсальный* и *экзистенциальный*. — *Прим. перев.*

Ввиду того что исчисление предикатов позволяет оперировать с отдельными объектами (переменными и константами), иногда полезно вводить предикат равенства между этими объектами, однако это не будет обязательным. Для обозначения этого предиката используется классический символ $=$. Переменные, предикатные и функциональные константы обозначаются буквами или словами, иногда снабженными верхними и/или нижними индексами. Можно полностью избежать риска перепутать эти три класса объектов, если ввести строгие типографские соглашения. Что касается предикатных и функциональных констант, то можно также потребовать явного указания числа мест (например, в виде показателя). Однако на практике соглашения менее строгие, а принадлежность объекта к определенному синтаксическому классу определяется по контексту или по неформальным комментариям к формуле. Это послабление позволяет давать объектам мнемонические имена. Они будут широко использоваться для представления знаний (гл. 3).

Следующая лексика является обычной и будет полезна для понимания примеров, приводимых в этом разделе:

x, y, z	переменные,
a, b, c	индивидуальные константы,
f, g, h	функциональные константы,
p, q, r	высказывания,
P, Q, R	предикатные константы.

1.2.3. Синтаксис исчисления предикатов

Словарь, введенный в предыдущем параграфе, позволяет определить *термы*, *формы* и *квантификации* (все они уже упоминались), а также *атомы* и *формулы*. Теперь изложим правила соответствующих построений.

- *Термом* является всякая переменная и всякая функциональная форма.
- *Функциональная форма* — это функциональная константа, соединенная с подходящим числом

термов. Если f — функциональная n -местная константа и t_1, \dots, t_n — термы, то соответствующая форма обычно обозначается через $f(t_1, \dots, t_n)$. Если $n = 0$, то пишут f вместо $f()$.

- *Предикатная форма* — это предикатная константа, соединенная с подходящим числом термов. Если P — предикатная m -местная константа и t_1, \dots, t_m — термы, то соответствующая форма обычно обозначается через $P(t_1, \dots, t_m)$. При $m = 0$ пишут P вместо $P()$.
- *Атом* — это предикатная форма или некоторое равенство, т. е. выражение типа $(s = t)$, где s и t — термы.
- Понятие *формулы* определяется рекурсивно (индуктивно) следующими правилами:
 - атом есть формула;
 - если A — формула, то $\neg A$ — формула;
 - если A и B — формулы, то $(A \wedge B)$, $(A \vee B)$, $(A \supset B)$ и $(A \equiv B)$ — формулы;
 - если A — формула и x — переменная, то $\forall x A$ и $\exists x A$ — формулы.

Замечания. Как и в исчислении высказываний, круглые скобки иногда заменяются фигурными или квадратными и даже опускаются совсем, если контекст это позволяет. Естественный язык не допускает такого употребления скобок, поэтому могут возникать двусмысленности, особенно когда число мест у предикатной или функциональной константы неизвестно.

Слова «функция» и «предикат» иногда используются вместо выражений «функциональная форма» и «предикатная форма», особенно в приложениях (гл. 3—6).

1.2.4. Свободные и связанные переменные, область действия

Переменные в логике играют роль, аналогичную их роли в алгебре или анализе. Прежде всего они

позволяют указать в структуре математического объекта те места, которые при использовании этого объекта будут заняты другими объектами. Например, линейная функция f может быть записана классической формулой

$$f(x) = x + 6$$

или еще с помощью обозначения, подчеркивающего статус объекта f , формулой типа

$$\dot{f} = \lambda x. x + 6.$$

В этом контексте x означает не объект, а место для объекта. Можно без хлопот *переобозначить* x , т. е. заменить два вхождения x , например, на y . Очевидно, что нельзя заменить только одно вхождение, а другое оставить без изменения. Такой тип замены называется *одновременной подстановкой*. Говорят, что x — *связанная* переменная, желая тем самым отразить то обстоятельство, что первое вхождение x вводит (описывает) связь, которой подчинено второе вхождение x . Переименование связанной переменной допустимо, так как сколько было различных вхождений переменных до одновременной подстановки, столько же останется и после. Например, выражение

$$\lambda x y. x * y$$

равняется

$$\lambda u y. u * y,$$

но не

$$\lambda u u. u * u.$$

Теперь рассмотрим такой контекст, где x тем или иным образом представляет число 3, а sq — квадратичная функция. Можно написать

$$sq(x) = x + 6,$$

но, очевидно, уже нельзя заменять x на y , так как на этот раз через x обозначен вполне определенный объект. Переменная x называется *свободной*, точнее, два вхождения x называются свободными.

Рассмотрим следующий пример:

$$A_k = \sum_{i=1}^n B_k^i v_i.$$

Единственной связанной переменной здесь является i . Связь вводится первым вхождением i . Остальные символы представляют свободные переменные или функциональные константы. Можно переименовать i в j или l , но не в k или n .

Наконец, в выражении

$$y = \lambda x. \left(y_0 + \int_{x_0}^x f(t, y(t)) dt \right)$$

связанные переменные — x и t . Связи вводятся соответственно первым вхождением x и последним вхождением t .

Теперь вернемся к логике. Из интуитивного понимания квантификации вытекает, что имеется связь между вхождениями переменной, содержащейся в квантификации. В формуле

$$\forall x [H(x) \supset M(x)]$$

переменная x «связана». Связь вводится первым вхождением, которое следует непосредственно после квантора общности.

Область действия некоторой квантификации есть формула, к которой применяется эта квантификация. Вхождение переменной x , появляющейся в квантификациях $\forall x$ или $\exists x$, называется *квантифицированным*. Каждое вхождение переменной x в область действия этой квантификации является связанным. Вхождение переменной x *свободное*, если оно не является ни квантифицированным, ни связанным.

Замечания. Часто употребляют выражение «область действия квантора» вместо «область действия квантификации».

Переменная, имеющая связанное вхождение в некоторую формулу, имеет также квантифицированное вхождение в эту формулу.

В некоторой данной формуле переменная является

квантифицированной (связанной, свободной), если она имеет хотя бы одно квантифицированное (связанное, свободное) вхождение.

В формулах $\forall x A$ и $\exists x A$ область действия x есть A . Если x и y — переменные, то области действия x и y либо не пересекаются, либо одна вложена в другую.

Рассмотрим следующие примеры:

$$\forall x [P(x, a) \supset \exists x Q(x)],$$

$$\forall x P(x, a) \supset \exists x Q(x),$$

$$\forall x P(x, a) \supset Q(x).$$

В первом случае две квантификации по одной и той же переменной перекрываются. Мы не желаем сопоставлять семантику этому типу формул, его мы исключим из рассмотрения, подчиняясь следующему правилу введения кванторов:

$\forall x A$ и $\exists x A$ — формулы только в том случае, если x — переменная и A — формула, не содержащая связанных вхождений x .

Во втором случае две связанные переменные случайно получили одно и то же имя. Это совсем незначительное затруднение.

В третьем случае и связанная, и свободная переменные получили одно и то же имя. Хотя это допустимо, лучше переименовать связанную переменную и написать, например,

$$\forall y P(y, a) \supset Q(x).$$

1.2.5. Семантика исчисления предикатов

Формулы исчисления предикатов, как и формулы исчисления высказываний, могут быть интерпретированы, т. е. могут получить значение истинности. Однако формулы исчисления предикатов состоят не только из подформул, но также и из термов. Следовательно, необходимо будет интерпретировать также термы. Терм интуитивно означает объект. Таким образом, интерпретация должна специфицировать множество объектов, называемое областью интерпретации.

Точнее, интерпретация I — это тройка (D, I_c, I_v) со следующими свойствами:

- D — непустое множество, область интерпретации.
- I_c — функция, которая сопоставляет каждой функциональной n -местной константе f некоторую функцию $I_c(f)$ из D^n в D и которая сопоставляет каждой предикатной m -местной константе P некоторую функцию $I_c(P)$ из D^m в $\{\mathbf{И}, \mathbf{Л}\}$ ¹⁾.
- I_v — функция, сопоставляющая каждой переменной некоторый элемент из D .

Теперь можно задать для интерпретации $I = (D, I_c, I_v)$ такие правила, которые каждой формуле A ставят в соответствие некоторое значение истинности $I(A)$, а каждому терму t сопоставляют элемент $I(t)$ из D .

- Если x — свободная переменная, то $I(x) =_{\text{def}} I_v(x)$.
- Если f — функциональная n -местная константа, t_1, \dots, t_n — термы, то $I(f(t_1, \dots, t_n)) =_{\text{def}} = (I_c(f))(I(t_1), \dots, I(t_n))$.
- Если P — предикатная m -местная константа, t_1, \dots, t_m — термы, то $I(P(t_1, \dots, t_m)) =_{\text{def}} = (I_c(P))(I(t_1), \dots, I(t_m))$.
- Если s и t — термы, то $I(s = t)$ есть $\mathbf{И}$ при $I(s) = I(t)$, а иначе $\mathbf{Л}$.
- Если A и B — формулы, то $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \supset B)$ и $(A \equiv B)$ интерпретируются как в исчислении высказываний.

Осталось задать интерпретацию для двух типов квантифицированных формул. Введем сначала однопонятие. Если I — интерпретация с областью D_I , x — переменная, d — элемент из D_I , то $I_{x/d}$ означает такую интерпретацию J , что $D_J = D_I$, $J_c = I_c$, $J_v(x) = d$ и $J_v(y) = I_v(y)$ для всех свободных переменных y , отличных от x .

Правила интерпретации будут такими:

- Если A — формула и x — переменная, то $I(\forall x A)$ есть $\mathbf{И}$ при условии, что $I_{x/d}(A)$ есть

¹⁾ По соглашению функция из D^0 в B есть просто элемент из B .

И для всех элементов d из D .

- Если A — формула и x — переменная, то $I(\exists x A)$ есть И при условии, что $I_{x/d}(A)$ есть И хотя бы для одного элемента d из D .

Формула A исчисления предикатов называется *истинной при интерпретации I* , если $I(A) = \text{И}$.

Теперь видно, что запрещение перекрытия кванторов, действующих на одну и ту же переменную, не является существенным ограничением. В частности, $\forall x \exists x A$ интерпретируется как $\exists x A$ и $\exists x \forall x A$ интерпретируется как $\forall x A$. Ясно также, почему требуется условие $D \neq \emptyset$: без него естественные импликации

$$\forall x A \supset A,$$

$$A \supset \exists x A$$

не всегда были бы истинны.

Заметим также, что наши правила интерпретации соответствуют интуитивным представлениям. В частности, формальное значение кванторов хорошо моделирует их естественное значение. Эти правила подтверждают также сказанное о переименовании переменных в § 1.2.4.

Как и формулы исчисления высказываний, формулы исчисления предикатов делятся на три класса: *общезначимые* формулы, которые истинны при всех интерпретациях, *невыполнимые*, которые ложны при всех интерпретациях, и *нейтральные* (или просто *выполнимые*), которые истинны только при некоторых интерпретациях. В противоположность тому, что имело место для исчисления высказываний, эти три класса не являются рекурсивными: не существует детерминированного алгоритма, который определял бы, к какому классу принадлежит произвольная формула исчисления предикатов. Позже мы вернемся к этому вопросу (§ 2.2.11). Вполне естественно, что формула исчисления предикатов *выполнима*, если она истинна хотя бы при одной интерпретации.

Приведем примеры нескольких формул, общезначимость которых легко установить. Они описывают

взаимоотношения между \forall -квантификацией¹⁾ и позиционными связками:

$$\begin{aligned}(\forall x A \wedge \forall x B) &\equiv \forall x (A \wedge B), \\(\forall x A \vee \forall x B) &\supset \forall x (A \vee B), \\ \forall x (A \supset B) &\supset (\forall x A \supset \forall x B), \\ \forall x (A \equiv B) &\supset (\forall x A \equiv \forall x B).\end{aligned}\tag{1.10}$$

Здесь речь идет по существу о *схемах* формул. Из этих схем можно получить общезначимые формулы, если заменить A и B на формулы, не содержащие квантификации по x .

Замечание. В трех последних схемах в качестве главной связки стоит импликация. Если ее заменить эквивалентностью, то формулы перестанут быть общезначимыми.

Приведем аналогичные схемы, касающиеся \exists -квантификации:

$$\begin{aligned}\exists x (A \vee B) &\equiv (\exists x A \vee \exists x B), \\ \exists x (A \wedge B) &\supset (\exists x A \wedge \exists x B), \\ \exists x (A \supset B) &\equiv (\forall x A \supset \exists x B).\end{aligned}\tag{1.11}$$

Следующая схема устанавливает связь между \forall -квантификацией и \exists -квантификацией:

$$\forall x \neg A \equiv \neg \exists x A.$$

Можно распространить *принцип двойственности* (§ 1.1.9) на исчисление предикатов. Если в некоторой логической эквивалентности, не содержащей связки \supset , поменять местами **И** и **Л**, \wedge и \vee и, наконец, \forall и \exists , то опять получим логическую эквивалентность. К исчислению предикатов возможен алгебраический подход, и он оказывается продуктивным. Это описано в [5].

¹⁾ Иначе, *универсальной квантификацией*. — Прим. перев.

Отметим также важные соотношения между различными квантификациями:

$$\forall x \forall y A \equiv \forall y \forall x A,$$

$$\exists x \exists y A \equiv \exists y \exists x A,$$

$$\exists x \forall y A \supset \forall y \exists x A.$$

Импликацию в последней схеме заменить на эквивалентность, конечно, нельзя. Например, в аддитивной группе целых чисел существование у каждого элемента противоположного ему выражается формулой

$$\forall y \exists x (x + y = 0).$$

Напротив, формула

$$\exists x \forall y (x + y = 0),$$

очевидно, некорректна: не существует такого элемента x , который был бы противоположным для всех целых чисел. Интерпретация, даваемая этим примером, эффективно опровергает схему $\exists x \forall y A \equiv \forall y \exists x A$.

Наконец, следующее правило показывает, что исчисление предикатов действительно является расширением исчисления высказываний.

Если $S(A_1, \dots, A_n)$ — общезначимая схема исчисления высказываний, то она является также общезначимой схемой исчисления предикатов.

Из общезначимой схемы исчисления высказываний $(\neg \neg A \equiv A)$ можно вывести не только $[\neg \neg (p \wedge q) \equiv (p \wedge q)]$, но и $[\neg \neg \forall x P(x) \equiv \forall x P(x)]$.

1.2.6. Подстановка и конкретизация

Подстановка есть отображение множества переменных V в множество термов T . Подстановка σ *конечна*, если $\sigma(x)$ отлично от x лишь для конечного числа элементов из V . В дальнейшем будут рассматриваться только конечные подстановки и поэтому соответствующее прилагательное употребляться не будет. Подстановка σ будет задаваться множеством таких пар $(x, \sigma(x))$, что $\sigma(x) \neq x$.

Пусть t — терм и σ — подстановка. Терм $\sigma[t]$ получается одновременной заменой всех вхождений переменной x в t на их образы относительно $\sigma(x)$. Вот пример:

$$\sigma = \{(x, f(x)), (y, g(x, z))\},$$

$$t = g(f(x), g(f(z), y)),$$

$$\sigma[t] = g(f(f(x)), g(f(z), g(x, z))).$$

Эта манипуляция сопоставляет каждой подстановке σ некоторую функцию из T в T , которая отображает t в $\sigma[t]$. Так как для всех переменных x имеем $\sigma[x] = \sigma(x)$, то эта новая функция является продолжением σ . Обозначим ее тоже через σ .

Композиция двух подстановок σ_1 и σ_2 есть функция $\sigma_2 \circ \sigma_1$, определяемая следующим образом:

$$(\sigma_2 \circ \sigma_1)[t] = \sigma_2[\sigma_1[t]].$$

Легко проверить, что это правило композиции подстановок имеет смысл для всех подстановок и дает подстановку. Оно ассоциативно и допускает в качестве нейтрального элемента тождественную функцию. Это означает, что множество подстановок с операцией композиции является моноидом (некоммутативным).

Если подстановки σ_1 и σ_2 таковы, что для любой переменной x хотя бы один из термов $\sigma_1(x)$, $\sigma_2(x)$ равен x , то объединение образов σ_1 и σ_2 определяет еще одну подстановку, которая называется *объединением* подстановок σ_1 и σ_2 и обозначается через $\sigma_1 \cup \sigma_2$.

Терм t_2 называется *конкретизацией* (для) терма t_1 , если существует подстановка σ , такая, что $t_2 = \sigma[t_1]$. Множество переменных терма t обозначим через $var(t)$. Терм t *вполне конкретизирован*, если $var(t) = \emptyset$.

Отношение «является конкретизацией (для)» обозначим символом $<$. Очевидно, оно рефлексивно и транзитивно. Оно не антисимметрично, но верен следующий результат: соотношения $t_1 < t_2$ и $t_2 < t_1$ имеют место тогда и только тогда, когда t_1 и t_2 отличаются только именами переменных. Точнее,

существует биективное соответствие f между $\text{var}(t_1)$ и $\text{var}(t_2)$, такое, что t_2 получается одновременной заменой всех вхождений переменной x на $f(x)$. В этом случае пишут $t_1 \simeq t_2$.

Отношение \simeq является отношением эквивалентности. На фактор-множестве T/\simeq множества термов по этому отношению эквивалентности отношение $<$ становится отношением частичного порядка. Существует максимальный элемент, а именно класс термов, составленных из единственной переменной. Минимальными элементами являются вполне конкретизированные термы. Наименьшая верхняя граница конечного множества термов всегда существует и ее легко найти. Наоборот, наибольшая нижняя граница существует не всегда.

Наибольшая нижняя граница для пары

$$\{h(x, f(a), g(y)), h(y, z, g(b))\}$$

есть терм

$$h(y, f(a), g(b)).$$

Понятие конкретизации непосредственно обобщается на предикатные формы, называемые *матрицами*, которые являются формулами, не содержащими квантификаций. Понятие конкретизации еще сохраняет смысл и для формул общего вида, содержащих квантификации, но при этом можно конкретизировать только свободные переменные. (Наличие одинаковых имен у свободной и связанной переменных здесь также оказалось бы источником сложностей.) Например, рассмотрим формулу

$$\forall x [P(x, y, z) \vee Q(a, x)] \supset \exists u R(b, f(u), v).$$

Свободные переменные — y, z и v . По обычному соглашению a, b и f — функциональные константы (соответственно с 0, 0 и 1 местами). Конкретизации этой формулы получаются применением подстановок типа $\{(y, t_1), (z, t_2), (v, t_3)\}$ при соблюдении следующего ограничения: t_1, t_2 и t_3 не должны содержать вхождений переменных x и u , связанных в исходной формуле.

Универсальным замыканием (или \forall -замыканием) формулы A , содержащей свободные переменные x_1, \dots, x_n , называется формула $\forall x_1 \dots \forall x_n A$. Точно так же экзистенциальным замыканием (или \exists -замыканием) формулы называется формула $\exists x_1 \dots \exists x_n A$. Ясно, что порядок следования переменных x_i роли не играет.

Рассмотрим формулу A , ее \forall -замыкание U_A и \exists -замыкание E_A , а также конкретизацию A' для A . Имеем следующие результаты:

$$\models A \Leftrightarrow \models U_A,$$

$$\models \neg A \Leftrightarrow \models \neg E_A,$$

$$\models U_A \Rightarrow \models A',$$

$$\models A' \Rightarrow \models E_A,$$

где обозначение $\models A$ означает общезначимость формулы A (в рамках исчисления предикатов).

Можно еще добавить, что формула общезначима тогда и только тогда, когда все ее конкретизации общезначимы, и что формула выполнима тогда и только тогда, когда хотя бы одна из ее конкретизаций выполнима. В этом предложении слово «конкретизация» можно заменить на «вполне конкретизация».

1.2.7. Предваренная и нормальные формы

В логике высказываний были введены две нормальные формы: конъюнктивная и дизъюнктивная. Мы видели, что приведение формулы логики высказываний к одной из этих нормальных форм позволяет упростить алгоритмы доказательства выполнимости общезначимости (§§ 1.1.11—1.1.14). По аналогичным причинам стоит ввести нормальные формы и в исчислении предикатов.

Проблемы, возникающие при оперировании с квантификациями и областями действия кванторов, иногда бывают достаточно сложными. В частности, к некоторым трудностям может привести одновременное присутствие в формуле свободных и связанных вхождений одной и той же переменной. Положение заведомо проще для случая предваренных форм.

Предваренной формой называется формула, состоящая из матрицы, перед которой стоит *префикс*, т. е. некоторая конечная последовательность квантификаций. Эти квантификации относятся к различным переменным, и их порядок вообще является существенным¹⁾. Таким образом, формула имеет вид

$$Q_1x_1 \dots Q_nx_nM,$$

где символ Q_i означает либо \forall , либо, \exists , для $i = 1, \dots, n$ и где M — формула (матрица), не содержащая квантификаций.

Без ограничения общности можно потребовать, чтобы квантифицированы могли быть только переменные, имеющие вхождение (обязательно свободное) в матрицу. Действительно, по правилам интерпретации если формула A не содержит вхождений переменной x , то формулы A , $\exists xA$ и $\forall xA$ имеют одно и то же значение истинности при всех интерпретациях.

Интерес к предваренным формам связан со следующей теоремой.

Теорема 1.9. *Для любой логической формулы существует логически эквивалентная ей предваренная форма.*

Доказательство. Алгоритм получения предваренной формы очень прост. Этапы таковы:

- Исключить связки эквивалентности и импликации по правилам преобразования из § 1.1.10.
- Переименовать (если необходимо) связанные переменные таким образом, чтобы никакая переменная не имела бы одновременно свободных и связанных вхождений. Это условие требуется не только для рассматриваемой формулы, но также для всех ее подформул.
- Удалить те квантификации, область действия которых не содержит вхождений квантифици-

¹⁾ В случае повторения квантификаций для одной и той же переменной можно принять, что существенна только самая правая квантификация: это согласуется с общим правилом интерпретации квантифицированных формул.

рованной переменной. Такие квантификации в действительности не нужны.

- Преобразовать все вхождения отрицания в стоящие непосредственно перед атомами в соответствии со следующими правилами:

$$\neg \forall x A \rightarrow \exists x \neg A,$$

$$\neg \exists x A \rightarrow \forall x \neg A,$$

$$\neg (A \wedge B) \rightarrow (\neg A \vee \neg B),$$

$$\neg (A \vee B) \rightarrow (\neg A \wedge \neg B),$$

$$\neg \neg A \rightarrow A.$$

- Переместить все квантификации в начало формулы. Для этого используется соответствующий набор правил преобразования. Здесь мы приведем эти правила для конъюнкции. Парные им правила для дизъюнкции получаются применением двойственности.

$$(\forall x A \wedge \forall x B) \rightarrow \forall x (A \wedge B),$$

$$(\forall x A \wedge B) \rightarrow \forall x (A \wedge B), \quad \text{если } B \text{ не содержит } x,$$

$$(A \wedge \forall x B) \rightarrow \forall x (A \wedge B), \quad \text{если } A \text{ не содержит } x,$$

$$(\exists x A \wedge B) \rightarrow \exists x (A \wedge B), \quad \text{если } B \text{ не содержит } x,$$

$$(A \wedge \exists x B) \rightarrow \exists x (A \wedge B), \quad \text{если } A \text{ не содержит } x.$$

Для полноты этого набора правил необходимо добавить возможность переименования некоторых связанных переменных. Например, формула $\exists x P(x) \wedge \forall x Q(x)$ будет сначала преобразована в $\exists x P(x) \wedge \forall y Q(y)$ (перед применением правил преобразования). Заметим также, что для упрощения формул в любой момент можно применять свойства коммутативности, ассоциативности и идемпотентности связок \wedge и \vee (§ 1.1.9). Итак, теорема конструктивно доказана. \square

В качестве приложения найдем предваренную форму, эквивалентную формуле

$$\forall x [P(x) \wedge \forall y \exists x (\neg Q(x, y) \supset \forall z R(a, x, y))].$$

Этапы этого поиска последовательно перечислены ниже.

Исключение связки импликации:

$$\forall x [P(x) \wedge \forall y \exists x (\neg \neg Q(x, y) \vee \forall z R(a, x, y))].$$

Переименование:

$$\forall x [P(x) \wedge \forall y \exists u (\neg \neg Q(u, y) \vee \forall z R(a, u, y))].$$

Удаление бесполезной квантификации:

$$\forall x [P(x) \wedge \forall y \exists u (\neg \neg Q(u, y) \vee R(a, u, y))].$$

Применение правил, касающихся отрицания:

$$\forall x [P(x) \wedge \forall y \exists u (Q(u, y) \vee R(a, u, y))].$$

Перемещение квантификаций:

$$\forall x \forall y [P(x) \wedge \exists u (Q(u, y) \vee R(a, u, y))],$$

$$\forall x \forall y \exists u [P(x) \wedge (Q(u, y) \vee R(a, u, y))].$$

Замечание. Одна формула может допускать много эквивалентных предваренных форм. Вид полученного результата зависит от порядка применения правил, а также от произвола при переименовании. Например, эквивалентны следующие предваренные формы:

$$\forall x (P(x) \wedge Q(x)) \text{ и } \forall x \forall y (P(x) \wedge Q(y)).$$

Понятия литеры, дизъюнкта и КНФ, введенные в исчислении высказываний, непосредственно распространяются на исчисление предикатов. *Литера* — это атом или его отрицание, *дизъюнкт* — дизъюнкция литер, *конъюнктивная нормальная форма* — предваренная форма, матрица которой есть конъюнкция дизъюнктов. Равным образом можно использовать двойственные понятия: *импликант (конъюнкт)* есть конъюнкция литер, *дизъюнктивная нормальная форма* — это предваренная форма, матрица которой есть дизъюнкция импликантов.

1.2.8. Сколемовские и клаузуальные формы

Механизм квантификации, очевидно, не только основа могущества, но и источник сложности исчисления предикатов. Предваренные и нормальные формы осо-

бенно интересны тем, что, с одной стороны, они требуют лишь «дисциплинированного» использования квантификации, и с другой стороны, они сохраняют всю выразительную силу исчисления предикатов. Действительно, любая формула допускает эквивалентную нормальную форму (хотя иногда более громоздкую или менее обозримую, чем исходная формула).

Можно установить еще более строгие пределы использования механизма квантификации ценой приемлемого уменьшения выразительной мощи. Точнее, будет предложено средство сопоставления каждой формуле A некоторой формулы S_A очень простого строения с гарантией, что формулы A и S_A либо обе выполнимы, либо обе невыполнимы. Эта связь между A и S_A строго слабее, чем логическая эквивалентность, но все-таки она очень полезна по следующей причине. Предположим, что мы хотим доказать, что заключение C является логическим следствием гипотез H_1 и H_2 . Это сводится к доказательству невыполнимости формулы G : $(H_1 \wedge H_2 \wedge \neg C)$, или же невыполнимости соответствующей формулы S_G (вообще это будет легче).

Без ограничения общности можно рассматривать только предваренные формы, потому что любая формула допускает эквивалентную предваренную форму. Кроме того, достаточно оперировать с замкнутыми формулами, т. е. без свободных переменных. Действительно, если A — формула со свободными переменными x_1, \dots, x_n , которая не содержит ни одного связанного вхождения переменных x_1, \dots, x_n (после осуществления переименования), то формула $\exists x_1 \dots \exists x_n A$, с одной стороны, замкнута и, с другой стороны, выполнима тогда и только тогда, когда выполнима формула A .

Форма S_A , которую мы сейчас опишем, известна под названием *сколемовской формы* (соответствующей формуле A). Сведение A к S_A представляет особый интерес, потому что доказательство невыполнимости становится эффективнее, если ограничиться только формулами, представленными в сколемовской форме. Приведение произвольной формулы исчисления

предикатов к сколемовской форме требует двух предварительных операций:

- Привести данную формулу к предваренной форме (§ 1.2.7), состоящей из префикса и матрицы.
- Привести затем матрицу к КНФ с помощью алгоритма § 1.1.10.

Результатом будет замкнутая предваренная форма. Сколемовская форма, соответствующая ей (и, следовательно, исходной формуле), получится применением следующей процедуры, называемой сколемовским преобразованием и служащей для исключения \exists -квантификаций.

- Сопоставить каждой \exists -квантифицированной переменной список \forall -квантифицированных переменных, предшествующих ей, а также некоторую еще не использованную функциональную константу, число мест у которой равно мощности списка.
 - В матрице нашей формулы заменить каждое вхождение каждой \exists -квантифицированной переменной на некоторый терм. Этот терм является функциональной константой, соответствующей данной переменной и снабженной списком аргументов, также соответствующим той же самой переменной.
 - Устранить из формулы все \exists -квантификации.
- Обратимся вновь к примеру из предыдущего параграфа. Преобразуем формулу

$$\forall x [P(x) \wedge \forall y \exists x (\neg Q(x, y) \supset \forall z R(a, x, y))].$$

Осуществленные в предыдущем параграфе преобразования привели нас к следующей КНФ:

$$\forall x \forall y \exists u [P(x) \wedge (Q(u, y) \vee R(a, u, y))].$$

Описанная выше процедура позволяет получить сколемовскую форму, соответствующую исходной формуле:

$$\forall x \forall y [P(x) \wedge (Q(f(x, y), y) \vee R(a, f(x, y), y))].$$

Рассмотрим более пристально связь между формулой и ее формой Сколема. Например, обратимся к

замкнутой предваренной форме

$$A: \exists u \forall v \exists w \forall x \forall y \exists z M(u, v, w, x, y, z).$$

Ей соответствует сколемовская форма

$$S_A: \forall v \forall x \forall y M(a, v, f(v), x, y, g(v, x, y)).$$

Предполагается, что функциональные константы a , f и g не входят в матрицу $M(u, v, w, x, y, z)$.

Констатируем сразу же, что формула $(S_A \supset A)$ общезначима. Отсюда следует, что если формула A невыполнима, то формула S_A тоже невыполнима. Наоборот, предположим, что формула A выполнима и принимает истинное значение при некоторой интерпретации I в области D . Положим $M = I_c(M)$. Для любых V, X, Y существуют элементы $A, B, C \in D$, такие, что $M(A, V, B, X, Y, C)$ истинно в D . Обратив внимание на относительное положение кванторов в формуле A , видим, что A не зависит от V, X и Y , что B может зависеть от V (но не от X и Y), что, наконец, C может зависеть от V, X и Y . Для получения интерпретации, при которой истинна формула S_A , достаточно интерпретировать f и g как подходящие функции выбора¹⁾ F и G , т. е. такие, что $F(V)$ и $G(V, X, Y)$ были бы допустимыми значениями B и C .

Отметим, что сколемовская форма есть предваренная форма, префикс которой содержит только \forall -квантификации. С другой стороны, каждая предваренная форма такого специального типа является ее собственной сколемовской формой. Поэтому удобно соединить эти два понятия. Когда позволяет контекст и особенно когда типографические условности дают возможность отличать индивидуальные константы от переменных, префикс сколемовской формы можно опустить: подразумевается, что каждая присутствующая в матрице переменная \forall -квантифицирована. В этом случае, пожалуй, лучше говорить о конкретизации для сколемовской формы, а не о конкретизации для матрицы сколемовской формы.

¹⁾ Используется также термин «сколемовские функции». — Прим. ред.

Клаузуальной формой называется такая сколемовская форма, матрица которой является КНФ. Любая сколемовская форма допускает эквивалентную клаузуальную форму.

1.2.9. Эрбранова интерпретация и компактность

Далее мы увидим, что не существует алгоритма, позволяющего распознать общезначимость, нейтральность или невыполнимость произвольной формулы исчисления предикатов. Это связано с существованием бесконечного числа возможных интерпретаций для формул исчисления предикатов. Может ведь потребоваться обязательное рассмотрение всех интерпретаций. Частные, но интересные результаты в этом направлении были получены Эрбраном. Они приводят к упрощенной проверке выполнимости формул. Так как каждой формуле F можно сопоставить такую клаузуальную форму S_F , что формулы F и S_F одновременно выполнимы (невыполнимы), то будем рассматривать только клаузуальные формы. Представленные здесь результаты относятся к исчислению предикатов без равенства. Более подробное изложение см. в [14] и [65].

Основная идея, подводящая к определению эрбрановой области, состоит в следующем. Клаузуальная форма невыполнима тогда и только тогда, когда она принимает значение **Л** при всех интерпретациях. Увы, рассмотрение всех областей возможной интерпретации исключается. Поэтому было бы неплохо попытаться выявить такую специальную область, которая была бы тесно связана с рассматриваемой клаузуальной формой и чтобы форма была невыполнима тогда и только тогда, когда она принимает значение **Л** при всех интерпретациях *на этой области*. Такая область действительно существует и называется эрбрановой областью. Она определяется так.

Эрбранова область клаузуальной формы G — это минимальное множество H_G , удовлетворяющее следующим условиям:

- Для любой индивидуальной константы a , имеющей вхождение в G , область H_G содержит соответ-

ствующую эрбранову константу, обозначаемую A .

- Если G не содержит индивидных констант, то тем не менее H_G содержит эрбранову константу, обозначаемую C .
- Для каждой n -местной функциональной константы f , имеющей вхождение в G , вводится функциональная константа F . Принадлежность элементов h_1, \dots, h_n к H_G влечет за собой принадлежность к H_G терма $F(h_1, \dots, h_n)$.

Эрбранова область считается «универсальной» (точный смысл этого термина описан в теореме 1.10). Таким образом, ее элементы не имеют конкретного значения: они лишь простые синтаксические объекты.

При отсутствии функциональных констант эрбранова область клаузуальной формы сводится к множеству из одного элемента $\{C\}$. Например, такой случай имеет место для клаузуальной формы $(P(x) \vee Q(x)) \wedge \neg Q(x)$. Напротив, при наличии хотя бы одной функциональной (не индивидной) константы эрбранова область всегда бесконечна. Для клаузуальной формы

$$P(f(x))$$

такой областью будет множество

$$\{C, F(C), F(F(C)), \dots\}.$$

Эрбрановой интерпретацией для формулы G называется интерпретация I для G , удовлетворяющая следующим условиям:

- Область интерпретации является эрбрановой областью H_G .
- Функция интерпретации констант I_c сопоставляет каждой индивидной константе a соответствующую эрбранову константу A . Эта функция сопоставляет каждой n -местной функциональной константе f функцию F :

$$H_G^n \rightarrow H_G : (h_1, \dots, h_n) \rightarrow F(h_1, \dots, h_n).$$

Существует много эрбрановых интерпретаций, так как интерпретации предикатных констант, как и

интерпретации переменных, могут быть выбраны произвольно.

Замечание. Для удобства эрбрановы символы часто отождествляют с соответствующими функциональными константами. Например, пишут $f(a)$ вместо $F(A)$.

Теорема 1.10. *Клаузальная форма G невыполнима тогда и только тогда, когда она ложна при всех эрбрановых интерпретациях.*

Доказательство. Необходимость этого утверждения очевидна. Для установления достаточности можем ограничиться доказательством, что каждой интерпретации I с $I(G) = \mathbf{И}$ можно сопоставить эрбранову интерпретацию I^h , такую, что $I^h(G) = \mathbf{И}$.

В действительности достаточно определить $I^h(P(t_1, \dots, t_n))$ для любой n -местной предикатной константы P и произвольных термов t_1, \dots, t_n . Каждому элементу из H_G естественным образом ставится в соответствие элемент из области D интерпретации I . Если элементу $I^h(t_i)$, здесь $h_i \in H_G$, сопоставлен элемент $d_i \in D$, то $I^h(P(t_1, \dots, t_n))$ есть $\mathbf{И}$ тогда и только тогда, когда $(I(P)(d_1, \dots, d_n))$ есть $\mathbf{И}$. \square

Эрбранова интерпретация задается значениями истинности, которые она приписывает предикатным формам $P(t_1, \dots, t_n)$, термы которых принадлежат эрбрановой области. Эти формы называются *фундаментальными формами*. Так как число предикатных констант конечно, а эрбранова область не более чем счетна, то множество фундаментальных форм, связанных с клаузальной формой, тоже не более чем счетно. Если вместо клаузальной формы взять бесконечное множество дизъюнктов, результат останется верным. Мощность множества фундаментальных форм не больше мощности множества формул.

Эрбрановы интерпретации похожи на интерпретации исчисления высказываний: каждая фундаментальная форма соответствует некоторому высказыванию. Примеры будут приведены в следующем параграфе.

Основываясь на теореме 1.10, можно распространить теорему о компактности из исчисления высказываний (§ 1.1.18) на исчисление предикатов.

Теорема 1.11. *Произвольная совокупность формул исчисления предикатов выполнима тогда и только тогда, когда все ее конечные подмножества выполнимы.*

Доказательство. Необходимость этого утверждения очевидна. Дадим только доказательство достаточности. Пусть S — некоторое множество формул исчисления предикатов. Без ограничения общности можно предположить, что S состоит только из дизъюнктов. Рассмотрим множество S' *фундаментальных конкретизаций*, то есть полных конкретизаций дизъюнктов в эрбрановой области.

По теореме 1.10 и следующим за ней замечаниям S выполнимо тогда и только тогда, когда S' выполнимо. Между тем множество S' состоит из дизъюнктов исчисления высказываний, потому что каждую фундаментальную форму можно уподобить подходящему высказыванию. В силу теоремы компактности из исчисления высказываний (§ 1.1.18) S' выполнимо тогда и только тогда, когда все конечные подмножества S' выполнимы. Таким образом, остается доказать, что каждое конечное подмножество множества S' выполнимо. Пусть R' именно такое множество и R — подмножество из S , образованное дизъюнктами, имеющими фундаментальную конкретизацию в R' . Множество R конечно, следовательно, выполнимо. Множество фундаментальных конкретизаций для R тоже выполнимо (по теореме 1.10) и содержит подмножество R' . Значит, и R' выполнимо. \square

Теорема 1.12. *Любое конечное или счетное выполнимое множество формул исчисления предикатов допускает конечную или счетную модель.*

Доказательство. Достаточно рассмотреть эрбранову модель. \square

Этот важный результат известен под названием *теоремы Лёвенгейма — Сколема*. Отметим, что существуют более сильные версии [3], [5]. Эта теорема

будет применена при изучении теорий первого порядка (§ 2.2.5).

1.2.10. Два простых примера

Рассмотрим следующую схему обычного рассуждения:

$$H_1 : \forall x [P(x) \supset Q(x)],$$

$$H_2 : \forall x [Q(x) \supset R(x)],$$

$$C : \forall x [P(x) \supset R(x)].$$

Попытаемся доказать, что заключение C действительно является логическим следствием гипотез H_1 и H_2 или что формула $(H_1 \wedge H_2 \wedge \neg C)$ невыполнима. Она эквивалентна следующей

$$\forall x ([P(x) \supset Q(x)] \wedge [Q(x) \supset R(x)]) \wedge \\ \wedge \exists y [P(y) \wedge \neg R(y)].$$

Эквивалентной предваренной формой является формула

$$\exists y \forall x ([P(x) \supset Q(x)] \wedge [Q(x) \supset R(x)] \wedge \\ \wedge [P(y) \wedge \neg R(y)]).$$

Эквивалентная КНФ есть

$$\exists y \forall x ([\neg P(x) \vee Q(x)] \wedge [\neg Q(x) \vee R(x)] \wedge \\ \wedge P(y) \wedge \neg R(y)).$$

Наконец, эквивалентная клаузная форма такова:

$$\forall x ([\neg P(x) \vee Q(x)] \wedge [\neg Q(x) \vee R(x)] \wedge \\ \wedge P(c) \wedge \neg R(c)).$$

Ее эрбрановой областью является множество из одного элемента $\{c\}$. Любая интерпретация в этой области сопоставляет x значение c , и единственной фундаментальной конкретизацией будет

$$[\neg P(c) \vee Q(c)] \wedge [\neg Q(c) \vee R(c)] \wedge P(c) \wedge \neg R(c).$$

Имеются три фундаментальные формы — это $P(c)$, $Q(c)$ и $R(c)$. Таким образом, существует $2^3 = 8$ эрбрановых интерпретаций. Ясно, что каждая из них

приписывает значение \mathbf{J} фундаментальной конкретизации, которая, следовательно, невыполнима. Тем самым установлено, что C действительно логическое следствие из H_1 и H_2 .

Теперь рассмотрим привычную рекуррентную схему в следующей простейшей форме:

$$B : P(a),$$

$$I : \forall x [P(x) \supset P(f(x))],$$

$$G : \forall x P(x).$$

Эти три формулы представляют соответственно базис индукции, индукционный шаг и общее утверждение. Используемая в этой схеме частичная интерпретация имеет своей областью множество N натуральных чисел, a интерпретируется как число 0 и f — как функция, которая каждому натуральному числу r сопоставляет следующее за ним число $r + 1$. В этом контексте, какой бы ни была данная интерпретация предиката P , схема общезначима в том смысле, что G — логическое следствие из B и I . (Этот результат перестает быть верным в более общем контексте.) Для доказательства можно установить выполнимость формулы $(B \wedge I \wedge \neg G)$. Она эквивалентна следующей:

$$P(a) \wedge \forall x [P(x) \supset P(f(x))] \wedge \exists y \neg P(y).$$

Матрица соответствующей клаузуальной формы будет, например, такова:

$$P(a) \wedge [\neg P(x) \vee P(f(x))] \wedge \neg P(b).$$

Эрбранова область, соответствующая этой форме, есть бесконечное множество

$$H = \{a, b, f(a), f(b), f(f(a)), \dots, f^{(n)}(a), f^{(n)}(b), \dots\},$$

где $f^{(n)}$ означает композицию n -го порядка (n -ую «степень» функции f). Таким образом, имеется бесконечное число эрбрановых интерпретаций. Каждая из них сопоставляет некоторое значение истинности каждому элементу бесконечного множества:

$$E = \{P(a), P(b), P(f(a)), P(f(b)), P(f(f(a))), \dots\}.$$

Существование некоторой истинной интерпретации достаточно для доказательства выполнимости $(B \wedge \wedge I \wedge \neg G)$.

Рассмотрим интерпретацию, которая сопоставляет значение **И** литере $P(f^{(n)}(a))$ и значение **Л** литере $P(f^{(n)}(b))$ для всех n . Ясно, что эта интерпретация является моделью множества фундаментальных конкретизаций:

$$\{P(f^{(m)}(a)) \wedge [\neg P(f^{(n)}(a)) \vee \vee P(f^{(n+1)}(a))] \wedge \neg P(f^{(n)}(b)) : m, n \in N\}.$$

Это позволяет получить объявленный результат.

Проблема проверки невыполнимости формулы наталкивается в действительности на тройное препятствие, связанное с понятием бесконечности:

- Существует бесконечно много областей интерпретации.
- Если выбранная область бесконечна, то формула допускает бесконечно много конкретизаций.
- Если выбранная область бесконечна, то существует бесконечно много интерпретаций относительно этой области. Каждая из них сопоставляет некоторое значение бесконечному множеству литер.

Результаты Эрбрана устраняют первое препятствие: можно рассматривать только эрбранову область. Если эта область конечна, исчезают и два последних препятствия.

Трудный (впрочем, часто встречающийся) случай, когда эрбранова область бесконечна. Тогда можно рассмотреть последовательность $S = (A_1, \dots, A_n, \dots)$ фундаментальных конкретизаций исследуемой формулы A . Положим $S_i = \{A_1, \dots, A_i\}$. Известно, что A (или, что то же самое, S) невыполнима тогда и только тогда, когда одна из S_i невыполнима. Таким образом, можно последовательно проверять выполнимость S_i для возрастающих значений i . Если A невыполнима, процесс закончится обнаружением невыполнимости некоторой S_i при очередной проверке. Этот процесс априори очень неэффективен.

Замечание. Можно перечислять конкретизации дизъюнктов вместо конкретизаций формул.

Замечание. Этот метод допускает применение к проверке выполнимости счетного множества формул $\{A^i | i \in N\}$. Действительно, можно объединить все конкретизации для A^i в одну последовательность (счетное объединение счетных множеств счетно). Это показывает, что счетное множество формул выполнимо тогда и только тогда, когда все его конечные подмножества выполнимы. Мы получили ослабленную версию теоремы компактности из предыдущего параграфа (теорема 1.11).

1.2.11. Алгоритм Куайна, Девиса и Патнема

Цель этого параграфа состоит в обобщении алгоритма Куайна и стратегии Девиса и Патнема на исчисление предикатов. Из § 1.1.11 мы знаем, как проверять выполнимость нормальных форм исчисления высказываний. Здесь будет показано, что проблема выполнимости клаузуальной формы в исчислении предикатов не является существенно иной.

Прежде всего известно, что такая формула невыполнима тогда и только тогда, когда ее эрбрановы интерпретации невыполнимы, т. е. тогда и только тогда, когда множество фундаментальных конкретизаций этой формулы невыполнимо. Естественно, можно рассмотреть фундаментальные конкретизации дизъюнктов, а не формул.

Первый пример из § 1.2.10 дает следующее множество фундаментальных конкретизаций:

$$S = \{\neg P(c) \vee Q(c), \neg Q(c) \vee R(c), P(c), \neg R(c)\}.$$

Стратегия Девиса и Патнема (§ 1.1.11) приводит к выбору, например, литеры $P(c)$ ввиду существования некоторого дизъюнкта, сводящегося к этой литературе. Невыполнимость S сводится к невыполнимости

$$S_1 = \{Q(c), \neg Q(c) \vee R(c), \neg R(c)\}.$$

Тогда выбираем $Q(c)$, что приводит к множеству

$$S_2 = \{R(c), \neg R(c)\}.$$

Так как S_2 , очевидно, невыполнимо, делаем вывод о невыполнимости исходной формулы.

Второй пример из § 1.2.10 сложнее, так как множество фундаментальных конкретизаций дизъюнктов бесконечно. Имеем

$$\begin{aligned} S = & \{P(a), \neg P(b)\} \cup \\ & \cup \{(\neg P(f^{(n)}(a)) \vee P(f^{(n+1)}(a))) : n \in N\} \cup \\ & \cup \{(\neg P(f^{(n)}(b)) \vee P(f^{(n+1)}(b))) : n \in N\}. \end{aligned}$$

Обобщение алгоритма Девиса и Патнема на бесконечный случай возможно, но не представляет большого интереса.

1.2.12. Фундаментальная резолюция

Как и алгоритм Девиса и Патнема, метод резолюций непосредственно обобщается на исчисление предикатов в случае, когда эрбранова область конечна. Вот исследование первого примера из § 1.2.10 в обозначениях § 1.1.13:

- | | |
|--------------------------|------------------------|
| 1. $\neg P(c) \vee Q(c)$ | элемент из S . |
| 2. $P(c)$ | элемент из S . |
| 3. $Q(c)$ | резольвента для 1 и 2. |
| 4. $\neg Q(c) \vee R(c)$ | элемент из S . |
| 5. $R(c)$ | резольвента для 3 и 4. |
| 6. $\neg R(c)$ | элемент из S . |
| 7. \perp | резольвента для 5 и 6. |

Если эрбранова область бесконечна, метод резолюций остается применимым, по крайней мере в принципе. В действительности известно, что если множество дизъюнктов невыполнимо, то оно допускает невыполнимое конечное подмножество. Второй пример из § 1.2.10 исследуется так:

- | | |
|-----------------------------|------------------------|
| 1. $P(a)$ | элемент из S . |
| 2. $\neg P(b)$ | элемент из S . |
| 3. $\neg P(a) \vee P(f(a))$ | элемент из S . |
| 4. $P(f(a))$ | резольвента для 1 и 3. |

5. $\neg P(f(a)) \vee P(f^{(2)}(a))$ элемент из S .
 6. $P(f^{(2)}(a))$ резольвента для 4 и 5.

Таким образом, можно порождать дизъюнкты $P(f^{(n)}(a))$ для всех значений n , но невозможно породить другие одноэлементные дизъюнкты или пустой дизъюнкт. Следовательно, исходное множество выполнимо.

Представленный здесь метод резолюций называется *фундаментальным*, так как он использует фундаментальные конкретизации дизъюнктов.

1.2.13. Унификация

Так как дизъюнкт может допускать бесконечно много конкретизаций, то фундаментальный метод резолюций непременно неэффективен. Идея Робинсона состоит в том, чтобы работать прямо с самими дизъюнктами без явного рассмотрения их фундаментальных конкретизаций. Эта идея требует использования операции *унификации* [14], [65].

Прежде чем объяснить, о чем идет речь, отметим, что унификация — это основной механизм при выполнении инструкций в логическом программировании (разд. 5.1 и гл. 6).

Пусть два дизъюнкта

$$C_1 = \{l_1, \dots\} \text{ и } C_2 = \{\neg l_2, \dots\}$$

принадлежат множеству S . Посредством переименования можно исключить одновременное появление одной и той же переменной в C_1 и C_2 . Предположим, что литеры l_1 и l_2 являются *унифицируемыми*, т. е. они обладают общей фундаментальной конкретизацией. Из каждой пары фундаментальных конкретизаций

$$C'_1 = \{l'_1, \dots\} \text{ и } C'_2 = \{\neg l'_2, \dots\},$$

таких, что $l'_1 = l'_2 = l'$, получается резольвента

$$R' = (C'_1 \setminus \{l'\}) \cup (C'_2 \setminus \{\neg l'\}).$$

Каждый дизъюнкт такого типа является логическим следствием из C_1 и C_2 . Задача состоит в том, чтобы найти такой дизъюнкт R , фундаментальные конкретизации которого были бы в точности конкретизациями того типа, как для R' . Этот дизъюнкт R «представлял бы» подходящим образом множество таких конкретизаций. Обозначим через l_u наибольшую нижнюю грань пары $\{l_1, l_2\}$ относительно порядка $<$, введенного в § 1.2.6. Имеем $l' < l_1$ и $l' < l_2$, что эквивалентно $l' < l_u$. По определению l_u существуют подстановки σ_1 и σ_2 , такие что $\sigma_1[l_1] = l_u$ и $\sigma_2[l_2] = l_u$. Так как никакая переменная не появляется одновременно в l_1 и l_2 , то имеем также $l_u = \sigma_u[l_1] = \sigma_u[l_2]$ для $\sigma_u = \sigma_1 \cup \sigma_2$. Подстановка σ_u является *наиболее общим унификатором* (НОУ) литер l_1 и l_2 (это обозначение не приводит к единственности σ_u).

Резольвентным дизъюнктом R с требуемым свойством будет

$$R = (\sigma_u[C_1] \setminus \{l_u\}) \cup (\sigma_u[C_2] \setminus \{\neg l_u\}).$$

Два атома унифицируемы, если они построены из одной предикатной константы, примененной к попарно унифицируемым термам. Таким образом, достаточно рассмотреть унификацию двух термов. Если один из них — переменная, то унификация получается мгновенно. В противном случае для унифицируемости два терма должны быть построены из одной и той же функциональной константы, примененной к попарно унифицируемым термам.

Очевидно, получается (рекурсивный) алгоритм унификации — очень простой и линейной сложности относительно числа термов. Все-таки следует указать на одну трудность. В частном случае, когда один из термов есть переменная x , а другой содержит x , но не сводится к x , унификация, конечно же, невозможна. Систематическая проверка невхождения некоторой переменной в терм, подлежащий унификации с этой переменной, приводит к неэффективному алгоритму. Тем не менее можно в некоторой мере найти средство для решения этой проблемы. Проверку невхождения часто опускают в логическом программи-

ровании, основные принципы которого будут изложены в § 1.2.15.

Замечание. Формально можно доказать, что НОУ для x и $f(x)$ является $f(f(\dots(x)\dots))$. В некоторых случаях этому «бесконечному терму» можно придать вполне определенный смысл.

В исчислении высказываний предпочтительно было работать с приведенными дизъюнктами, т. е. с дизъюнктами без повторяющихся литер (§ 1.1.12). Естественно, это остается верным во всем исчислении предикатов, но нужно учесть одну особенность: приведенный дизъюнкт может допускать неприведенную конкретизацию. Например, $P(x, a) \vee P(f(y), z)$ допускает $P(f(y), a) \vee P(f(y), a)$. Действительно, $\{(x, f(y)), (z, a)\}$ есть НОУ двух литер исходного дизъюнкта. Устранив повторение в результирующем дизъюнкте, получим *фактор* исходного дизъюнкта. Всякий дизъюнкт имплицитно имеет любой из своих факторов.

1.2.14. Метод резолюций

С помощью унификации можно легко распространить алгоритм резолюций (§ 1.1.12) на исчисление предикатов. Пусть S — множество дизъюнктов, исследуемое на невыполнимость. Алгоритм таков:

Пока $L \notin S$,
искать l_1, l_2, s_1, s_2 , такие, что
 s_1 и s_2 есть дизъюнкты или факторы
дизъюнктов;
 $l_1 \in s_1, \neg l_2 \in s_2$ и l_1, l_2 унифицируемы,
вычислить резольвентный дизъюнкт r ,
заменить S на $S \setminus \{l_1, l_2\} \cup \{r\}$.

Преобразованный таким способом метод резолюций полностью остается в силе.

Замечание. Напомним, что все переменные, входящие в дизъюнкт, связанные и их можно переименовывать.

Для примера применим метод резолюций к доказательству того, что если G — группа, каждый элемент

которой является обратным самому себе, то G коммутативна.¹⁾ Гипотезы и заключение такие:

$$H_1: \forall x \forall y \forall z [(x \cdot y) \cdot z = x \cdot (y \cdot z)],$$

$$H_2: \forall x [x \cdot e = x = e \cdot x],$$

$$H_3: \forall x [x \cdot x = e],$$

$$C: \forall x \forall y [x \cdot y = y \cdot x].$$

Для исключения предиката равенства, не введенного в нашем изложении данного метода, полагаем

$$P(x, y, z) =_{\text{def}} x \cdot y = z.$$

Представив гипотезы и отрицание заключения в сколемовской форме, получим множество дизъюнктов, исследуемое на невыполнимость:

1. $\neg P(x, y, u) \vee \neg P(y, z, v) \vee \neg P(u, z, w) \vee P(x, v, w).$
2. $\neg P(x, y, u) \vee \neg P(y, z, v) \vee \neg P(x, v, w) \vee P(u, z, w).$
3. $P(x, e, x).$
4. $P(e, x, x).$
5. $P(x, x, e).$
6. $P(a, b, c).$
7. $\neg P(b, a, c).$

Первые два дизъюнкта означают ассоциативность. Точнее, первый гласит, что $(x \cdot y) \cdot z$ можно переписать как $x \cdot (y \cdot z)$, тогда как второй — что $x \cdot (y \cdot z)$ можно переписать как $(x \cdot y) \cdot z$. Отметим также, что отношение $a \cdot b \neq b \cdot a$ эквивалентно формуле $\exists y [a \cdot b = y \wedge b \cdot a \neq y]$, сколемовская форма которой есть $a \cdot b = c \wedge b \cdot a \neq c$.

Пустой дизъюнкт \perp можно получить так, как показано ниже. (В правой части списка указаны но-

¹⁾ Этот пример неоднократно использовался для иллюстрации различных вариантов метода резолюций.

мера дизъюнктов, над которыми осуществляется резолюция, а также используемые унификаторы.)

$$8. \neg P(x, z, v) \vee \neg P(e, z, w) \vee P(x, v, w) \quad 1_{\{(y, x), (u, e)\}}, 5.$$

$$9. \neg P(b, z, v) \vee \neg P(a, v, w) \vee P(c, z, w) \quad 2_{\{(x, a), (y, b), (u, c)\}}, 6.$$

$$10. \neg P(x, z, v) \vee P(x, v, z) \quad 4_{\{(x, z)\}}, 8_{\{(w, z)\}}.$$

$$11. \neg P(a, e, w) \vee P(c, b, w) \quad 5_{\{(x, b)\}}, 9_{\{(z, b), (v, e)\}}.$$

$$12. P(c, b, a) \quad 3_{\{(x, a)\}}, 11_{\{(w, a)\}}.$$

$$13. P(c, a, b) \quad 10_{\{(x, c), (z, b), (v, a)\}}, 12.$$

$$14. \neg P(x, y, u) \vee \neg P(x, e, w) \vee P(u, y, w) \quad 2_{\{(v, e), (z, y)\}}, 5_{\{(x, y)\}}.$$

$$15. \neg P(x, y, u) \vee P(u, y, x) \quad 3, 14_{\{(w, x)\}}.$$

$$16. P(b, a, c) \quad 13, 15_{\{(x, c), (y, a), (u, b)\}}.$$

$$17. \text{Л} \quad 7, 16.$$

Правило *согласия* двойственно правилу резолюций. Если литеры l_1 и l_2 допускают НОУ σ , то *согласием конъюнкций* $l_1 \wedge C_1$ и $\neg l_2 \wedge C_2$ является конъюнкция $\sigma[C_1] \wedge \sigma[C_2]$.

Правила согласия и резолюций обобщаются для неклаузуальных форм. Пусть A_1 и A_2 — формулы исчисления предикатов, содержащие соответственно литеры l_1 и l_2 . Предполагаем, что l_1 и l_2 унифицируемы. Пусть σ — НОУ и $l = \sigma(l_1) = \sigma(l_2)$. В обозначениях § 1.1.15 операторы согласия и резолюций определяют следующим образом:

$$\top_l(\sigma[A_1], \sigma[A_2]) =_{\text{def}} \sigma[A_1]^{l=\text{Л}} \wedge \sigma[A_2]^{l=\text{И}},$$

$$\perp_l(\sigma[A_1], \sigma[A_2]) =_{\text{def}} \sigma[A_1]^{l=\text{Л}} \vee \sigma[A_2]^{l=\text{И}}.$$

Пример доказательства от противного с использованием неклаузуальной резолюции \perp приведен в § 3.1.20.

1.2.15. Принцип логического программирования

Понятие хорновского дизъюнкта, введенное в § 1.1.16 для исчисления высказываний, очевидным

образом распространяется на исчисление предикатов. Метод резолюций — хорошее средство проверки выполнимости множества хорновских дизъюнктов в исчислении предикатов при условии использования подходящей стратегии выбора. Найдено много интересных стратегий, некоторые из них описаны в [63].

Алгоритмические свойства некоторой функции можно представить множеством дизъюнктов и использовать метод резолюций для вычисления значений этой функции. Этот принцип *логического программирования* мы проиллюстрируем простым примером из арифметики.

Хорошо известный метод вычисления наибольшего общего делителя двух натуральных чисел — алгоритм Евклида. Его основу можно, пользуясь формализмом § 1.1.17, резюмировать тремя следующими дизъюнктами:

1. $pgcd(x, x, x) : -$.
2. $pgcd(x, y, z) : - x > y, pgcd(x - y, y, z)$.
3. $pgcd(x, y, z) : - y > x, pgcd(x, y - x, z)$.

Предикатная форма $pgcd(x, y, z)$ истинна, если z является наибольшим общим делителем x и y . Бинарный предикат « $>$ » и бинарная операция « $-$ » имеют свой обычный арифметический смысл. Для вычисления НОД двух натуральных чисел, например 4 и 6, добавим к описанию алгоритма четвертый дизъюнкт:

4. $\text{Л} : - pgcd(4, 6, z)$.

Как и в исчислении высказываний, этот набор из четырех дизъюнктов будет рассматриваться как некая грамматика, имеющая Л исходным символом. Будем искать вывод пустой цепочки ϵ . Для этого необходима последовательность унификаций. В частности терм, которым будет конкретизирована переменная z , даст решение (задачи). В данном слу-

чае имеем следующую последовательность преобразований:

$$\text{Л}(4) \text{ } pgcd(4, 6, z) (3) 6 > 4, \text{ } pgcd(4, 6 - 4, z)$$

$$pgcd(4, 2, z) (2) 4 > 2, \text{ } pgcd(4 - 2, 2, z)$$

$$pgcd(2, 2, z) (1) \varepsilon [z = 2]^*$$

«Арифметико-логическая» машина, базирующаяся на принципе резолюций, может выполнить алгоритм, описанный набором хорновских дизъюнктов. Язык программирования Пролог пронизан этим принципом. Пролог вводится в разд. 5.1 и более подробно излагается в гл. 6.

2. Аксиоматические системы

2.1. Аксиоматический подход к логике

2.1.1. Введение

Понятие *аксиоматической системы* очень старо. Такая система состоит из множества аксиом, т. е. выражений, считающихся общезначимыми¹⁾, и множества *правил вывода*, т. е. механизмов, позволяющих строить новые общезначимые выражения из аксиом и уже полученных общезначимых выражений. В этом контексте построенные указанным способом общезначимые выражения называются *теоремами*. *Доказательством* (или *выводом*) теоремы называется последовательность из аксиом, правил вывода и уже доказанных теорем, позволяющая получить данную теорему.

В этой главе выражениями всегда будут формулы исчислений высказываний или предикатов.

Евклидова геометрия — древнейшая из аксиоматических теорий.

Тот факт, что в некоторой аксиоматической системе формула A является выводимой, будет обозначаться так:

$$\vdash A.$$

Вообще, если E — множество формул, то выражение

$$E \vdash A$$

означает, что A выводима из E , т. е. что A является доказуемой, если элементы из E рассматриваются как дополнительные аксиомы. В этом контексте эле-

¹⁾ Слово *общезначимый* имеет здесь свой естественный смысл. Аксиома считается всегда истиной. Хотя конечная цель всякой аксиоматической системы — улавливать истину, это понятие можно считать лишь абстрактной системой получения одних цепочек символов из других.

менты E являются *гипотезами*. Обозначение $\vdash A$ можно считать сокращением для $\emptyset \vdash A$. Множество E называется *синтаксически невыполнимым*, если $E \vdash \perp$.

2.1.2. Свойства аксиоматических систем

Аксиоматическая система может появляться в двух весьма различных контекстах. Прежде всего, она может быть предназначена для аксиоматизации уже известной теории. Именно так и будет для исчисления высказываний: вводимые нами аксиомы лишь «дублируют» уже определенную семантику, а понятия тавтологии и теоремы должны совпасть. С другой стороны, множество аксиом может послужить отправной точкой для новой теории. Именно так обстояло дело с теорией групп. Наконец отметим, что существуют промежуточные случаи. Наиболее известный из них касается теории множеств: аксиоматическая система внесла *поправку* в интуитивную теорию.

Первое требование, которому должна удовлетворять аксиоматическая система, заключается в невозможности вывода отрицания уже доказанного выражения (которое, следовательно, считается общезначимым).

Можно также пожелать, чтобы система была *минимальной*, т. е. не содержащей бесполезных (лишних) аксиом и правил вывода. Точнее, некоторое выражение *независимо* от аксиоматической системы, если его нельзя вывести с помощью этой системы. В минимальной системе каждая аксиома независима от остальной системы. Вопрос о независимости постулата о параллельных в аксиоматической системе Евклида неотступно преследовал весь математический мир в течение двух тысяч лет. Наконец, его независимость в прошлом веке была доказана изящно и полностью. Это было сделано путем построения математических моделей, в которых истинны все геометрические аксиомы, за исключением названного постулата.

Обратимся снова к аксиоматическим системам вообще. Для начала нам достаточно выделить те

свойства, которыми должна обладать аксиоматическая система, соответствующая исчислению высказываний (предикатов). Это, прежде всего, свойство адекватности, т. е. все теоремы должны быть общезначимыми формулами. Требуется также полнота, т. е. взаимность адекватности: всякая общезначимая формула должна быть теоремой. Система *адекватна*, если для любой формулы A имеем

$$\vdash A \Rightarrow \models A.$$

Соответственно система *полна*, если для любой формулы A

$$\models A \Rightarrow \vdash A.$$

Заметим, что никакая адекватная логическая система не может порождать одновременно некоторое выражение и его отрицание. В действительности адекватная система позволяет выводить¹⁾ только тавтологии. Аксиоматический подход к логике высказываний (предикатов) изложен в §§ 2.1.3—2.1.6 (2.1.8—2.1.9).

2.1.3. Простая аксиоматическая система исчисления высказываний

Рассмотрим следующие три схемы формул:

$$(A1) (X \supset (Y \supset X)),$$

$$(A2) ((X \supset (Y \supset Z)) \supset ((X \supset Y) \supset (X \supset Z))).$$

$$(A3) ((\neg X \supset \neg Y) \supset ((\neg X \supset Y) \supset X)).$$

Всякий раз, когда в любой из этих схем X , Y , Z заменяются произвольными формулами, получается тавтология. Следовательно, эти схемы формул могут служить схемами аксиом в некоторой адекватной системе.

Правило вывода, называемое обычно *modus ponens*²⁾, формулируется следующим образом:

$$(MP) \quad \begin{array}{l} \text{Если } X \text{ и } (X \supset Y) \text{ — теоремы,} \\ \text{то } Y \text{ есть теорема.} \end{array}$$

¹⁾ «Выводить» здесь синоним для «доказывать».

²⁾ Иное название — *правило отделения*. — *Прим. перев.*

Это правило вывода, добавленное к нашим схемам аксиом, дает аксиоматическую систему. Система остается адекватной, так как наше правило вывода сохраняет общезначимость. Точнее, правило (*МР*) семантически корректно, так как оно «заменяет» слово «тавтология» на «теорема».

Данный результат, выражающий включение множества выводимых формул в множество общезначимых, служит примером *метатеоремы*. В нашем изложении все выражения, перед которыми стоят слова *теорема*, *лемма* или *следствие*, фактически являются метатеоремами.

Каждая аксиома здесь независима от совокупности остальных аксиом. Чтобы установить невыводимость аксиомы *A* из некоторого множества аксиом *E*, пользуются методом моделей. Он состоит в приписывании конкретных значений символам, входящим в *A* и *E*. Затем указывается структура, в которой элементы истинны, тогда как *A* ложно. Доказательства независимости для обсуждаемой системы содержатся, например, в [72].

Рассматриваемая нами система содержит только связки \neg и \supset . Следовательно, она не позволяет судить о формулах, содержащих другие связки. Это, однако, не является существенным ограничением, если рассматривать связки \neg и \supset как первичные, а три другие связки — как полученные из них по формулам (1.3) из § 1.1.5. Вопрос о полноте этой системы отложим до § 2.1.5.

2.1.4. Несколько интересных теорем

Докажем для примера, что $(p \supset p)$ — теорема введенной в предыдущем параграфе аксиоматической системы.

1. $\vdash (p \supset ((p \supset p) \supset p))$ A1
2. $\vdash ((p \supset ((p \supset p) \supset p)) \supset ((p \supset (p \supset p)) \supset (p \supset p)))$ A2
3. $\vdash ((p \supset (p \supset p)) \supset (p \supset p))$ 1, 2, МР
4. $\vdash (p \supset (p \supset p))$ A1
5. $\vdash (p \supset p)$ 4, 3, МР

Каждая строка формального доказательства содержит порядковый номер, саму формулу и обоснование ее вывода. Существуют три типа обоснования:

- если формула является гипотезой, обоснование содержит имя (номер) этой гипотезы,
- если формула получается из какой-либо схемы аксиом, обоснование содержит имя (номер) этой схемы,
- если формула получена применением некоторого правила к ранее полученным формулам, обоснование содержит порядковые номера строк этих формул и имя правила.

Этого нехитрого примера достаточно для демонстрации возможной нудности доказательств. Ниже приведены несколько элементарных теорем без формальных доказательств.

$$p \supset p,$$

$$p \supset \neg \neg p,$$

$$\neg \neg p \supset p,$$

$$(\neg p \supset \neg q) \supset (q \supset p),$$

$$(p \supset q) \supset ((\neg p \supset q) \supset q).$$

Построение формальных доказательств можно упростить, привлекая одну достаточно полезную метатеорему, принадлежащую Эрбрану и Тарскому. Мы приведем ее здесь в довольно урезанном виде. Общую версию см., например, в [72].

*Метатеорема дедукции*¹⁾ утверждает, что необходимым и достаточным условием выводимости A из гипотез $E \cup \{B\}$ является выводимость $(B \supset A)$ из E . Этот результат можно записать так:

$$E, B \vdash A \Leftrightarrow E \vdash (B \supset A).$$

Достаточность доказывается простым применением правила *modus ponens*. Необходимость устанавливается индукцией по длине вывода $(E, B \vdash A)$. Детальные рассуждения приведены в [72]. Эта мета-

¹⁾ Иные названия — *теорема дедукции* или *теорема о дедукции*. — Прим. ред.

теорема значительно упрощает доказательства. Она формализует обычный подход того математика, который, чтобы установить, что гипотеза влечет некоторое следствие, прямо устанавливает это следствие в предположении истинности гипотезы. Например, доказательство того, что $\vdash (p \supset p)$ сводится к доказательству (тривиальному) $p \vdash p$.

Замечание. Метатеорема дедукции тривиально верна в адекватной полной системе, но может оставаться верной и в неполной системе.

Замечание. Входящие в доказательство формулы, вообще говоря, не являются теоремами (за исключением частного случая, когда гипотезы отсутствуют).

Формальное доказательство должно быть конечным. Если же множество E бесконечно, обозначение $E \vdash A$ означает, что в E существует такое конечное подмножество E' , что $E' \vdash A$.

Другой полезной метатеоремой является следующее *правило замены*:

Если X , $(Y \supset Z)$ и $(Z \supset Y)$ теоремы, если Y есть X или подформула формулы X , если U получается заменой в X вхождения Y на Z , то U является теоремой.

Принцип *одновременной подстановки* (см. § 1.1.2) можно представить в виде следующей метатеоремы:

Если p — высказывание, $X(p)$ — теорема и A — формула, если $X(A)$ получается заменой из $X(p)$ путем замены всех вхождений p на A , то $X(A)$ — теорема.

Замечание. Эти два правила становятся очевидными, если слово «теорема» заменить на выражение «общезначимая формула». Значит, они останутся верными в адекватной полной аксиоматической системе, в том числе и в используемой здесь (§ 2.1.5).

Как приложение формализуем и обоснуем заново принцип рассуждений, базирующийся на разборе случаев.

Лемма 2.1. Если $E, B \vdash A$ и $E, \neg B \vdash A$, то $E \vdash A$.

Доказательство. Имеем последовательно

- | | |
|--|--------------------|
| 1. $E, B \vdash A$ | гипотеза. |
| 2. $E \vdash (B \supset A)$ | 1, дедукция. |
| 3. $\vdash ((p \supset q) \supset ((\neg p \supset q) \supset q))$ | теорема. |
| 4. $\vdash ((B \supset A) \supset ((\neg B \supset A) \supset A))$ | 3, подстановка. |
| 5. $E \vdash ((\neg B \supset A) \supset A)$ | 2, 4, МР. |
| 6. $E, \neg B \vdash A$ | гипотеза. |
| 7. $E \vdash (\neg B \supset A)$ | 6, дедукция. |
| 8. $E \vdash A$ | 7, 5 МР. \square |

Замечание. Это доказательство демонстрирует автоматическое получение вывода $E \vdash A$ из выводов $E, B \vdash A$ и $E, \neg B \vdash A$.

2.1.5. Полнота

Набросаем основные штрихи доказательства полноты, принадлежащего Кальмару (подробности см., например, в [72]). Ограничимся случаем формул, содержащих лишь связки \neg и \supset .

Лемма 2.2. Пусть A — формула, построенная из высказываний p_1, \dots, p_n и связок \neg, \supset . Пусть I — некоторая интерпретация. Пусть p'_k означает p_k при $I(p_k) = \text{И}$ и $\neg p_k$ при $I(p_k) = \text{Л}$ и, аналогично, A' есть A при $I(A) = \text{И}$ и $\neg A$ при $I(A) = \text{Л}$.

Тогда

$$\{p'_1, \dots, p'_n\} \vdash A'.$$

Доказательство. Применим индукцию по числу связок в A . Если оно равно 0, то A сводится к одному из высказываний p_k и результат очевиден. В противном случае A является формулой вида $\neg B$ или $(B \supset C)$. Предположив, что результат верен для B и C , легко доказать в каждом случае, что он верен для A . \square

Следствие. Если формула A из леммы 2.2 — тавтология и если p'_k — общее обозначение для p_k и $\neg p_k$, то, какова бы ни была n -ка p'_1, \dots, p'_n ,

$$\{p'_1, \dots, p'_n\} \vdash A$$

Теорема 2.3. Введенная в § 2.1.3 аксиоматическая система полна.

Доказательство. Пусть A — тавтология, построенная из множества высказываний $\{p_1, \dots, p_n\}$. По лемме 2.2 имеется 2^n выражений $\{p'_1, \dots, p'_n\} \vdash A$, где p'_k есть либо p_k , либо $\neg p_k$. Отсюда следует

$$\{p'_1, \dots, p'_{n-1}\} \vdash (p_n \supset A),$$

$$\{p'_1, \dots, p'_{n-1}\} \vdash (\neg p_n \supset A).$$

Значит, по лемме 2.1 получается 2^{n-1} выражений

$$\{p'_1, \dots, p'_{n-1}\} \vdash A.$$

Конечный результат $\vdash A$ получается после осуществления n таких шагов. \square

Можно также установить полноту некоторой аксиоматической системы исчисления высказываний, показав, что *метод резолюций есть метатеорема этой системы*. В действительности надо доказать три метатеоремы:

- Если C — КНФ, соответствующая A , то $\vdash C \equiv A$.
- Если C и C' — нормальные формы, отличающиеся лишь порядком дизъюнктов и/или порядком литер в пределах одного дизъюнкта, и/или числом повторений одной и той же литеры в дизъюнкте, то $\vdash C \equiv C'$.
- Правило резолюций:

$$E, \quad p \vee X, \quad \neg p \vee Y \vdash X \vee Y.$$

Доказательства этих метатеорем простые, но длинные. Вот, например, основные этапы доказательства третьего пункта:

$$X \vdash X \vee Y,$$

$$Y \vdash X \vee Y,$$

$$E, (\neg p \supset X), (p \supset Y), \neg p \vdash X,$$

$$E, (\neg p \supset X), (p \supset Y), \neg p \vdash X \vee Y,$$

- $E, (\neg p \supset X), (p \supset Y), p \vdash Y,$
 $E, (\neg p \supset X), (p \supset Y), p \vdash X \vee Y,$
 $E, (\neg p \supset X), (p \supset Y) \vdash X \vee Y,$
 $E, (p \vee X), (\neg p \vee Y) \vdash X \vee Y.$

2.1.6. Польза аксиоматических систем

Выяснилось, что доказательство тавтологий — более трудоемкое дело, чем просто проверка общезначимости. Следовательно, может возникнуть вопрос о пользе аксиоматических систем. Действительно, аксиоматизация исчисления высказываний сама по себе бесполезна, так как существуют относительно эффективные алгоритмические средства для определения того, является или нет формула тавтологией.

С этой точки зрения исчисление высказываний представляет собой исключение. Другим замечательным исключением является евклидова геометрия. Истины этой теории были классически доказаны в аксиоматической системе Евклида. Доказательства эти часто весьма изящны. Напротив, доказательства, опирающиеся на аналитический метод Декарта, довольно механистичны.

Честно говоря, существует много теорий, для которых аксиоматическая точка зрения оказывается самой простой, даже единственно возможной. Сошлемся, в частности, на логику предикатов, теорию чисел и теорию групп. Ознакомление с некоторыми аксиоматическими системами исчисления высказываний является хорошей подготовкой к изучению более сложных аксиоматических систем, таких, как логика предикатов (§§ 2.1.8—2.1.9).

С другой стороны, понятие аксиоматической системы исчисления высказываний интересно само по себе. Наряду с классической логикой высказываний были разработаны различные альтернативные теории, оказавшиеся полезными в некоторых контекстах.

Теория высказываний состоит в выделении некоего подмножества из множества формул. Ниоткуда не следует, что множество тавтологий (или теорем), определенное в разд. 1.1, является единственным ин-

тересным подмножеством. И действительно, были выделены и изучаются другие подмножества формул — интересные либо с семантической точки зрения (многозначные логики), либо с аксиоматической (в особенности интуиционистская логика). Изучение этих неклассических логик выходит за рамки настоящей главы. Отметим только, что наибольшие споры вызывает тавтология $(p \vee \neg p)$, называемая обычно *законом исключенного третьего*. Этот принцип позволяет, например, математику провести следующее рассуждение:

Множество алгебраических чисел счетно,
множество действительных чисел несчетно,
следовательно, существуют неалгебраические действительные числа.

Обоснование данного рассуждения в некоторых отношениях недостаточно удовлетворительное, потому что оно не указывает неалгебраического действительного числа. Доведенная до крайности, такая критическая позиция приводит к отказу от важного принципа теории доказательств: ставится под сомнение неявно фигурирующий в доказательстве принцип исключенного третьего. Это привело к развитию логик с более чем двумя значениями. В § 3.1.16 мы встретимся с примером использования трехзначной логики при задании семантики некоторых модальных логик.

Двузначность логики высказываний может стеснять администратора базы данных, информация p в которой может иметь три состояния:

- Истинно: p подтверждается в базе данных (БД).
- Ложно: p явно отрицается в БД.
- Неопределенно: по поводу p ничего не сказано в БД.

Можно было бы ввести четвертое состояние — наличие в БД противоречивой информации по поводу p . Обычно это запрещено.

Сделаем последнее замечание о неклассических логиках. Они определяют все подмножества множе-

ства тавтологий (по крайней мере те, которые используют классический язык). Неклассические логики получаются обеднением множества тавтологий, являющегося в некотором роде максимальным¹⁾.

Присоединение к множеству тавтологий какой-либо необщезначимой формулы неизбежно приводит к невыполнимости системы: множество «теорем» совпадает тогда с множеством всех формул. Предположим, что некий не являющийся общезначимым дизъюнкт, например $(p \vee \neg q)$, был бы объявлен общезначимым и добавлен к множеству аксиом. Правило одновременной подстановки позволяет получить дизъюнкт $(p \vee \neg \neg p)$ путем замены высказывания q формулой $\neg p$. Из этого последнего дизъюнкта легко выводится p , так как $((p \vee \neg \neg p) \supset p)$ является теоремой. Наконец, из p выводится какая угодно формула A (по правилу одновременной подстановки).

Вторгшаяся формула буквально «отравила» систему. Эта хрупкость присуща не только исчислению высказываний, но распространяется на большинство формальных систем. Тем самым она угрожает и предостерегает: логик будет осторожен при выборе аксиом и правил вывода, зная о санкциях за каждую ошибку такого рода.

Замечание. Не следует смешивать случаи, когда некоторый дизъюнкт рассматривается как истинный (что часто приходится делать), с теми случаями, когда какой-либо дизъюнкт считается общезначимым. Например, вывод

$$(p \vee \neg q) \vdash A$$

некорректен, если A — произвольная формула, но он корректен во всех тех случаях, когда A — логическое следствие из $(p \vee \neg q)$.

2.1.7. Система натурального²⁾ вывода

Существуют различные способы проведения доказательств в логике. В аксиоматической системе из

¹⁾ Это не касается *немонотонных логик*, которые вводятся в гл. 4.

²⁾ Иногда говорят «система естественного вывода». — *Прим. ред.*

§ 2.1.3 использовалось одно правило вывода *modus ponens* и несколько аксиом. Такой тип доказательства (который, исходя из упомянутой аксиоматической системы, приводит к требуемому результату) называется «доказательством в смысле Гильберта».

Аксиоматическая система, рассматриваемая в этом параграфе, не содержит аксиом, зато использует немало правил вывода. Тип доказательства, связанный с такой системой, называется «доказательством в смысле Генцена». Эта аксиоматическая система введена Генценом (см. [3], [53], [65]). Правила вывода записываются в следующей форме:

$$\frac{S_1 \dots S_n}{S}$$

Это обозначение выражает возможность вывода S из S_1, \dots, S_n . Каждой связке сопоставлены правила введения и удаления. Они задают семантику связок и указывают «способ применения». Название естественный (или натуральный) вывод дано этой системе потому, что используемый тип рассуждений приближается к обычному человеческому (естественному) рассуждению.

Метод натурального вывода позволяет оперировать с формальными объектами, представляющими рассуждения. В логике рассуждение можно представить выражением типа

H влечет C ,

где H — последовательность формул, C — формула. Рассуждение общезначимо, если его заключение является логическим следствием его гипотез (§ 1.1.6). Вывод или доказательство есть формальное представление рассуждения. Метасимвол \Rightarrow является формальным эквивалентом слова «влечет». Вывод

$$H \Rightarrow C$$

общезначимый, если C — логическое следствие H .

Множество правил оперирования с выводами представлено ниже, E — множество формул, A , B и C — формулы. Выражение E, A служит сокращением для $E \cup \{A\}$. В частности, если E содержит A , то E

эквивалентно E, A . Как обычно, \mathbf{J} представляет произвольную невыполнимую формулу.

Вот для начала два правила (называемые *базисными*):

$$\frac{}{E, A \Rightarrow A} \text{ и } \frac{E \Rightarrow A}{E, B \Rightarrow A},$$

формализующие *принцип монотонности*. Первое из них означает, что всякий вывод, заключение которого совпадает с одной из гипотез, обязательно общезначим. Второе гласит, что присоединение гипотезы к общезначимому выводу дает общезначимый вывод. А теперь выпишем правила введения связок:

$$(\wedge) \quad \frac{E \Rightarrow A \quad E \Rightarrow B}{E \Rightarrow A \wedge B}$$

$$(\vee) \quad \frac{E \Rightarrow A}{E \Rightarrow A \vee B} \quad \frac{E \Rightarrow B}{E \Rightarrow A \vee B}$$

$$(\supset) \quad \frac{E, A \Rightarrow B}{E \Rightarrow A \supset B}$$

$$(\neg) \quad \frac{E, A \Rightarrow \mathbf{J}}{E \Rightarrow \neg A}$$

$$(\equiv) \quad \frac{E, A \Rightarrow B \quad E, B \Rightarrow A}{E \Rightarrow A \equiv B}$$

Наконец, приведем правила удаления связок:

$$(\wedge) \quad \frac{E \Rightarrow A \wedge B}{E \Rightarrow A} \quad \frac{E \Rightarrow A \wedge B}{E \Rightarrow B}$$

$$(\vee) \quad \frac{E \Rightarrow A \vee B \quad E, A \Rightarrow C \quad E, B \Rightarrow C}{E \Rightarrow C}$$

$$(\supset) \quad \frac{E \Rightarrow A \supset B}{E, A \Rightarrow B}$$

$$(\neg) \quad \frac{E \Rightarrow A \quad E \Rightarrow \neg A}{E \Rightarrow \mathbf{J}} \quad \frac{E \Rightarrow \neg \neg A}{E \Rightarrow A}$$

$$(\equiv) \quad \frac{E \Rightarrow A \equiv B}{E, A \Rightarrow B} \quad \frac{E \Rightarrow A \equiv B}{E, B \Rightarrow A}$$

Эта система (натурального вывода) является адекватной: если выводы в числителе произвольного правила общезначимы, то вывод в знаменателе того же правила также общезначим. Кроме того, она является полной: каждый общезначимый вывод можно получить из пустого вывода (т. е. «из ничего»), применяя конечное число подходящих правил.

В качестве примера рассмотрим заново метод доказательств разбором случаев (§ 1.1.11). Требуется доказать

$$\{h, h \supset (p \vee q), p \supset c, q \supset c\} \Rightarrow c.$$

Обозначив множество гипотез через E , имеем следующий вывод:

- | | |
|---|--|
| 1. $E, h \supset (p \vee q) \Rightarrow h \supset (p \vee q)$ | базисное правило. |
| 2. $E \Rightarrow h \supset (p \vee q)$ | $E, h \supset (p \vee q) = E$. |
| 3. $E, h \Rightarrow p \vee q$ | 2, правило \supset -удаления, |
| 4. $E \Rightarrow p \vee q$ | $E, h = E$. |
| 5. $E, p \supset c \Rightarrow p \supset c$ | базисное правило. |
| 6. $E \Rightarrow p \supset c$ | $E, p \supset c = E$. |
| 7. $E, p \Rightarrow c$ | 6, правило \supset -удаления. |
| 8. $E, q \supset c \Rightarrow q \supset c$ | базисное правило. |
| 9. $E \Rightarrow q \supset c$ | $E, q \supset c = E$. |
| 10. $E, q \Rightarrow c$ | 9, правило \supset -удаления. |
| 11. $E \Rightarrow c$ | 4, 7, 10,
правило \vee -удаления. |

Этого простого примера, пожалуй, достаточно для демонстрации того, что система натурального вывода гораздо удобнее, чем классическая аксиоматическая система из § 2.1.3.

2.1.8. Классические аксиомы для квантификации

Аксиоматическую систему из § 2.1.3 можно приспособить к исчислению предикатов без равенства. Схемы аксиом (A1), (A2) и (A3) и правило (MP)

остаются без изменений. Две дополнительные схемы дают возможность оперировать с кванторами.

Введем сокращение, которое будет использоваться при записи одной из схем аксиом. Говорят, что *терм t свободен для переменной x в формуле Q* , если ни x , ни произвольная переменная из t не квантифицированы в Q . При этом через $Q_{x/t}$ обозначается формула, полученная из Q путем одновременной замены всех вхождений x на t . Пять схем аксиом таковы:

$$(A1) \quad (P \supset (Q \supset P)),$$

$$(A2) \quad ((P \supset (Q \supset P)) \supset ((P \supset Q) \supset (P \supset R))),$$

$$(A3) \quad ((\bigwedge P \supset \bigwedge Q) \supset ((\bigwedge P \supset Q) \supset P)),$$

$$(A4) \quad \forall x (P \supset Q) \supset (P \supset \forall x Q)$$

(x не содержится в P и не является связанной в Q),

$$(A5) \quad (\forall x Q \supset Q_{x/t})$$

(x свободна для t в Q).

Второе правило вывода (правило *обобщения*) управляет квантификацией. Оба правила рассматриваемой аксиоматической системы сформулированы ниже:

(MP) Если X и $(X \supset Y)$ — теоремы,
то Y — теорема.

(G) Если x не связана в теореме P ,
то $\forall x P$ — теорема.

В этом контексте квантор существования не вводился потому, что формула $\exists x A$ считается сокращением для $\neg \forall x \neg A$.

Адекватность описанной аксиоматической системы устанавливается легко, как и в случае исчисления высказываний. Можно также установить полноту, но доказательство соответствующего факта является значительно более тонким. Довольно длинное классическое доказательство дано в [5]. Можно интуитивно убедиться в полноте системы, выведя в ней правила унификации и резолюций.

Пусть A — формула, E — конечное множество формул и Σ — аксиоматическая система. Обозначение $E \vdash_{\Sigma} A$ или просто $E \vdash A$ означает, что A можно вывести в системе Σ , к которой добавлены в качестве аксиом некоторые формулы из E .

Возникает вопрос: останется ли метатеорема дедукции справедливой в рассматриваемой здесь аксиоматической системе Σ . Точнее, надо выяснить, справедлива ли следующая эквивалентность:

$$E, B \vdash A \Leftrightarrow E \vdash (B \supset A).$$

Это соотношение неверно, если в выводе формулы A из E, B используется правило обобщения (G) по некоторой переменной, имеющей свободное вхождение в B . Например, вывод $P \vdash \forall x P$ вполне законный, тогда как импликация $(P \supset \forall x P)$, естественно, необщезначима. Но если запретить применение в выводе такой операции, то метатеорема дедукции останется справедливой. Существует довольно длинное прямое доказательство, которое проводится индукцией по сложности вывода $E, B \vdash A$.

2.1.9. Натуральный вывод в логике предикатов

Введенная в исчислении высказываний система натурального вывода (§ 2.1.7) обобщается на исчисление предикатов путем добавления правил введения и удаления кванторов общности и существования. *Правила введения* таковы:

- (\forall) $\frac{E \Rightarrow A(x)}{E \Rightarrow \forall x A(x)}$ (x не имеет свободных вхождений в E),
- (\exists) $\frac{E \Rightarrow A(t)}{E \Rightarrow \exists x A(x)}$ (терм t свободен для x в $A(x)$).

Правила удаления следующие:

- (\forall) $\frac{E \Rightarrow \forall x A(x)}{E \Rightarrow A(t)}$ (терм t свободен для x в $A(x)$),
- (\exists) $\frac{E \Rightarrow \exists x A(x)}{E \Rightarrow A(c)}$ (c не встречается ни в E , ни в $A(x)$).

В этих правилах x — переменная, c — константа, $A(x)$ — формула, в которой x не имеет связанных вхождений, t — терм и E — множество формул.

В качестве примера выведем в этой системе аксиому (A4), сформулированную в § 2.1.8. Предполагаем, что переменная x не входит в формулу P и не имеет связанных вхождений в формулу Q . Подходящий вывод таков:

1. $\forall x(P \supset Q) \Rightarrow \forall x(P \supset Q)$ базисное правило.
2. $\forall x(P \supset Q) \Rightarrow (P \supset Q)$ 1, правило \forall -удаления.
3. $\forall x(P \supset Q), P \Rightarrow Q$ 2, правило \supset -удаления.
4. $\forall x(P \supset Q), P \Rightarrow \forall xQ$ 3, правило \forall -введения.
5. $\forall x(P \supset Q) \Rightarrow (P \supset \forall xQ)$ 4, правило \supset -введения.

2.1.10. Равенство в исчислении предикатов

Введенное в § 1.2.3 исчисление предикатов содержит связку равенства. Однако, начиная с § 1.2.9, мы часто рассматривали исчисление предикатов без этой связки, что, в частности, упростило доказательство принципа резолюций (§ 1.2.12). При отсутствии связки « $=$ » отношение равенства можно моделировать и даже аксиоматизировать. Рассмотрим двуместный предикат E и три следующие формулы:

$$\forall x E(x, x),$$

$$\forall x \forall y [E(x, y) \supset E(y, x)],$$

$$\forall x \forall y \forall z [(E(x, y) \wedge E(y, z)) \supset E(x, z)].$$

Они характеризуют отношения эквивалентности, т. е. рефлексивные, симметричные и транзитивные отношения. Отношение равенства — частный случай эквивалентности. Для моделирования присущих ему свойств введем два следующих правила вывода:

$$\frac{E(s_1, t_1) \wedge E(s_2, t_2) \wedge \dots \wedge E(s_m, t_m)}{P(s_1, s_2, \dots, s_m) \equiv P(t_1, t_2, \dots, t_m)},$$

$$\frac{E(s_1, t_1) \wedge E(s_2, t_2) \wedge \dots \wedge E(s_n, t_n)}{E(f(s_1, s_2, \dots, s_n), f(t_1, t_2, \dots, t_n))}.$$

Символы P и f m -местная предикатная и n -местная функциональная константы соответственно.

Чтобы приспособить метод резолюций для исчисления предикатов с равенством, были предложены разнообразные приемы. Основной принцип, заложенный в них, состоит в возможности замены терма равным термом. Один из наиболее широко применяемых методов — *парамодуляция* (см. [94]). Заметим, что, за исключением явно искусственных случаев, доказательства обычно очень длинны.

Для иллюстрации последнего утверждения обратимся к одной классической теореме теории групп. Рассмотрим произвольную группу. Операция обозначается точкой между операндами. Нейтральный элемент обозначен через e , обратный к x — через x^{-1} . Определим x^k для всех целых k следующим образом: $x^0 = e$, $x^{n+1} = x^n \cdot x$ и $x^{-(n+1)} = x^{-n} \cdot x^{-1}$ для всех $n \geq 0$. Полезным оказывается понятие *коммутатора* двух элементов x и y — элемента $[x, y]$:

$$[x, y] =_{\text{def}} (x \cdot y) \cdot (x^{-1} \cdot y^{-1}).$$

Ясно, что два элемента перестановочны тогда и только тогда, когда их коммутатор равен e . Обратимся к следующей элементарной теореме:

Если для всех x справедливо равенство $x^3 = e$, то для всех x и всех y имеем $[[x, y], y] = e$.

Следующие формулы соответствуют гипотезам и заключению.

$$H_1 : \forall x \forall y \forall z [(x \cdot y) \cdot z = x \cdot (y \cdot z)].$$

$$H_2 : \forall x [x \cdot e = x = e \cdot x].$$

$$H_3 : \forall x [x \cdot x^{-1} = e = x^{-1} \cdot x].$$

$$H_4 : \forall x [x^3 = e].$$

$$H_5 : \forall x \forall y ([x, y] = (x \cdot y) \cdot (x^{-1} \cdot y^{-1})).$$

$$C : \forall x \forall y ([x, y], y] = e).$$

Сперва рассмотрим полуформальное доказательство. В силу ассоциативности все скобки опускаются. Общая величина $(x \cdot y) \cdot z$ и $x \cdot (y \cdot z)$ обозначается $x y z$ (это могло бы быть предметом некой леммы об обобщенной ассоциативности). В этих новых обозначениях имеем следующую лемму:

$$L : \forall x \forall y [x y x = y^{-1} x^{-1} y^{-1}].$$

Доказательство легкое:

$$\begin{array}{ll}
 yxux = e & H_4 \text{ (для } yx), \\
 yxux = y^{-1} & \text{умножили слева обе части на } y^{-1}, \\
 yxux = x^{-1}y^{-1} & \text{умножение слева на } x^{-1}, \\
 yux = y^{-1}x^{-1}y^{-1} & \text{умножение слева на } y^{-1}.
 \end{array}$$

Доказательство теоремы тоже простое:

$$\begin{array}{ll}
 yuxxuxxux = e & (yux)^3 = e, \\
 yuxxuxy^{-1}x^{-1}y^{-1} = e & \text{лемма } L, \\
 yux^{-1}yxy^{-1}x^{-1}y^{-1} = e & x^2 = x^{-1}, \\
 yux^{-1}(y^{-1}y)yxy^{-1}x^{-1}y^{-1} = e & y^{-1}y = e, \\
 (yux^{-1}y^{-1})y(yxy^{-1}x^{-1})y^{-1} = e & \text{ассоциативность,} \\
 (yux^{-1}y^{-1})y(yux^{-1}y^{-1})^{-1}y^{-1} = e & (uv)^{-1} = v^{-1}u^{-1}, \\
 [x, y]y[x, y]^{-1}y^{-1} = e & \text{определение коммутатора,} \\
 [[x, y], y] = e & \text{определение коммутатора.}
 \end{array}$$

Это доказательство можно формализовать. Дизъюнкты первой группы представляют гипотезы:

1. $E(f(f(x, y), z), f(x, f(y, z)))$.
2. $E(f(x, e), x)$.
3. $E(f(e, x), x)$.
4. $E(f(x, i(x)), e)$.
5. $E(f(i(x), x), e)$.
6. $E(f(f(x, x), x), e)$.
7. $E(c(x, y), f(f(x, y), f(i(x), i(y))))$.

Функции f , i и c представляют здесь соответственно групповую операцию, взятие обратного элемента и коммутатор. Далее, три дизъюнкта (8, 9, 10) означают, что предикат E представляет отношение эквивалентности, а пять дополнительных дизъюнктов (11, 12, 13, 14, 15) позволяют подстановку равных

термов:

8. $E(x, x)$.
9. $\neg E(x, y) \vee E(y, x)$.
10. $\neg E(x, y) \vee \neg E(y, z) \vee E(x, z)$.
11. $\neg E(u, v) \vee E(f(u, x), f(v, x))$.
12. $\neg E(u, v) \vee E(f(x, u), f(x, v))$.
13. $\neg E(u, v) \vee E(i(u), i(v))$.
14. $\neg E(u, v) \vee E(c(u, x), c(v, x))$.
15. $\neg E(u, v) \vee E(c(x, u), c(x, v))$.

Наконец, последний дизъюнкт представляет отрицание заключения:

16. $\neg E(c(c(a, b), b), e)$.

Здесь символы a и b — сколемовские константы.

Доказательство, базирующееся на классическом методе резолюций, содержит минимум 137 этапов, тогда как парамодуляция позволяет уменьшить это число до 48 [94]. Неформальное доказательство содержит менее 20 этапов, но в нем используются многочисленные «очевидные» леммы, ссылки на которые отсутствуют. Формальный подход надежнее, чем неформальные рассуждения (по крайней мере в принципе). Из-за чрезмерной длины формальные доказательства без автоматизации практически неподъемны.

2.2. Теории первого порядка

2.2.1. Введение

В этом разделе будет показано, как исчисление предикатов может служить источником теорий, позволяющих изучать некоторые специфические структуры. Мы заинтересуемся также, какие свойства можно ожидать у этих теорий. Наиболее важным из них является разрешимость, т. е. существование алгоритма, позволяющего различать истинные и ложные выражения. Тем самым необходимо критически проанализировать понятие алгоритма, его возможности и

границы. Рассматриваются и алгоритмические языки — под необычным углом зрения их выразительных возможностей. Проявляются связи между понятиями вывода и вычисления. Во многих случаях читатель должен приноравливаться к неразрешимым теориям и проблемам. Если говорить конкретнее, то, например, в общем случае невозможно алгоритмически распознать, выводима ли некая формула в данной аксиоматической системе, а также — можно ли некую функцию вычислить, используя данный алгоритм. Излагаются и комментируются (кратко) различные примеры. Логика — это прежде всего язык, а уже затем средство дедукции (построения выводов). Польза от этого языка есть, даже если дедуктивный аппарат не введен. Язык программирования Пролог (гл. 6) служит неплохой иллюстрацией этого положения.

2.2.2. Неформальные и формальные теории

Прежде чем ввести понятие формальной теории, интересно исследовать обыденный смысл слова. Любая *теория* прежде всего обладает *языком*, на котором сформулированы *выражения*. Набор *правил* позволяет отличать выражение от простой последовательности символов (если даже в некоторых случаях различие субъективно). Другие правила позволяют выделять среди выражений некие базисные (первичные, примитивные), называемые *аксиомами* или *постулатами* и считающиеся «истинными» (без необходимости ссылаться при этом на какую-либо действительность). Язык должен быть достаточно богатым и содержать, в частности, понятие отрицания. *Способ рассуждения* позволяет получать, исходя из постулатов, новые выражения, тоже считающиеся «истинными». Марксизм и теория эволюции — примеры теорий. Их язык — естественный (иногда слегка расширенный).

Исчисление предикатов можно считать формальным эквивалентом понятия теории. Есть язык и правила, позволяющие отличать выражения (формулы) от простых наборов символов. Впрочем, исчисление

предикатов оснащено адекватной и полной аксиоматической системой для получения из малого числа аксиом и правил вывода всех истин языка. Исчисление предикатов редко употребляют как таковое. На практике чаще всего рассуждения строятся над довольно ограниченным множеством предикатных и функциональных констант. Их семантика лимитирована набором соответствующих аксиом. (В гл. 3 такое применение исчисления предикатов продемонстрировано для представления знаний.)

Сигнатурой называется набор предикатных и функциональных констант, каждая из которых обладает определенным числом мест. Не привлекая синтаксиса исчисления предикатов, сигнатура определяет некий язык, являющийся множеством термов и формул, которые разрешается строить.

Структурой относительно какого-то языка называется некая интерпретация этого языка. Каждой предикатной или функциональной константе сигнатуры сопоставляются предикат или функция подходящего типа на области интерпретации. Термы и формулы языка интерпретируются по правилам исчисления предикатов (§ 1.2.5).

Теорией относительно определенного языка называется некое множество формул этого языка. Последные называются *аксиомами*.

Теорема — это логическое следствие из аксиом. Термин «теория» иногда означает множество теорем (на практике такой разницей в толковании термина «теория» затруднений не вызывает). Структура, в которой истинны все аксиомы некой теории, называется *моделью* (для) этой теории.

Теория в нашем смысле (т. е. основанная на исчислении предикатов) часто именуется *теорией первого порядка*. Это название напоминает о том, что переменные здесь — только индивидные: нет ни функциональных, ни предикатных переменных. Однако таковые были бы очень полезны. Например, с их помощью очень просто вводилось бы определение равенства:

$$a = b =_{\text{def}} \forall P (P(a) \equiv P(b)).$$

Логики «высших порядков» обладают заведомо превосходящими выразительными возможностями. К сожалению, такое преимущество оплачивается отсутствием полной аксиоматизации для теорий высших порядков.

Итак, теория первого порядка является сужением исчисления предикатов на ту или иную заданную область. Аксиоматическая система любой конкретной теории получается присоединением специфических для этой теории аксиом к адекватной полной аксиоматической системе исчисления предикатов. Это не означает, что теорема исчисления предикатов обязательно остается теоремой в рассматриваемой теории.

Ведь язык этой теории, вообще говоря, более ограничен, чем язык исчисления предикатов. Очень часто он располагает лишь конечным числом различных предикатных и функциональных констант.

2.2.3. Польза теорий

К созданию и развитию формальной или неформальной теории могут приводить различные причины. В большинстве случаев теорию разрабатывают для отражения какой-то реальности (или того, что некоторые принимают за таковую), которая может быть более или менее конкретной или абстрактной (физика, химия, политика, астрология и т. д.). Тогда понятие истины предшествует теории и цель теории состоит в том (и ни в чем другом), чтобы вывести из множества истинных выражений, принятых за аксиомы, все касающиеся реальности истинные выражения.

Вполне естественно желание отделить их от ложных. Наоборот, неестественно тотчас вводить для этого некую теорию: может найтись средство попроще. Именно так обстоит дело с исчислением высказываний, для которого желаемое разделение осуществляется почти непосредственно (без привлечения таблиц истинности) — из рассмотрения самой реальности или по крайней мере ее модели (§ 1.1.4). Помимо этого прямого семантического средства можно использовать и алгоритмическое (например, алгоритм Куайна).

Аксиоматический подход будет применяться при невозможности или затруднительности использования прямого семантического и алгоритмического подходов. Даже в этом (увы, частом) случае цель аксиоматической теории — получить «отразить» семантику, т. е. изучаемую реальность.

Физика — пример неформальной теории, разработанной для лучшего понимания реальности. Физик прежде всего должен выработать подходящий язык и ввести понятия, например, массы, движения, силы и траектории. В принципе этот этап сравнительно прост, хотя уместность того или иного понятия не всегда легко осознать. Второй этап — сбор серии экспериментальных фактов и перевод их, насколько возможно, на язык физики. Третий этап самый деликатный. Он состоит в введении множества постулатов, из которых можно вывести все физические истины, и только их.

Многочисленные ловушки подстерегают физика. Первая состоит в несостоятельности или неадекватности: никакое противоречие, никакое ложное выражение не должно быть выводимым из множества предложенных постулатов. Отметим, что в неформальных теориях несостоятельность может быть как более, так и менее весомой. Высказывание о том, что Земля описывает круговую траекторию вокруг Солнца, несостоятельно (ибо она эллиптическая), но отчасти оно приемлемо (эксцентриситет этого эллипса мал).

Другая опасность менее серьезная, но которую труднее обойти, это неполнота: некоторые экспериментальные факты остаются необъяснимыми, но не входят в противоречие с существующими постулатами. Например, равенство инертной и гравитационной масс не объясняется классическими теориями, однако это не отрицает пользу последних.

Наконец, физик сталкивается с критерием качества. Теории Кеплера и Ньютона учитывают относительное движение звезд. Однако теория Ньютона лучше, так как она более общая: в ней постулаты Кеплера становятся просто теоремами.

Теперь рассмотрим одну формальную теорию, разработанную для отражения некой априорной

реальности (семантики). У нас «первой теорией» является само исчисление предикатов. Его язык максимален, а сигнатура содержит все функциональные и предикатные константы. Очевидно, это множество бесконечно. Счетна эта бесконечность или нет, знать не обязательно. Единственное ограничение состоит в том, чтобы была возможность эффективно решать, является ли данное выражение объектом языка или нет. Множество аксиом, наоборот, минимально, ибо пусто. Значит, ничего не должно добавляться к аксиоматической системе, приведенной в § 2.1.8.

Польза аксиоматического подхода спорна. Действительно, конечная цель теории состоит в различении истинного и ложного. В этом плане аксиоматический подход не лучше алгоритмического, в частности представленного методом резолюций (§ 1.2.14).

Несколько иной случай с арифметикой (теорией натуральных чисел). Со времен греческих математиков множество натуральных чисел изучалось аксиоматически и никакой другой подход никогда не смог утвердиться. Очевидно, лучший способ убедиться в правильности теории чисел — доказать ее, т. е. логически вывести из подходящего множества постулатов. Поиски такого адекватного полного множества — обширная тема.

Проблема существования «идеальной аксиоматизации» теории чисел, задаваемой множеством постулатов, вызывает следующие вопросы:

- Описывает ли множество постулатов натуральные числа, исключая любые другие структуры?
- Будут ли «эквивалентны» все описываемые этими постулатами структуры? Точнее, будет ли всякое истинное в данной структуре выражение истинным (в силу самого установленного факта) во всех других структурах?
- Кроме того, является ли любое выражение языка либо теоремой, либо ее отрицанием?
- Можно ли по крайней мере определить для каждого выражения языка, является ли оно теоремой, т. е. логическим следствием постулатов?

- Можно ли хотя бы гарантировать, что все теоремы действительно выражают истины теории чисел?

Далее мы увидим, что только на последний вопрос можно дать безоговорочно утвердительный ответ, что может сильно разочаровать читателя.

Аксиоматический подход к исчислению предикатов столь же ограничен, но это меньше разочаровывает. Действительно, это исчисление предназначено для описания не одной структуры, а всевозможных структур.

Иногда интересно изучать теорию саму по себе, не ставя первоочередной целью познание некой действительности. Например, пространственно-временные преобразования Лоренца были поначалу изучены по «эстетическим» причинам: они удовлетворяют (в отличие от классических преобразований Галилея) неким уравнениям. Не было обращения к действительности и в тот момент, когда Эйнштейн высказал предположение, что инвариантность этих уравнений должна быть законом природы. Это привело к созданию специальной теории относительности.

Подобный подход распространен и плодотворен также и в математике. Сражаясь с классической проблемой решения «в радикалах» полиномиальных уравнений, Галуа натолкнулся на весьма специальную структуру (группу перестановок корней многочлена). Исследование этой структуры позволило в итоге не только прояснить первоначальный вопрос, но и положить начало автономному изучению обширной категории структур, похожих на исходную. Это было рождением теории групп, оказавшейся полезной в самых различных областях математики. В следующем параграфе мы рассмотрим теорию такого же типа: ее предмет — довольно обширная категория структур, возникающих в разнообразных контекстах.

2.2.4. Теория частичного порядка

Здесь мы описываем одну специальную теорию, так называемую теорию частичного порядка. Делается это по двум причинам. С одной стороны, она позволяет

ярко проиллюстрировать различные общие понятия, встретившиеся в наших рассуждениях. С другой стороны, она (вместе с некоторыми обобщениями) играет очень важную роль в логике, информатике и многих областях математики.

Мы хотим определить теорию, модели которой — в точности упорядоченные множества. Единственное специфическое понятие — *отношение порядка*. Сигнатура содержит единственную двуместную предикатную константу \leq . По обыкновению префиксное обозначение $\leq(x, y)$ заменяют на инфиксное $x \leq y$. Язык здесь очень прост. Термы — только переменные. Атомов два вида: предикатные формы $x \leq y$ и равенства $x = y$.

Отношение порядка описывается тремя классическими аксиомами:

$$\forall x (x \leq x),$$

$$\forall x \forall y [(x \leq y \wedge y \leq x) \supset x = y],$$

$$\forall x \forall y \forall z [(x \leq y \wedge y \leq z) \supset x \leq z].$$

Теперь мы располагаем формальным средством, позволяющим распознавать, является множество K упорядоченным или нет. Все предельно просто, если K — структура¹⁾: предикатная константа \leq интерпретируется как функция из $K \times K$ в $\{И, Л\}$. Соответствующее этой функции бинарное отношение на K также обозначается символом \leq . Структура K является моделью, если три указанные аксиомы истинны в K (иными словами, если отношение \leq рефлексивно, антисимметрично и транзитивно).

Классическими примерами упорядоченных множеств являются числовые множества \mathbf{N} , \mathbf{Z} , \mathbf{Q} и \mathbf{R} ²⁾. Множество конечных последовательностей, составленных из конечного алфавита, упорядочивается лексикографически. Множества структурированных объектов упорядочены естественным образом: объект a предшествует объекту b , если a — есть конституента b . Множество теорем, выведенных в заданной аксио-

¹⁾ См. определение структуры в § 2.2.2. — Прим. перев.

²⁾ Соответственно множества натуральных, целых, рациональных и вещественных (действительных) чисел. — Прим. перев.

матической системе, тоже можно естественно упорядочить. Для этого фиксируем метод последовательного получения всех теорем и полагаем, что теорема A предшествует теореме B , если A получена раньше B .

2.2.5. Модели теории

Есть тесная связь между теорией T , языком L и возможными моделями для T . Справедлива следующая метатеорема.

Теорема 2.4. *Теоремы теории T это все те и только те выражения языка L , которые истинны во всех моделях для T .*

Доказательство. Результат мгновенно следует из определений понятий логического следствия (§ 1.1.6) и теоремы (§ 2.2.2). \square

Замечание. Другое определение теоремы дано в § 2.1.1: теорема есть формула, выведенная в фиксированной аксиоматической системе. По теореме 2.4 два этих определения эквивалентны при условии, что выбранная система является адекватной и полной аксиоматической системой исчисления предикатов, к которой добавлено множество аксиом конкретной теории.

Это очень мощный результат. Если есть множество K и совокупность отображений из K^n в K или в $\{И, Л\}$, то всегда можно построить теорию, для которой K будет моделью. Взаимно все множество формул T можно построить в рамках теории. Если T выполнимо, то всегда можно задать модель для T , например эрбранову (§ 1.2.9) или модель Хенкина. Общий метод построения последней изложен в [3].

Теория с бесконечным множеством аксиом E выполнима, если все конечные подмножества из E выполнимы. При этом существует модель, мощность которой не превосходит мощности E . Это следствие теорем компактности и Лёвенгейма — Сколема (§ 1.2.9).

При всем том существуют различные проблемы, связанные с понятием теории. Коснемся некоторых из них.

Прежде всего, если T — некоторая теория языка L и если A — формула, принадлежащая L , то (вообще

говоря) невозможно эффективно определить, является ли A теоремой в T . Точнее, если A в самом деле теорема, то ее можно доказать, например, пользуясь методом резолюций. Наоборот, если A не является теоремой, то, вообще говоря, мы не располагаем эффективным методом, чтобы это установить. Такого рода вопросы важны в информатике: эффективный метод для информатика тот, который можно запрограммировать.

Можно также задаться вопросом о том, насколько полно теория описывается своими моделями. Действительно, теория задает их общие свойства, но не специфические для частных видов моделей. Примером свойства, присущего одним упорядоченным множествам и не присущего другим, служит *тотальность*¹⁾ порядка:

$$\forall x \forall y [x \leq y \vee y \leq x].$$

С другой стороны, если все модели некоторой теории имеют в точности одни и те же свойства, то это еще не означает их идентичности. Действительно, некоторое свойство принимается во внимание только тогда, когда ему соответствует какая-нибудь формула языка. Например, бесконечное упорядоченное множество может быть счетным или несчетным, но теореме Лёвенгейма — Сколема никакая формула языка не позволяет различить эти два случая. Впрочем, имея одну модель, всегда можно построить из нее другую, меняя имена составляющих. Полученные таким способом варианты *изоморфны* исходной модели.

Эти рассуждения приводят к следующим определениям:

- Теория *разрешима*, если существует алгоритм, позволяющий за конечное число шагов решить, является ли некая (произвольная) формула A теоремой или отрицанием таковой или же ни тем, ни другим.
- Теория *полна*, если всякая формула языка есть теорема или отрицание теоремы.
- Теория *категорична*, если она допускает един-

¹⁾ Иное название — линейность. — Прим. ред.

ственную (с точностью до изоморфизма) модель.

Замечание. По теореме Лёвенгейма — Сколема конечные или счетные теории, допускающие несчетную модель, не являются категоричными. Отчасти по этой причине вводят следующее, более слабое, понятие. При заданной мощности α теория α -категорична, если она допускает не более одной модели мощности α .

Алгоритм решает некоторую *проблему*, т. е. отвечает на некоторый вопрос со многими (обычно двумя) альтернативами, допускающий несчетное множество конкретизаций. Например, вопрос, соответствующий решаемой алгоритмом Куайна проблеме, таков: «Является ли данная формула общезначимой, нейтральной или невыполнимой?». Множество конкретизаций — множество формул исчисления высказываний.

С каждой теорией органично связана проблема распознавания теорем. Конкретизации таких проблем — это формулы языка соответствующей теории. Вопрос ставится так: «Является ли данная формула теоремой?»

Некоторые проблемы не связаны прямо с теорией. Знаменитый пример — «десятая проблема» Гильберта. Множество конкретизаций — полиномиальные уравнения с целыми коэффициентами и многими неизвестными. Вопрос: «Имеет ли данное уравнение целочисленное решение?».

Подчеркнем тот факт, что проблема (в только что определенном смысле) имеет две существенные составляющие: вопрос со многими вариантами ответа и некоторое счетно-бесконечное множество конкретизаций. Это определение более ограниченное, чем широко распространенное понимание слова «проблема». Например, в этом смысле «великая теорема Ферма» проблемой не является, ибо имеет лишь одну конкретизацию.¹⁾

Заметим между тем, что большей частью «проблемы», затронутые в этой главе, действительно являются проблемами в техническом смысле этого термина.

¹⁾ Действительно, эта теорема выражается одной формулой, в которой все переменные связанные.

Впрочем, отметим, что подчас можно так переформулировать вопрос, что получится проблема. Эта тема освещена, например, в [36].

Понятия категоричности, полноты и разрешимости взаимозависимы. Справедлива следующая метатеорема.

Теорема 2.5. *Любая категоричная теория α -категорична для всех мощностей α . Любая (не имеющая конечной модели) выполнимая и α -категоричная хотя бы для одного бесконечного кардинального числа α теория полна. Любая полная теория разрешима.*

Доказательство. Первый пункт непосредственно следует из определений. Второй, принадлежащий Кантору, доказывается от противного. Пусть T — выполнимая неполная и допускающая только бесконечные модели теория. Пользуясь неполнотой, можно написать на языке теории T такую формулу A , что обе теории $T \cup \{A\}$ и $T \cup \{\neg A\}$ будут выполнимыми (модели обязательно бесконечные). По теореме Лёвенгейма — Сколема каждая из теорий допускает модель мощности α . Так как A истинна в одной из этих моделей, но не в другой, T не является α -категоричной. Третий пункт довольно прост. Пусть теория T полна: Для всякой формулы A , принадлежащей языку теории T , одно и только одно множество из $T \cup \{A\}$ и $T \cup \{\neg A\}$ невыполнимо. Метод резолюций позволяет определить, какое именно, и, следовательно, решить за конечное число шагов, принадлежит ли A данной теории. \square

Предметом *теории моделей* является в первую очередь построение моделей, соответствующих априорно заданной теории. Для этой цели были разработаны весьма общие методы построения [3], [13]. Данная тема выходит за рамки нашей книги.

Интересные теории никогда не бывают категоричными: это следствие теоремы Лёвенгейма — Сколема (§ 1.2.9). Создаваемые или используемые информатиками теории довольно редко оказываются полными. Действительно, не всегда полезно применять исчерпывающую теорию: часто предпочтительна рудиментарная, но зато удобная.

Наоборот, очень интересно знать, является ли (неполная) теория разрешимой. В какой-то мере это можно определить методами, используемыми для распознавания теорем. Например, для информатизации евклидовой геометрии предпочтительно использовать некую разрешающую процедуру (в духе аналитической геометрии). Напротив, для информатизации теории чисел, которая неразрешима [5], [3], [53], следует применять совсем иной метод, основанный на ИИ.

Замечание. Даже к разрешимым теориям иногда применяются методы ИИ, когда не известно никакого разрешающего алгоритма. В частности, именно так обстоит дело в области теории игр.

Можно захотеть пополнить неполную теорию за счет присоединения новых аксиом. Таковые должны быть достаточно сильны для достижения полноты, но довольно слабы, чтобы оставаться истинными в соответствующих моделях (по крайней мере в некоторых из них).

2.2.6. Алгоритмы и разрешимость

Понятие разрешимости существенно по двум причинам. С точки зрения логики, разрешимые теории достаточно многочисленны и поэтому их изучение представляет интерес. Это лишь в малой степени можно отнести к полным теориям, не говоря уже о категоричных. С точки зрения информатика, важно знать теоретические пределы программирования и понимать, что можно решить (вычислить) на ЭВМ. Важная ветвь логики, *теория рекурсии*, посвящена этому вопросу. Некоторые ее основные элементы будут здесь представлены.

Определение разрешимости из § 2.2.5 использует понятие алгоритма, никак формально не определенное. Перейдем к рассмотрению некоторых проблем, для которых разрешающие алгоритмы или процедуры уже упоминались.

- Является ли данный набор символов формулой исчисления высказываний (предикатов)?

- Является ли данная формула исчисления высказываний тавтологией? [Метод таблиц истинности, алгоритм Куайна.]
- Является ли данное утверждение евклидовой геометрии теоремой? [Классический аксиоматический и аналитический подходы.]
- Является ли данная формула исчисления предикатов общезначимой, нейтральной или невыполнимой? [Метод резолюций, натуральный вывод, аксиоматический подход.]
- Является ли данная формула исчисления предикатов хорновским дизъюнктом?
- Является ли данное множество хорновских дизъюнктов выполнимым?

Для начала заметим, что все эти проблемы касаются символьных объектов (цепочек символов) и что предлагаемые методы сводятся к манипулированию с символами, подчиненному точным правилам (одни цепочки получены из других). Разумеется, символам и цепочкам приписано некоторое значение. Впрочем, правила манипулирования вводятся независимо от этого значения. Однако, будучи введенными, они применяются «вслепую»: знание значений объектов манипуляции больше не требуется и даже не используется. Значения вновь станут полезны лишь при интерпретации полученных результатов.

Многие повседневные дела можно описать алгоритмически: воспользоваться лифтом, состряпать, идти (поставить одну ногу впереди другой, затем повторить...). У нас будет алгоритмически описываться разве что манипулирование с символами. С точки зрения информатика (ибо функционирование ЭВМ — слепое манипулирование с символами), алгоритмическая задача программируема.

На этой стадии не требуется уточнять, какой язык программирования используется. На деле большинство этих языков равны по выразительности. Разумеется, задача численного анализа будет, например, решаться на Фортране, а для задачи ИИ предпочтут Пролог, но лишь соображения удобства и эффективности повлияют на этот выбор.

Очевидно, формальная строгость охраняет свои права. Если предлагается разрешающий метод для данной теории и утверждается, что он является алгоритмом, следует иметь возможность записать его на точном, совершенно определенном языке.

Было создано огромное количество алгоритмических языков как для теоретических потребностей логиков, так и для практических нужд информатиков. Язык Пролог, очень полезное практическое средство, будет представлен в гл. 6. Два теоретических языка появятся в двух очередных параграфах.

Есть каноническое средство установления равной выразительности алгоритмических языков A и B . Сперва задают алгоритм перевода произвольной программы на языке A в эквивалентную на языке B , а затем алгоритм обратного перевода. Ничто не мешает писать эти алгоритмы на языках A или B .

Напомним, что две программы эквивалентны, если они дают одинаковые результаты для одних и тех же данных. В нашем контексте отсутствие результата («зацикливание») считается неким специальным результатом. Удобство и эффективность не учитываются.

Большинство рассматриваемых алгоритмов должны ответить на некоторый вопрос. Варианты ответа множественны (чаще всего дихотомичны). Именно так обстоит дело в примерах, приведенных выше. Собственно ни один из них не был представлен чисто формально. Впрочем, здесь это необязательно. Некоторые описаны точно (например, метод резолюций), а другие лишь упомянуты (как аналитический подход к геометрии). Любое описание алгоритма приемлемо, если переводимо в программу (на каком-то языке).

Важнее то, что среди затронутых методов одни всегда дают адекватный (положительный или отрицательный) ответ, а другие гарантируют лишь положительный (но не всегда отрицательный). Формально название *алгоритм* закрепим за первым типом. Методы второго типа суть *полуалгоритмы (процедуры)*. Метод резолюций — это процедура исчисления предикатов (§ 1.2.14), при условии, что принята стратегия адекватного выбора [14].

По определению теория разрешима, если она допускает разрешающий алгоритм. Она *полуразрешима* или *частично разрешима*, если допускает разрешающую процедуру. Разрешимость теории можно установить построением более или менее формально представленного алгоритма. А вот для установления неразрешимости теории необходим формальный язык: надо доказать, что никакой алгоритм, который можно написать на этом языке, не предоставляет разрешающего метода для этой теории.

Иные проблемы просты с виду, но на самом деле неразрешимы. Одна из известнейших — *проблема соответствия Поста*. При заданном конечном алфавите хотя бы из двух элементов (например, $\{0, 1\}$) *система Поста* — это конечная последовательность занумерованных пар слов. Например:

$\{1: (101, 0), 2: (1, 11)\}$.

Решение системы Поста — конечная последовательность номеров, где одно и то же слово получается пригонкой начал соответствующих пар, с одной стороны, и пригонкой концов этих же пар, с другой стороны. Последовательность $(2, 1, 2)$ — решение системы из примера. Получается слово 11011 . Для проблемы соответствия Поста множество конкретизаций — это множество систем Поста. Вопрос состоит в существовании решения. Неразрешимость этой проблемы о «головоломке» доказана, например, в [65].

Мы знаем, что исчисление высказываний разрешимо (но это немного выходит за рамки наших рассуждений, потому что речь фактически идет о теории «нулевого порядка», без квантификации) и исчисление предикатов полуразрешимо. Но имеет место и следующий результат, показывающий, насколько ограничено в своей основе то применение логики, которое может позволить себе информатик.

Теорема 2.6. *Исчисление предикатов неразрешимо.*

Доказать ее можно методом, называемым *редукцией*¹⁾: проблема соответствия Поста редуцируема к

¹⁾ Иное название — *метод сведения*. — Прим. ред.

проблеме разрешимости исчисления предикатов. Подробное доказательство см. в [65].

Замечание. *Исчисление монадических (унарных) предикатов*, в котором есть только одноместные предикаты (и нет никаких функций), разрешимо. Разрешающая процедура кратко описана в [72].

2.2.7. Алгоритмический язык Тьюринга

Тьюринг — автор простого и удобного алгоритмического языка, а также многих результатов из области разрешимости и вычислимости. Весьма полное изложение предложенного им языка можно найти в [73]. Мы же ограничимся здесь кратким очерком.

Неформально *машина Тьюринга* — очень примитивная программируемая вычислительная машина. Ее память состоит из бесконечной последовательности элементарных ячеек. Удобно полагать, что каждая ячейка помечена соответствующим целым числом. Она содержит символ из фиксированного конечного алфавита, содержащего n (не менее двух) элементов. Специальный элемент называется «пробел». Имеется головка, которая в каждый момент времени указывает одну из ячеек «ленты памяти». Третьей составляющей машины является программа.

Программа строится из конечного числа *состояний*, не принадлежащих упомянутому алфавиту символов. Единственное из них выделено как начальное. Нуль, одно или несколько состояний считаются заключительными. Каждому незаключительному состоянию соответствует *инструкция*, т. е. функция, сопоставляющая каждому символу алфавита тройку: символ алфавита, состояние и одну из констант $+$ или $-$. Четвертая и последняя составляющая машины Тьюринга — специальная ячейка (регистр программы) с определенным состоянием. Обычно машину Тьюринга изображают схемой — как на рис. 2.1.

Теперь опишем работу (манипуляцию символами), осуществляемую этой машиной. Фиксируется начальная конфигурация ленты памяти, в которой лишь конечное число ячеек содержит непробельные символы алфавита. (Это гарантирует конечность описания

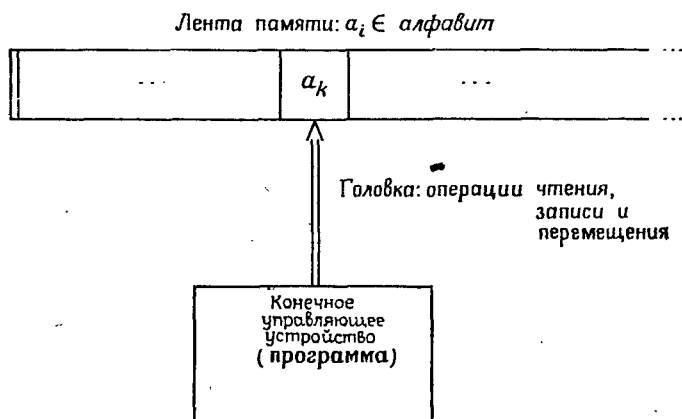


Рис. 2.1. Машина Тьюринга.

любой конфигурации ленты.) Головка указывает ячейку 0 на ленте. Регистр программы содержит начальное состояние. Манипуляция (действие, функционирование, работа) состоит из не более чем счетной последовательности шагов. Вот как выглядит описание шага.

Пусть i — номер указанной головкой ячейки, α — символ из этой ячейки и q — состояние из регистра программы. Пусть f_q — инструкция, соответствующая q , и (β, q', Δ) — тройка образа для символа α при f_q . Содержимое i -й ячейки заменяется на β , содержимым регистра становится q' , и головка теперь указывает $i\Delta$ 1-ю ячейку.

Если после выполнения шага в регистре появляется заключительное предписание, то работа машины прекращается (машина останавливается). Результат состоит из содержимого ленты, номера указанной головкой ячейки и содержимого программы в момент остановки.

Рассмотрим частный случай машины Тьюринга. Предположим, что для любого состояния q и любого элемента алфавита α $f_q(\alpha)$ имеет вид $(\alpha, q', +)$. Иначе говоря, лента памяти никогда не модифицируется и проходится лишь однажды, с 0-й ячейки в положительном направлении. Такая машина Тьюринга называется *конечным автоматом*.

Язык конечных автоматов используется в различных формах и в весьма разнообразных областях (как теоретических, так и практических). Выразительные возможности этого языка относительно слабы, но именно поэтому разрешимы в самой общей постановке следующие задачи: об эквивалентности двух (произвольных) конечных автоматов и об остановке (завершении работы) конечного автомата. В § 2.2.11 мы подробнее узнаем о неразрешимости этих важных проблем (эквивалентности и остановки) для машин Тьюринга. Более детально о машинах Тьюринга и конечных автоматах говорится в разд. 5.2 (в рамках иерархии Хомского для формальных грамматик).

Проиллюстрируем понятие машины Тьюринга простым примером. Алфавит $\{0, 1, b\}$, множество состояний $\{q_0, q_1, q_2, q_{\text{да}}, q_{\text{нет}}\}$. Начальное состояние q_0 , заключительные — $q_{\text{да}}$ и $q_{\text{нет}}$. До начала функционирования машины все ячейки, пронумерованные отрицательными числами, содержат символ b (пробел). Сплошная последовательность ячеек (массив) с номерами от 0 до n (любое натуральное число) содержит только нули и единицы. Наконец, все остальные ячейки содержат b . Инструкции программы описаны в таблице на рис. 2.2. Пример работы машины:

$q_0: \dots b \underline{1} 0 1 0 0 b \dots$

$q_0: \dots b \underline{1} 0 1 0 0 b \dots$

$q_0: \dots b 1 0 \underline{1} 0 0 b \dots$

$q_0: \dots b 1 0 1 \underline{0} 0 b \dots$

$q_0: \dots b 1 0 1 0 \underline{0} b \dots$

$q_0: \dots b 1 0 1 0 0 \underline{b} \dots$

$q_1: \dots b 1 0 1 0 \underline{0} b \dots$

$q_{\text{да}}: \dots b 1 0 1 0 \underline{b} b \dots$

Каждая строка описывает машину в определенный момент ее работы. В первом столбце указаны текущие состояния, в остальных столбцах — содержимое ленты. Подчеркиванием выделены указываемые головкой ячейки.

$f_q(\alpha)$	$\alpha = 0$	$\alpha = 1$	$\alpha = b$
$q = q_0$	$(q_0, 0, +)$	$(q_0, 1, +)$	$(q_1, b, -)$
$q = q_1$	$(q_{\text{да}}, b, -)$	$(q_2, b, -)$	$(q_{\text{нет}}, b, -)$
$q = q_2$	$(q_{\text{нет}}, b, -)$	$(q_{\text{нет}}, b, -)$	$(q_{\text{нет}}, b, -)$

Рис. 2.2. Инструкции программы.

Эта машина Тьюринга решает задачу выявления четности: если первоначально записанная на ленте последовательность нулей и единиц представляет четное двоичное число (т. е. с 0 в младшем разряде), то заключительным состоянием будет $q_{\text{да}}$, а иначе — $q_{\text{нет}}$. Можно доказать, что работа машины всегда завершается, т. е. на некотором шаге обязательно появляется заключительное состояние. Таким образом, мы имеем настоящий алгоритм, а не просто процедуру. Более того, при положительном ответе заключительная конфигурация ленты дает частное от деления на 2 числа, представленного начальной конфигурацией. Это упрощенная версия примера из [36]. Разнообразные варианты изложения соответствующего материала и различные примеры можно найти в [49], [65] и [73].

Замечание. Представленный выше язык не вполне отвечает обычному понятию языка программирования, ибо он описывает также «абстрактную машину» (воображаемую и с неограниченной памятью), выполняющую написанные на этом языке алгоритмы. Но это лишь видимость несоответствия: вполне возможно описать абстрактные машины, соответствующие, например, Фортрану и Прологу.

Язык машин Тьюринга кажется очень неразвитым по сравнению с обычными алгоритмическими языками (даже с языками Ассемблера примитивных компьютеров). Усовершенствования, осуществленные в обычных языках, безусловно, сделали их удобнее для применений, но ничуть не прибавили им выразительных возможностей. Ряд теоретических свойств языков программирования проясняется в рамках изрядно ободранного языка.

2.2.8. Алгоритмический язык Гёделя

Алгоритмический язык, который мы сейчас введем, принадлежит в основном Гёделю, еще одному пионеру изучения разрешимости. (Подробнее об этом см. в [5] и [96].)

Пусть A и B счетно-бесконечные множества. Займемся функциями из A в B . Они образуют несчетное множество. Некоторые из них являются *вычислимыми*, т. е. могут быть запрограммированы. Вычислимые функции образуют очень важное подмножество, к изучению которого мы приступаем. Оно не более чем счетное. Действительно, при использовании любого алгоритмического языка всякая программа состоит из конечной последовательности символов конечного или счетного алфавита. Отсюда следует, что множество программ счетно-бесконечно.

Удобно зафиксировать A и B . Подходящие кандидаты — множества натуральных чисел, целых чисел, конечных последовательностей над конечным алфавитом, конечных деревьев над конечным алфавитом, а также конечные декартовы произведения перечисленных множеств. Будем интересоваться главным образом функциями, отображающими множество $A = \mathbb{N}^n$, где n — произвольное натуральное число, в множество $B = \mathbb{N}$.

Идея Гёделя состояла в том, чтобы получить все вычислимые функции из существенно ограниченного класса базисных (исходных) функций с помощью элементарных алгоритмических средств. Базисные функции таковы:

- константа 0,
- одноместная функция следования, сопоставляющая натуральному числу x натуральное число $x + 1$,
- функция проецирования p_{ni} с параметрами (n, i) , где $1 \leq i \leq n$; функция p_{ni} сопоставляет каждой n -ке натуральных чисел ее i -й элемент ¹⁾.

¹⁾ Такую функцию часто обозначают через $I_i^{(n)}$ и называют *селекторной* или *функцией выбора аргумента*. — Прим. ред.

Гёдель ввел сначала два простых, но позволяющих получать нетривиальные вычислимые функции, механизма. Первый — обычная *подстановка* (композиция, или суперпозиция) функций. Это механизм явно алгоритмический: для вычисления, например, суперпозиции $u = f(g(x, y), h(z))$ достаточно вычислить последовательно $a = g(x, y)$, $b = h(z)$ и $u = f(a, b)$. Второй (несколько менее классический) механизм — *примитивная рекурсия*. Он действует так: из n -местной функции f и $(n + 2)$ -местной функции g строится $(n + 1)$ -местная функция h по следующей схеме:

$$h(x_1, \dots, x_n, 0) =_{\text{def}} f(x_1, \dots, x_n),$$

$$h(x_1, \dots, x_n, y + 1) =_{\text{def}} g(x_1, \dots, x_n, y, h(x_1, \dots, x_n, y)).$$

Такие определения нередки в математике. Классический пример — определение факториала:

$$0! =_{\text{def}} 1,$$

$$(x + 1)! =_{\text{def}} (x + 1) * x!.$$

Легко проверить, что это определение — конкретизация схемы примитивной рекурсии.

Две упомянутые схемы позволяют задать лишь всюду определенные отображения или тотальные функции (т. е. функции с областью определения \mathbb{N}^n). Может представить интерес рассмотрение также частичных функций, т. е. не всюду определенных. Такие функции порождаются с помощью третьего гёделева механизма. Схема *минимизации* позволяет получить n -местную функцию g из $(n + 1)$ -местной функции f :

$$g(x_1, \dots, x_n) =_{\text{def}} \mu y f(x_1, \dots, x_n, y).$$

Выражение $\mu x E$ означает наименьшее значение x , при котором E обращается в 0. Например, целый квадратный корень с недостатком из натурального числа x можно задать формулой

$$rc(x) =_{\text{def}} \mu y [x + 1 \div (y + 1)^2],$$

где \div означает вычитание в \mathbb{N} : выражение $a \div b$ есть $a - b$ при $a > b$, а иначе оно равно 0.

Очевидно, что даже если E вычислимо и всюду определено, но нигде не обращается в 0, то $\mu x E$ всюду не определено. В этом случае $\mu x E$ приписывают специальное значение, обозначаемое \perp или ∞ . Естественный путь вычисления $\mu x E$ состоит в подсчете значения E последовательно для $x = 0, 1, 2, \dots$ до тех пор, пока не найдется x , обращающее E в 0. Этот алгоритм «зациклится», если E нигде не обращается в 0.

2.2.9. Кодирование и тезис Чёрча

Языки Тьюринга и Гёделя априори трудно сравнить, ибо данные и результаты в них записываются в разных формализмах. Это препятствие легко преодолеть.

Рассмотрим машину Тьюринга с алфавитом $\{0, 1, b, \#\}$. Интерпретируем 0 и 1 как цифры двоичной системы счисления. Таким образом, конечные цепочки из 0 и 1 представляют натуральные числа. Символ b — пробел, а $\#$ — разделитель. Предполагаем, что в начальный момент отрицательная часть ленты памяти содержит только пробельные ячейки, а положительная содержит последовательность из n двоичных слов, отделенных друг от друга разделителем. За последним словом стоят две разделительные ячейки, что означает конец последовательности. Остальная лента пробельная. Понятно, что начальное содержимое ленты можно интерпретировать как элемент (x_1, \dots, x_n) области определения некой вычислимой функции f , значение которой $f(x_1, \dots, x_n)$ получается численной интерпретацией заключительного содержимого ленты.

В используемом формализме ничто не мешает численно интерпретировать вычисления, осуществляемые машиной Тьюринга. Алфавит, ввиду наличия хотя бы двух элементов, позволяет осуществлять *кодирование* информации любого типа. Таким образом, кодирование — это соответствие между множеством слов и множеством объектов. Подчеркнем, что это соответствие должно быть алгоритмическим в обе стороны.

Информация в ЭВМ закодирована с помощью 0 и 1. Прежде всего, это числовые и символьные данные. Роль «вычислительной машины» понимается широко: ЭВМ — средство обработки любой (а не только числовой) информации.

Это подсказывает способ доказательства вычислимости по Тьюрингу вычислимой по Гёделю функции. Сначала надо придумать машины Тьюринга для вычисления основных функций, используя, например, вышеупомянутое кодирование. Далее нужно выписать «алгоритмические рецепты» построения из машин для вычисления функций g и h , машины для вычисления f , получаемой из g и h по схеме примитивной рекурсии. Наконец, нужно выполнить подобное для подстановки и схемы минимизации. Доказательство технически нудновато, но не составляет непреодолимой теоретической проблемы. Заинтересованный читатель может обратиться, например, к книге [96], где приводится классическое изложение соответствующего материала, и к [5], где дается более современное изложение, в котором *регистровая машина* заменяет тьюрингову (что удобнее и ближе к реальным ЭВМ).

В ЭВМ кодируются не только данные, но и инструкции. Действительно, программа есть не что иное, как последовательность сообщений, которую можно подвергнуть двоичному кодированию. Ниже I^* означает код I .

Этот пункт узловой, ибо он подсказывает понятие *универсальной* машины Тьюринга. Такая машина принимает в качестве данных закодированное описание T^* машины T , наряду с другими данными x и в качестве результата выдает значение $f_T(x)$, где f_T — функция, вычисляемая на машине T . Мы допускаем существование универсальной машины Тьюринга U (что формально и конструктивно доказано в [73]). Для любой машины T имеем $f_U(T^*, x) = f_T(x)$.

Для доказательства вычислимости по Гёделю вычислимой по Тьюрингу функции достаточно доказать рекурсивность вычисляемой на универсальной машине U функции f_U . Сначала надо снабдить f_U «арифметикой». Для этого зададим функцию кодирования, сопоставляющую каждой конфигурации s машины U число s^* . Это соответствие, а также обратное ему должны быть алгоритмическими. Зададим частичную функцию φ из $\mathbf{N} \times \mathbf{N}$ в \mathbf{N} : $\varphi(x, y) = z$ тогда и только тогда, когда существуют такие конфигурации s и d , что $s^* = x$, $d^* = z$ и конфигурация d получается из s за

у шагов. Таким образом, φ описывает поведение U . Можно доказать, что φ рекурсивна. Можно также определить f_U , исходя из φ , и доказать рекурсивность f_U . Напомним, что функция f_U частичная. Подробности см., например, в [73].

Понятие универсальной машины может дезориентировать. Однако оно близко к реальности информатики. Возьмем обычный язык программирования, например Фортран. Оснащенная им ЭВМ допускает «универсальную машину Фортрана», состоящую из компилятора, загрузчика и супервизора. Она принимает программу на Фортране с данными и выдает требуемый результат.

Напомним, что имеется простое концептуально простое средство проверки равной выразительности двух языков программирования: достаточно написать «кросс-компиляторы», т. е. программы перевода с одного языка на другой. Не будем описывать эти программы. Лишь отметим, что для некоторых языков программирования (таких как Лисп) соответствие программы ее коду довольно очевидно. Возьмем, например, следующий текст:

```
(DEFUN FACT(N)
  (COND((ZEROP N) 1)
        (T (TIMES N (FACT(SUB1 N)))))).
```

Его можно рассматривать как программу вычисления факториала и как «символьное выражение», т. е. бинарное дерево, листьями которого являются атомы. Такие объекты — обычные данные программы на Лиспе. Следовательно, здесь почти полное тождество программы и ее представления в виде данных, возможно, полученных другой программой.

Изобретено великое множество алгоритмических языков. Можно установить, что у них одинаковые выразительные возможности. Это обстоятельство приводит к тезису Чёрча: вычислимы в интуитивном смысле функции вычислимы по Тьюрингу (или Гёделю). Очевидно, он недоказуем, ибо касается неформального понятия. (См. по этому поводу [5], [23] и [73].)

2.2.10. Класс вычислимых функций

Согласно тезису Чёрча бесполезно стремиться к расширению множества вычислимых функций. Напротив, желательно поточнее его очертить и поискать методы, позволяющие выяснить, вычислима ли функция, а в случае, когда она вычислима, указывающие, как ее вычислить. Известно, что множество вычислимых функций содержит частичные функции, ибо существуют программы, функционирование которых завершается не для всех данных. Заманчиво располагать языком, программирующим тотальные (всюду определенные) вычислимые функции, и только их.

Увы, такого языка нет. Это легко доказать от противного. Допустим, язык L позволяет записывать все тотальные вычислимые функции, и только их. Множество программ счетно. Тем лучше, можно приписать каждой из них кодовый номер и задать алгоритмический способ перехода от программы к ее номеру и обратно. Итак, пусть P_x — программа с номером x и f_x — функция, которую она вычисляет. Функция U , сопоставляющая паре (x, y) значение $f_x(y)$, очевидно, вычислима: сначала используется указанное выше алгоритмическое соответствие между x и P_x , а затем P_x применяется к y . Значит, вычислима функция $g(x) = U(x, x)$ и функция $h(x) = g(x) + 1$. Функция h тотальна, следовательно, она программируема в L . Пусть ее номер n . Имеем: $h(x) = f_n(x)$ для всех x , в частности, $h(n) = f_n(n)$. С другой стороны, из определения функции h вытекает, что $h(n) = U(n, n) + 1 = f_n(n) + 1$. Пришли к противоречию. Заметим, что функция U соответствует «универсальной машине» из § 2.2.8. Вывод таков: всякий язык, который является достаточно мощным, в том смысле, что допускает универсальную машину, описываемую самим этим языком, неизбежно допускает описание частичных функций. Вычислимая функция U непременно частичная.

Если язык Гёделя лишить схемы минимизации, тогда все выражимые в нем функции будут тотальными. Эти функции называются *примитивно рекурсивными*. Однако существуют тотальные вычислимые

функции, не являющиеся примитивно рекурсивными. Приведем пример такой функции. Сначала рассмотрим несколько важных функций из теории чисел:

$$\text{succ}(x) =_{\text{def}} x + 1;$$

$$\text{plus}(x, 0) =_{\text{def}} x.$$

$$\text{plus}(x, y + 1) =_{\text{def}} \text{succ}(\text{plus}(x, y));$$

$$\text{times}(x, 0) =_{\text{def}} 0.$$

$$\text{times}(x, y + 1) =_{\text{def}} \text{plus}(x, \text{times}(x, y)).$$

Очевидно, они примитивно рекурсивны. Построим далее последовательность функций:

$$\text{exp}_0 =_{\text{def}} \text{times},$$

$$\text{exp}_{n+1}(x, 0) =_{\text{def}} 1,$$

$$\text{exp}_{n+1}(x, y + 1) =_{\text{def}} \text{exp}_n(x, \text{exp}_{n+1}(x, y)),$$

Все эти функции примитивно рекурсивны. Функция exp_1 — классическая экспонента. Построение функции exp_n требует $(n + 2)$ применений схемы примитивной рекурсии. Легко проверить, что функция

$$f(x) =_{\text{def}} \text{exp}_x(x, x)$$

не является примитивно рекурсивной, хотя она вычислима и всюду определена. Действительно, она растет быстрее любой примитивно рекурсивной функции.

Немного о терминологии. Вычислимые функции называют также *рекурсивными*¹⁾ Подмножество в \mathbb{N} *рекурсивно*²⁾ тогда и только тогда, когда его характеристическая функция рекурсивна. (Заметим, что характеристическая функция обязательно является тотальной.) Пусть f — рекурсивная функция. Соответствующее f *перечисление* есть последовательность $(f(n))$, $n \in \mathbb{N}$. Множество E *рекурсивно перечислимо*, если существует такая функция f , что $E = \{f(n) \mid n \in \mathbb{N}\}$.

¹⁾ Или *частично рекурсивными*. — Прим. ред.

²⁾ Или *разрешимо*. — Прим. ред.

Ясно, что множество E , состоящее из натуральных чисел, рекурсивно тогда и только тогда, когда существует алгоритм решения проблемы принадлежности к E . Если принять за определение этот критерий, то он показывает приложимость понятия рекурсивного множества не только к множествам натуральных чисел. Это уже использовалось в § 1.1.3, где формулы исчисления высказываний составили рекурсивное подмножество во множестве наборов символов. Тавтологии образуют рекурсивное подмножество формул исчисления высказываний (§ 1.1.5).

Понятие рекурсивно перечислимого множества обобщается аналогично. Множество E рекурсивно перечислимо, если имеется алгоритм перечисления его элементов.¹⁾ В исчислении предикатов и вообще в любой теории первого порядка теоремы всегда образуют рекурсивно перечислимое подмножество формул. Однако оно рекурсивно лишь в случае разрешимости теории.

Пусть A — рекурсивное множество, построенное с помощью конечного числа правил образования. (К этому типу принадлежат: множество всех натуральных чисел, языки исчисления высказываний и всех теорий первого порядка.) Ясно, что любое рекурсивное подмножество A рекурсивно перечислимо. Для установления разрешимости некоторых теорий используется следующее частичное обращение.

Теорема 2.7. *Если A — рекурсивное множество, B — рекурсивно перечислимое подмножество из A , то B рекурсивно тогда и только тогда, когда $A \setminus B$ рекурсивно перечислимо.*

Доказательство. Если B рекурсивно, то $A \setminus B$ также рекурсивно и, следовательно, рекурсивно перечислимо. Обратно: если B и $A \setminus B$ рекурсивно перечислимы, то они рекурсивны. Действительно, чтобы решить, принадлежит ли B элемент x из A , достаточно перечислять параллельно B и $A \setminus B$. Рано или поздно элемент x появится в одном из перечислений, что и решит вопрос. \square

¹⁾ При этом повторения не исключаются. — *Прим. ред.*

2.2.11. Проблема остановки

Основополагающей теоретической и практической проблемой для информатика является *проблема остановки*. Она допускает множество вариантов. Вот три из них:

- | | |
|-----------------------------|---|
| 1. Множество конкретизаций: | множество программ P . |
| Поставленный вопрос: | завершится ли выполнение программы P при отсутствии данных? |
| 2. Множество конкретизаций: | множество программ P . |
| Поставленный вопрос: | завершится ли выполнение программы P для данных P^* (код P)? |
| 3. Множество конкретизаций: | множество пар (P, X) , где P — программа и X — данные. |
| Поставленный вопрос: | завершится ли выполнение программы P для данных X ? |

Вариант 3 самый общий, но два других также интересны. Для варианта 2 интересен случай, когда P — компилятор (или интерпретатор), а используемый для кодирования программы P язык является входным для P .

Неразрешимость проблемы остановки — прямое следствие того, что вычисляемая универсальной машиной U функция обязательно частичная. Точнее, в § 2.2.10 показано существование программы P с кодом $P^* = n$, таким, что $U(n, n)$ не определено. Это правило для вариантов 2 и 3. Наконец, если бы вариант 1 был разрешимым, то вариант 2 — тоже. Чтобы выяснить, завершается ли выполнение программы P в применении к P^* , достаточно сконструировать программу Q_P , которая сперва применяет P к

P^* , а затем принимает (бесполезные) данные D . Видно, что выполнение P в применении к P^* завершается тогда и только тогда, когда выполнение Q_P завершается в применении к D .

Существуют и другие неразрешимые проблемы относительно класса программ (вычислимых функций). Например, невозможно в полной общности решить, заикливается ли программа P или еще, эквивалентна ли P программе Q . Если бы такое решение было возможно, то (в противоречии с предыдущим) решалась бы и проблема остановки.

3. Представление знаний и рассуждений

3.1. Логическое представление

3.1.1. Введение

Представление — это действие, делающее некоторое понятие воспринимаемым посредством фигуры, записи, языка или формализма. Теория *знаний* изучает связи между субъектом (изучающим) и объектом. Знание (в объективном смысле) — то, что известно (то, что знаем после изучения).

Представление знаний — фõрмализация истинных убеждений посредством фигур, записей или языков. Нас особенно интересуют формализации, воспринимаемые (распознаваемые) ЭВМ. Возникает вопрос о представлении знаний в памяти ЭВМ, т. е. о создании языков и формализмов представления знаний. Они преобразуют наглядное представление (созданное посредством речи, изображением, естественным языком, вроде французского или английского, формальным языком, вроде алгебры или логики, рассуждениями и т. д.) в пригодное для ввода и обработки в ЭВМ. Результат формализации должен быть множеством инструкций, составляющих часть языка программирования.

Представлению знаний присущ пассивный аспект: книга, таблица, заполненная информацией память. В ИИ подчеркивается активный аспект представления: *знать* должно стать активной операцией, позволяющей не только запоминать, но и извлекать воспринятые (приобретенные, усвоенные) знания ¹⁾ для рассуждений на их основе. Следовательно, истоки представления знаний — в науке о познании ²⁾, а его конечная цель — программные средства информатики.

¹⁾ Точнее, знания, которые запомнились. — Прим. ред.

²⁾ Эпистемология, гносеология. — Прим. перев.

Во многих случаях подлежащие представлению знания относятся к довольно ограниченной области, например:

- описание состояния морфлота,
- описание ситуаций в игре (например, расположения фигур в шахматах),
- описание размещения персонала предприятия,
- описание пейзажа.

Для характеристики некой области говорят об «области рассуждений» или «области экспертизы». Численная формализация таких описаний в общем малоэффективна. Напротив, использование символического языка, такого, как язык математической логики, позволяет формулировать описания в форме, одновременно близкой и к обычному языку, и к языку программирования. Впрочем, математическая логика позволяет рассуждать, базируясь на приобретенных знаниях: логические выводы действительно являются активными операциями получения новых знаний из усвоенных.

В силу всех этих причин математическая логика лежит в основе различных представлений в ИИ. Данный раздел посвящен представлению знаний с помощью логики предикатов. Для этого используются результаты разд. 1.2. Логическое представление служит также отправной точкой для других представлений (таких как «сетевые» и «объектные»), используемых в ИИ.

3.1.2. Синтаксис логики предикатов

Язык логики предикатов задается синтаксисом (§ 1.2.3). Для представления знаний базисные синтаксические категории языка изображаются такими символами, которые несут достаточно четкую информацию и дают довольно ясную картину об области рассуждений (экспертизы).

Логика предикатов, называемая также логикой первого порядка, допускает четыре типа выражений (§ 1.2.2):

- *Константы.* Они служат именами индивидуумов (в отличие от имен *совокупностей*): объектов, людей или событий. Константы представляются символами вроде *Жак_2* (добавление 2 к слову *Жак* указывает на вполне определенного человека среди людей с таким именем), *Книга_22*, *Посылка_8*.
- *Переменные.* Обозначают имена совокупностей, таких как человек, книга, посылка, событие. Символ *Книга_22* представляет вполне определенный экземпляр, а символ *книга* указывает либо множество «всех книг», либо «понятие книги». Символами *x*, *y*, *z* представлены имена совокупностей (определенных множеств или понятий).
- *Предикатные имена.* Они задают правила соединения констант и переменных, например правила грамматики, процедуры, математические операции. Для предикатных имен используются символы наподобие следующих: *Фраза*, *Посылать*, *Писать*, *Плюс*, *Разделить*. Предикатное имя иначе называется *предикатной константой*.
- *Функциональные имена* представляют такие же правила, как и предикаты. Чтобы не спутать с предикатными именами, функциональные имена пишут одними строчными буквами: *фраза*, *посылать*, *писать*, *плюс*, *разделить*. Их называют также *функциональными константами*.

Символы, которые применяются для представления констант, переменных, предикатов и функций, не являются «словами русского¹⁾ языка». Они суть символы некоторого представления — слова «объектного языка» (в нашем случае «языка предикатов»).

Представление должно исключать всякую двусмысленность языка. Поэтому имена индивидуумов содержат цифры, приписываемые к именам совокупностей. *Жак_1* и *Жак_2* представляют двух людей с одинаковыми именами. Эти представления суть

¹⁾ В оригинале — французского. — *Прим. ред.*

конкретизации имени совокупности «Жак». *Предикат* — это предикатное имя вместе с подходящим числом термов. Предикат называют также *предикатной формой* (§ 1.2.3).

3.1.3. Примеры

Проиллюстрируем синтаксис логики предикатов, сопоставляя нескольким русским фразам их переводы на язык логического формализма.

- По-русски: Жак посылает книгу Мари,
Логически: *Посылка* (*Жак-2, Мари-4, Книга-22*).
- По-русски: Каждый человек прогуливается,
Логически: $\forall x (\text{Человек}(x) \supset \text{Прогуливается}(x))$.
- По-русски: Некоторые люди прогуливаются,
Логически: $\exists x (\text{Человек}(x) \wedge \text{Прогуливается}(x))$.

(Сравнивая два последних примера, видим, что замена прилагательного «каждый» на «некоторые» влечет при переводе не только замену квантора \forall на \exists , но и замену связки \supset на \wedge . Это иллюстрирует тот факт, что перевод фразы естественного языка на логический, вообще говоря, не является трафаретной операцией.)

- По-русски: Ни один человек не прогуливается,
Логически: $\neg (\exists x (\text{Человек}(x) \wedge \text{Прогуливается}(x)))$.

3.1.4. Преобразование унарных предикатов в бинарные

Унарные и *бинарные предикаты* имеют соответственно один и два аргумента. Фраза «Сократ есть человек» представима унарным предикатом *Человек*(*Сократ*), а «Жак любит Мари» — бинарным *Любит*(*Жак-2, Мари-4*). Бинарные предикаты играют особую роль в представлении знаний.

Любой унарный предикат можно преобразовать в бинарный следующим образом. Унарный предикат состоит из предикатного имени и значения своего единственного аргумента. Следовательно, его формат такой: *Предикатное_имя* (*значение_аргумента*). Значение аргумента есть конкретизация предикатного

имени или переменной. Например, *Человек(Сократ)* указывает, что имя собственное «Сократ» — конкретизация имени совокупности «человек», а *Человек(x)* указывает, что x — человек (не что-либо иное). Точно так же *Женщина(Мари_4)* означает, что «Мари» — «женщина». Введем бинарный предикат *Конкр* (значение_1, значение_2) и придадим ему следующий смысл: значение первого аргумента — имя индивидуума или элемент вполне определенного типа, значение второго аргумента — имя совокупности этого типа. Фразу «Сократ — человек» можно представить бинарным предикатом *Конкр(Сократ, человек)*.

Таким образом, введенный бинарный предикат эквивалентен исходному унарному в том смысле, что они оба представляют одну и ту же фразу. Преобразование следующее:

Унарный предикат	→ Бинарный предикат,
<i>Имя_совокупности</i>	→ <i>Конкр(имя_индивиду-</i>
<i>(имя_индивидуума)</i>	<i>ума, имя_совокуп-</i>
	<i>ности).</i>

Таким образом, предикатное имя *Конкр* означает: «является элементом некоторого типа». Иногда этот предикат обозначают так: *Есть_некий*.

3.1.5. Примеры

Преобразуем унарные предикаты из примеров § 3.1.3 в эквивалентные бинарные. Во втором примере имеются несколько эквивалентных версий.

- По-русски: Каждый человек прогуливается,
Логически: $\forall x (\text{Конкр}(x, \text{человек}) \supset \text{Конкр}(x, \text{прогуливается}))$.
- По-русски: Ни один человек не прогуливается,
Логически-1: $\neg (\exists x (\text{Конкр}(x, \text{человек}) \wedge \text{Конкр}(x, \text{прогуливается})))$,
Логически-2: $\neg (\exists x \exists y (\text{Конкр}(x, \text{человек}) \wedge \text{Конкр}(y, \text{прогуливается}) \wedge \text{Субъект}(y, x)))$,
Логически-3: $\forall x \forall y (\neg \text{Конкр}(x, \text{человек}) \vee \neg \text{Конкр}(y, \text{прогуливается}) \vee \neg \text{Субъект}(y, x))$.

Предикат *Субъект* (y, x) означает: переменная x есть субъект события y ; в данном случае «человек» — субъект события «прогуливается». Три логических представления эквивалентны. Они представляют синонимы одного и того же утверждения: «Неправда, что существует прогуливающийся человек»; «Неправда, что существуют x и y , где x — человек, y — прогуливается и x — субъект события y », «Для всех x и всех y либо x не человек, либо y не означает прогуливаться, либо же x не субъект события y ».

3.1.6. Преобразование t -арных предикатов в произведение бинарных

Фразу «Жак посылает книгу Мари» нетрудно представить тернарным (т. е. с тремя аргументами) предикатом:

Посылка (*Жак*_2, *Мари*_4, *Книга*_22).

Определив новые предикаты, можно представить эту фразу произведением (конъюнкцией) бинарных предикатов:

Отправитель (*Посылка*, *Жак*_2) \wedge
Получатель (*Посылка*, *Мари*_4) \wedge
Объект (*Посылка*, *Книга*_22).

Данная логическая формула читается так: «отправитель посылки — Жак, получатель посылки — Мари и объект посылки — книга».

Этот пример подсказывает правило преобразования t -арных предикатов (с t аргументами) в эквивалентное произведение бинарных предикатов. Всякий t -арный предикат состоит из предикатного имени и t значений аргументов:

Предикатное_имя (*значение*_1, *значение*_2, ...
..., *значение*_t).

В таких обозначениях подразумеваемые предикатом функциональные отношения выражены порядком аргументов. Например, предикат для описания посылки некоторого объекта отправителем получателю может иметь следующий формат:

Посылка (*отправитель*, *получатель*, *объект*).

Функция «отправитель посылки» учитывается первым аргументом, функция «получатель посылки» — вторым, функция «объект посылки» — третьим. Первоначально выбранный порядок аргументов для соответствия «функция — аргумент» может быть произвольным, но должен сохраняться неизменным. Функциональную организацию m -арного предиката можно представить так:

Предикатное_имя (*функция_1*, *функция_2*, ..., *функция_m*).

При записи m -арного предиката через бинарные предикаты используется специальное соглашение, позволяющее сохранить и явно указать соответствующие функциональные отношения. Каждая функция становится именем бинарного предиката, первый аргумент которого является именем исходного m -арного предиката, а второй — значением относящегося к этой функции аргумента. Функциональная организация исходного m -арного предиката и значения его аргументов представляются произведением бинарных предикатов:

Функция_1 (*предикатное_имя*, *значение_1*) \wedge
 ...
 \wedge *Функция_m* (*предикатное_имя*, *значение_m*).

Таким образом, имеем следующее преобразование:

Предикатное_имя (*значение_1*, ..., *значение_m*) \rightarrow
 \wedge_j *Функция_j* (*предикатное_имя*, *значение_j*).

Запись в терминах бинарных предикатов содержит в явном виде имена функциональных отношений, которые могли бы быть неявно представлены порядком следования аргументов при записи в терминах m -арных предикатов. Пара \langle *функция_j*, *значение_j* \rangle называется *слотом* (по-английски *slot*). Слот состоит из своих *имени* (которое именует функциональное отношение) и *ключа* (который представляет значение функционального отношения, то есть значение j -го аргумента исходного m -арного предиката). Соответствующие английские термины *slot*, *slot-name*, *slot-value*;

бинарный предикат вида *Функция_j (предикатное_имя, значение_j)* называется *slot-assertion notation*.

3.1.7. Явное представление ссылок

При представлении знаний в логической форме важно исключить двусмысленности, которые вообще имеются в любом неформализованном представлении. Именно поэтому мы дали имена каждому индивиду, появляющемуся в логической формуле. Символы объектного языка, такие как *Жак_2*, *Мари_4* и *Книга_22*, введены для того, чтобы избежать двусмысленности ссылок на вполне определенных людей и книгу.

Впрочем, можно постулировать, что фраза «Жак посылает книгу Мари» указывает определенное «почтовое событие» (или «действие отправления»), на которое желательно иметь возможность ссылаться в дальнейшем. Следовательно, придется дать ему имя индивидуума *Посылка_8* и указать, что оно — часть множества событий с именем совокупности посылки. Перевод фразы на бинарные предикаты выглядит так:

Отправитель (Посылка_8, Жак_2) ∧
Получатель (Посылка_8, Мари_4) ∧
Объект (Посылка_8, Книга_22) ∧
Элем (Посылка_8, посылки).

Предикатное имя *Элем* означает «есть элемент такого-то множества» и является логическим переводом символа принадлежности \in . Предикатное имя *Элем* не следует путать с предикатным именем *Конкр* из § 3.1.5. Принадлежность к типу — не то, что к множеству: *Элем (Посылка_8, посылки)* принадлежность к множеству «почтовых посылок», тогда как *Конкр (Посылка_8, посылка)* — к абстрактному типу «почтовая посылка». К этому важному различию мы еще вернемся.

В этом параграфе высвечено явное построение множества «ссылок» некоторой фразы.

3.1.8. Представление функциями

Отношения между значением *Посылка_8* и аргументами тернарного предиката *Посылка* (§ 3.1.6) можно выразить также функциями на множестве почтовых посылок (одной переменной). *Функция* называется также *функциональной формой* (§ 1.2.3). Фраза «Жак посылает книгу Мари» выражается в терминах функций следующей формой:

Равно (*отправитель* (*Посылка_8*), *Жак_2*) \wedge
Равно (*получатель* (*Посылка_8*), *Мари_4*) \wedge
Равно (*объект* (*Посылка_8*), *Книга_22*) \wedge
Элем (*Посылка_8*, *посылки*).

Предикат *Равно* представляет отношение равенства. Эти выражения используют определенные на множестве «посылок» функции, значения которых представляют конкретизации, касающиеся *Посылки_8*.

3.1.9. Примеры

Рассмотрим фразы той же синтаксической формы, что и «Жак посылает книгу Мари», но с кванторами.

- По-русски: Жак посылает что-то каждому (всем одно и то же),
 Логически: $\exists y \forall x \text{Посылка}(\text{Жак}_2, x, y)$.
- По-русски: Жак посылает что-то каждому (но не обязательно всем одно и то же),
 Логически-1: $\forall x \exists y \text{Посылка}(\text{Жак}_2, x, y)$,
 Логически-2: $\forall x \exists y \exists \text{посылка}(\text{Отправитель}(\text{посылка}, \text{Жак}_2) \wedge \text{Получатель}(\text{посылка}, x) \wedge \text{Объект}(\text{посылка}, y))$,
 Логически-3: $\forall x \exists y \exists z (\text{Отправитель } z(\text{Жак}_2) \wedge \text{Получатель}(z, x) \wedge \text{Объект}(z, y) \wedge \text{Элем}(z, \text{посылки}))$.

Три представления фразы «Жак посылает что-то каждому» соответствуют трем представлениям фразы «Жак посылает книгу Мари» из §§ 3.1.6—3.1.7.

3.1.10. Семантика логики предикатов

Стратегия определения семантических значений компонент и формул логики предикатов базируется на понятии интерпретации логической формулы, введенном в §§ 1.1.4 и 1.2.5. Прежде всего задаем семантическое значение для каждого базисного выражения. Затем вводим семантические правила вычисления семантических значений сложных логических формул по известным семантическим значениям компонент. Иначе говоря, приписываем семантические значения все более и более крупным составляющим логической формулы, так что в конце концов семантическое значение будет приписано и всей формуле (§ 1.1.4). Этот процесс называется композиционным методом.

Предположим для определенности, что универсум рассуждения содержит лишь имена трех людей: «Жак Дюпон», «Мари Дюран», «Джордж Буль» — и название одной книги: «Законы мышления». С каждым именем индивидуума, используемым в логических формулах, можно следующим образом связать одно из перечисленных имен, которое станет семантическим значением:

Имена индивидуумов	Семантические значения
<i>Жак_2</i>	Жак Дюпон
<i>Мари_4</i>	Мари Дюран
<i>Джордж_5</i>	Джордж Буль
<i>Книга_22</i>	Законы мышления

Левый столбец составлен из лингвистических сущностей, т. е. из лексических компонент особой синтаксической категории некоего языка. Правый — из сущностей реального мира.

Подчеркнем, что следует приписывать уникальное семантическое значение каждому базисному выражению. Это устраняет лексические двусмысленности реального мира. Напротив, некоторые объекты реального мира могут вообще не получать индивидуального имени из языка или получать несколько имен. Итак, мы хотим определить функцию (в математиче-

ском смысле), отображающую имена объектного языка на сущности реального мира. Сами эти сущности выражены на так называемом *метаязыке*, роль которого играет здесь русский язык:

Фундаментальным понятием семантики является понятие *истины реального мира*. Более общее понятие — *истина в модели*. Здесь моделью является реальный мир (§§ 2.2.2—5). Состояние реального мира позволяет приписать семантические значения «истинно» или «ложно» предикатам и функциям.

Например:

Предикат	Семантическое значение
<i>Посылка</i> (Жак_2, Мари_4, Книга_22)	истинно
<i>Посылка</i> (Мари_4, Жак_2, Книга_22)	ложно
<i>Написанное</i> (Джордж_5, Книга_22)	истинно
<i>Написанное</i> (Жак_2, Книга_22)	ложно

Композиционный метод гарантирует, что семантическое значение сложного выражения всегда является функцией его синтаксических составляющих и способа их комбинирования. Если семантические значения формул F и G известны, то можно определить семантические значения формул $\neg F$, $F \wedge G$, $F \vee G$, $F \supset G$ и $F \equiv G$ с помощью таблиц истинности логики высказываний (§ 1.1.4).

Например:

Формула	Семантическое значение
<i>Посылка</i> (Жак_2, Мари_4, Книга_22) \wedge \neg <i>Написанное</i> (Жак_2, Книга_22)	истинно
<i>Посылка</i> (Жак_2, Мари_4, Книга_22) \wedge <i>Написанное</i> (Джордж_5, Книга_22)	истинно

Основной задачей представления знаний является перевод неформальных выражений или описаний метаязыка в фразы объектного языка. Приведенные выше примеры показывают, что выбор предикатов, числа их аргументов, констант и переменных в значительной мере отдан во власть аналитика. Исчисление

предикатов не предоставляет никаких возможностей обоснования этого выбора.

3.1.11. Модальная логика предикатов

В обыденном языке часто говорят о допустимости чего-либо, о гипотетических событиях, целях, которые можно пытаться достигнуть, догадках о будущем. Большая часть фраз языка может быть то истинной, то ложной в зависимости от обстоятельств, текущего момента, точки зрения каждого из нас. В естественных языках модальности «возможный», «необходимый» и «допустимый» выражаются вспомогательными глаголами, такими как «должен» и «могу»¹⁾.

Возможность и необходимость называются *алетическими модальностями* или *модальностями возможности*. Так же, как кванторы «для всех» и «существует» вводились в синтаксисе логики первого порядка, можно построить формальный язык, используя пару понятий «возможно»/«необходимо» как кванторы, действующие на формулы. Логическая система, базирующаяся на операторах «возможно, что] и «необходимо, чтобы», называется *логикой возможного*, или *алетической логикой*.

Для обозначения модальности «необходимо» используется символ \Box . Формула $\Box F$ читается «необходимо, чтобы F » или « F необходимо». Двойственный \Box оператор обозначается \Diamond . Формула $\Diamond F$ читается «возможно, что F » или « F возможно». Один из этих операторов принимается за основной, а другой определяется через него и отрицание (эквивалентность $\Box F \equiv \neg \Diamond \neg F$ можно установить, применяя доводы, подобные тем, которые можно использовать при доказательстве соотношения $\forall x F \equiv \neg \exists x \neg F$).

В обыденном языке употребляются и другие модальные формы, которые важно перенести в логику. *Деонтическая логика* вводит модальности «разрешено» «обязательно», реализующие модальные языковые конструкции «разрешается» «надо, чтобы». *Эпи-*

¹⁾ Далее следует фраза, которой авторы поясняют применение модальности во французском языке. — *Прим. перев.*

стемическая¹⁾ логика, или логика знания,²⁾ исследует модальности «знания» и «веры», тогда как *временная логика* вводит модальности «иногда» и «всегда» (в «будущем» и «прошлом») вместе с их отрицаниями «часто» и «никогда».

Ввиду формального сходства между этими логическими системами их часто изучают все вместе или хотя бы по группам. Часто используют термин *модальная логика* для обозначения совокупности этих логик. Старейшая среди них — логика возможного. Часто именно ее называют модальной.

Название *модальная логика* происходит от того, что модальные логические системы вводят такие операторы над логическими формулами, которые позволяют модифицировать их интерпретацию. Например, в утверждениях «Возможно, что F », «Поль думает, что F », «Часто правда, что F », «В будущем, возможно, будет правда, что F », предшествующие логической формуле F выражения являются модальными операторами. Они могут относиться к какой угодно логической формуле F . Значение истинности этих формул зависит не только от значения истинности формулы F , которую они содержат, но и от момента провозглашения модальной формулы (временные логики), от человека, который думает, что F , или верит, что F (логики веры), или от необходимого, возможного или случайного характера некоторого факта (логики возможного).

3.1.12. Модальные операторы

Имеется сходство между определениями каждой из пар операторов (вроде «возможно»/«определенно», «иногда»/«всегда» и т. д.) и определением пары кванторов существования/общности в логике первого порядка. Это подсказывает следующее общее определение модальных операторов.

Модальный оператор общности Δ является:

- *либо имеем модального оператора,*

¹⁾ Или иначе — *эпистемологическая*. — Прим. перев.

²⁾ В философском контексте встречаются и другие названия — *логика веры, восприятия, предвидения, сомнения, вопросов*. — Прим. ред.

- либо выражением, состоящим из имени модального оператора, за которым следует список (t_1, \dots, t_n) термов t_i ($i = 1, \dots, n$).

Модальный оператор существования ∇ определяется через отрицание модального оператора общности:

$$\nabla F =_{\text{def}} \neg \Delta \neg F. \quad (3.1)$$

3.1.13. Примеры модальных операторов

- Алетические операторы:

— \Box : необходимо;

$\Box A$ истинно тогда и только тогда, когда A necessarily истинно, (b)

обходимо истинно, (a)

или A истинно во всех возможных мирах. (c)

— \Diamond : возможно;

$\Diamond A$ истинно, если A может оказаться истинным, (a)

или если A условно истинно, (b)

или если A истинно в некотором возможном мире. (c)

(Интерпретации (a), (b) и (c) для \Box и \Diamond соответственны; в каждой из пар (a), (b) и (c) сгруппированы двойственные интерпретации.)

- Временные операторы:

— G : всегда (в будущем);

GA истинно, если A останется истинным навсегда.

— H : всегда (в прошлом);

HA истинно, если A всегда было истинным.

— F : иногда (в будущем);

FA истинно, если A иногда будет истинным.

— P : иногда (в прошлом);

PA истинно, если A иногда оказывалось истинным.

— U : до тех пор, пока;

$U(A, B)$ истинно, если A истинно (начиная с текущего момента) до тех пор, пока B не станет истинным в некоторый момент в будущем. (G, F и H, P двойственные операторы в смысле

$FA \equiv \neg G \neg A$ и $PA \equiv \neg H \neg A$. Некоторые временные логики учитывают только будущее. В этом случае часто употребляют обозначения \square и \diamond соответственно для операторов G и F .)

● Эпистемические операторы:

— *верит*(a)

верит(a) A истинно, если индивид a верит в формулу A .

Очевидно, что модальные формулы бесчисленны. Философы предложили и использовали их в огромном количестве. Все эти формулы представимы с помощью модальных операторов.

3.1.14. Синтаксис модальной логики предикатов

Пусть M_1, M_2, \dots, M_n — модальные операторы. Правила образования модальных формул таковы:

- Все правила построения из логики предикатов (первого порядка) являются также правилами построения в модальной логике предикатов.
- Если F — формула и M_j — модальный оператор, то $M_j F$ — формула.

И опять основной задачей представления знаний является перевод фраз или описаний, относящихся к области экспертизы, в формулы модальной логики предикатов. Семантическое значение этих модальных формул должно соответствовать истинам области экспертизы.

3.1.15. Примеры

- По-русски: Возможно, что Жак посылает книгу Мари,
Логически: $\diamond \text{Посылка}(\text{Жак_2}, \text{Мари_4}, \text{Книга_22})$.
- По-русски: Не было возможным, чтобы Жак посылал что-нибудь каждому,
Логически: $P(\neg(\diamond(\forall x \exists y \exists z$
 $\text{Отправитель}(z, \text{Жак_2}) \wedge$
 $\text{Получатель}(z, x) \wedge$
 $\text{Объект}(z, y) \wedge$
 $\text{Элем}(z, \text{посылки}))))).$

- (Словесное прочтение этой формулы таково: в прошлом (P) не является (\neg) возможным (\Diamond), чтобы Жак посылал что-нибудь каждому.)
- По-русски: Если Жак верит данному высказыванию, то он верит, что верит ему (аксиома позитивной интроспекции),
 - Логически: $\text{верит}(\text{Жак}_2) p \supset \text{верит}(\text{Жак}_2) (\text{верит}(\text{Жак}_2) p)$.
 - По-русски: Если Жак не верит данному высказыванию, то он верит, что не верит ему (аксиома негативной интроспекции),
 - Логически: $\neg \text{верит}(\text{Жак}_2) p \supset \text{верит}(\text{Жак}_2) (\neg \text{верит}(\text{Жак}_2) p)$.
 - По-русски: Если факт, что Жак верит данному высказыванию, влечет, что оно истинно, то Жак знает это высказывание,
 - Логически: $(\text{верит}(\text{Жак}_2) p \supset p) \supset (\text{знает}(\text{Жак}_2) p)$. (Во многих случаях желательно, чтобы убеждения людей были истинными. Это приводит к понятию скорее знания, чем веры.)
 - По-русски: Жак верит, что он послал книгу (а не что-нибудь иное) Мари,
 - Логически: $\text{верит}(\text{Жак}_2)(P\exists x[\text{Посылка}(\text{Жак}_2, \text{Мари}_4, x) \wedge \text{Элем}(x, \text{книги})])$.
 - По-русски: Жак верит, что он послал книгу (вполне определенную) Мари,
 - Логически: $\exists x[\text{верит}(\text{Жак}_2)(P\text{Посылка}(\text{Жак}_2, \text{Мари}_4, x)) \wedge \text{Элем}(x, \text{книги})]$.

(Последнее выражение показывает, что в системе убеждений Жака имеется формула типа $[P \text{Посылка}(\text{Жак}_2, \text{Мари}_4, x) \wedge \text{Элем}(x, \text{книги})]$. Но ее окончательный вид зависит от конкретного объекта, к которому относится переменная x . Предпоследнее выражение показывает, что в системе убеждений Жака содержится формула $(P\exists x \text{Посылка} \wedge \text{Элем})$.)

3.1.16. Трехзначная семантика для модальной логики предикатов

Семантику для модальной логики предикатов можно определить, как для классической (§ 1.2.5). Простоты ради временно проигнорируем более сложные языки, основанные на модальной логике. Проиллюстрируем модальную семантику, введя аппроксимацию некоторых форм логики возможного с помощью трехзначной логики. Бинарная логика с двумя значениями {Л или 0, И или 1} самая элементарная. Истина и ложь — это два множества высказываний, и законы (классической) логики утверждают, что любое высказывание — есть элемент хотя бы одного из этих множеств (закон исключенного третьего) и что никакое высказывание не является элементом сразу двух этих множеств (закон противоречия).

Какие изменения надо внести в эту теорию, если вводятся модальности «возможно» и «необходимо»? Надо рассмотреть несколько классов высказываний [75]. Обозначим через N класс «необходимых» высказываний, через P — «возможных», через I — «невозможных» (или «абсурдных») и через C — «нейтральных» (или «возможно (случайно) ложных»). Никакое высказывание не принадлежит одновременно N и C или I и P . Это закон противоречия. Далее, класс N содержится в P , а класс I — в C . Это отражено в законах следования возможного из необходимого и нейтрального из абсурдного:

*любое необходимое высказывание возможно,
любое абсурдное высказывание не является необходимым.*

Существуют высказывания, которые являются возможными и нейтральными одновременно. Их называют «проблематичными». Множество таких высказываний обозначим через U . Имеет место закон исключенного четвертого:

любое высказывание принадлежит либо N , либо U , либо I .

Посмотрим, как эту теорию модальности можно перевести в алгебраическую форму. Каждому из классов

		$F \wedge G$			$F \vee G$			$F \supset G$			$F \equiv G$			
F	$\neg F$	$F \setminus G$	0	1	2	0	1	2	0	1	2	0	1	2
0	2	0	0	0	0	0	1	2	2	2	2	2	1	0
1	1	1	0	1	1	1	1	2	1	2	2	1	2	1
2	0	2	0	1	2	2	2	2	0	1	2	0	1	2

Рис. 3.1. Таблица трехзначных функций (связки Лукасевича).

F	$\Diamond F$	$\Box F$
0	0	0
1	2	0
2	2	2

Рис. 3.2. Трехзначная таблица модальных операторов.

N , U , I соответствует своя интерпретация: «необходимо», «проблематично», «невозможно». Возьмем три символа 2, 1, 0. Это логические значения, сопоставляемые указанным интерпретациям. Каждому высказыванию можно приписать логическое значение. Эта трехзначная логика предложена Лукасевичем [64].

В логике Лукасевича каждое высказывание обладает одним из значений 0, 1 или 2 (интерпретируемых как семантическое значение высказывания). Семантическое значение модальной формулы можно находить, используя таблицы, приведенные на рис. 3.1 и 3.2. Они задают семантику для аппроксимации модальной логики возможного с помощью трехзначной логики. Иное описание этой семантики дается в следующем параграфе.

3.1.17. Семантика возможных миров

Множество истинных событий можно разделить на «случайно истинные» (которые истинны, но могли бы оказаться ложными, если бы развитие мира шло иначе; таковы исторические события) и «необходимо истинные» (которые нельзя отрицать, не ставя под сомнение нормальный смысл слов языка; таковы математические и логические истины). Точно так же лож-

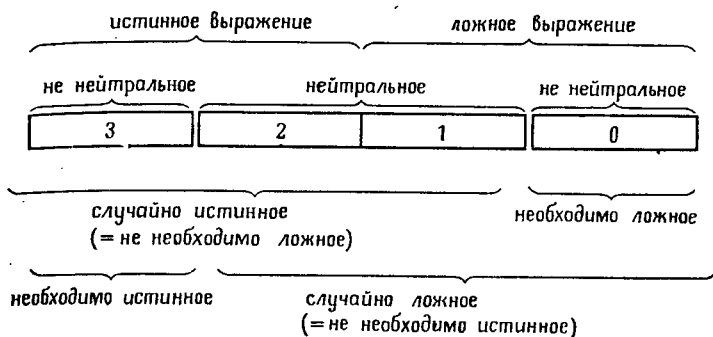


Рис. 3.3. Семантика возможных миров.

ные события делят на «случайно ложные» и «необходимо ложные». Будем предполагать, что «необходимая истина» (в существующем мире) связана с событием, подтверждающимся не только в существующем, но и во всех возможных мирах. Таким же образом «возможная истина» будет связана с событием, подтвердившимся в существующем, но не во всех возможных мирах. «Необходимо ложное» событие не подтверждается ни в каком из возможных миров, тогда как «случайно ложное» ложно в существующем, но не во всех возможных мирах. Это деление событий можно изобразить в виде диаграммы (см. рис. 3.3).

Интерпретациям *необходимо истинно*, *нейтрально истинно*, *нейтрально ложно*, *необходимо ложно* можно сопоставить соответственно логические значения 3, 2, 1, 0. Итак, модальная логика, связанная с операторами *необходимо* и *случайно*, оказалась погруженной в четырехзначную логику [91], [22].

Предположим, что семантика возможных миров сводится к семантике с двумя возможностями: существующий мир X и возможный Y . Каждому высказыванию припишем одно из логических значений 0, 1, 2 или 3 в соответствии с тем, какому из приводимых ниже условий оно удовлетворяет:

- ложно в X и в Y (необходимо ложно),
- ложно в X и истинно в Y (случайно, но не необходимо ложно),

		$F \wedge G$						
F	$\neg F$	$F \setminus G$	0	1	2	3	$\Box F$	$\Diamond F$
0	3	0	0	0	0	0	0	0
1	2	1	0	1	0	1	0	3
2	1	2	0	0	2	2	0	3
3	0	3	0	1	2	3	3	3

(a)

(b)

(c)

(d)

(a)

(b)

(c)

(d)

Рис. 3.4. Четырехзначные логические операции и модальные операторы.

- истинно в X и ложно в Y (случайно, но не необходимо истинно),
- истинно в X и в Y (необходимо истинно).

Указанная интерпретация логических значений приводит к таблицам, представленным на рис. 3.4 для отрицания, конъюнкции и операторов \Box и \Diamond . Это задает некоторую модальную семантику для четырехзначной логики.

Заметим, что операторы отрицания и конъюнкции, описанные таблицами на рис. 3.4, являются прямым произведением двух экземпляров аналогичных операторов из бинарной системы $\{0, 1\}$. Это легко проверяется с использованием следующего двоичного кодирования логических значений 0, 1, 2, 3:

$$0 = (0, 0), \quad 1 = (0, 1), \quad 2 = (1, 0), \quad 3 = (1, 1).$$

3.1.18. Ламбда-исчисление

Обогатим модальное исчисление предикатов новым оператором над логическими переменными — *ламбда-оператором* (или, короче, λ -оператором). Читатель, несомненно, знаком с таким способом задания множеств, как в нижеследующих примерах:

$$\{x \mid x \text{ является преподавателем университета}\},$$

$$\{x \mid -1 < x < 1, x \text{ — действительное число}\}.$$

При таком определении конкретного множества двусмысленности нет (§ 1.2.4); нужно только явно указать ту переменную, на которой основано определение

множества. Можно поискать замену неформального определения формально логическим. Это приводит к обозначениям λ -исчисления, в котором множество описывается следующим образом:

λx (логическая формула, содержащая переменную x).

Например, рассмотрим русскую фразу «Кто-то посылает что-то Мари» и ее перевод на язык логики предикатов *Посылка* (x , *Мари*₄, y).

Выражение (называемое λ -выражением или λ -абстракцией)

$$\lambda y \text{ [Посылка(Жак}_2, \text{ Мари}_4, y)] \quad (3.2)$$

определяет характеристическую функцию множества объектов, посылаемых Жаком Мари, тогда как выражение

$$\lambda x \text{ [Посылка}(x, \text{ Мари}_4, \text{ Книга}_{22})] \quad (3.3)$$

определяет характеристическую функцию множества индивидов, посылающих указанную книгу Мари. Точно так же выражение

$$\lambda x, y \text{ [Посылка}(x, \text{ Мари}_4, y)] \quad (3.4)$$

определяет характеристическую функцию множества пар (отправитель, посылаемый объект), для которых Мари является получателем.

Если F — формула, то $\lambda x F$ указывает описанное с помощью F множество, на элементах которого (являющихся конкретизациями для x) F истинна. Выражения (3.2) и (3.3) можно интерпретировать и как унарные предикаты с аргументами y и x соответственно. Оператор λ , как и кванторы общности и существования, служит для связывания вхождений переменной.

Дадим более формализованное определение λ -выражения.

n -арное λ -выражение $\lambda x_1, \dots, x_n F$ состоит из логической формулы F , называемой телом выражения, и множества n переменных x_1, \dots, x_n формулы F , являющихся формальными параметрами выражения.

Следующий за λ список параметров служит для того, чтобы отличить формальные параметры от других аргументов (или концептов) формулы F .

Телом λ -выражения является формула логики предикатов. Как только будут заданы значения n формальных параметров, λ -выражение станет n -арным предикатом, принимающим значения «истинно» или «ложно» в зависимости от того, какие индивидуальные значения (конкретизации) присвоены формальным параметрам.

Синтаксис и семантика λ -выражения проистекает из его определения.

Синтаксис Если F — логическая формула и x — переменная, то $\lambda x F$ — λ -выражение.

Семантика Значения λ -выражения $\lambda x_1, \dots, x_n F$ есть значения n -арного предиката, полученного присваиванием конкретизаций формальным параметрам.

Выражение (3.4) становится бинарным предикатом, который можно обозначить, например, *Лямбда-посылка*(x, y). Он примет значение **И** или **Л** в зависимости от осуществленного присваивания конкретизаций переменным x и y . В некоторых системах баз данных λ -выражения используются для постановки вопросов. Выражение (3.4) соответствовало бы вопросу (запросу): «Назовите посланные для Мари объекты и соответствующих отправителей».

3.1.19. Рассуждения, использующие логические формулы

Исчисление предикатов содержит правила вывода, применимые к одним логическим формулам для получения других. Особенно важны правила «modus ponens» и «обобщения» (§§ 2.1.3, 2.1.8). Правила вывода порождают некое множество формул из примитивных (исходных, первоначальных) формул. В исчислении предикатов выведенные формулы называются «теоремами», а последовательность примененных для их получения правил вывода ¹⁾ — «доказатель-

¹⁾ С указанием используемых в них формул. — *Прим. ред.*

ством теоремы». Многочисленные приложения исчисления предикатов к ИИ можно толковать как методы доказательства теорем.

При доказательстве теорем обычно используют *процедуры опровержения*. Множество гипотез $\{H_j\}$, подходящих для доказательства теоремы, добавляют к множеству присущих области экспертизы аксиом $\{A_i\}$. Затем стремятся получить опровержение (или прийти к противоречию), присоединяя $\{\neg C\}$ — отрицание утверждения теоремы — к системе $\{H_j, A_i\}$ и пытаясь доказать формулу.

$$\bigwedge_j H_j \wedge \bigwedge_i A_i \wedge \neg C \supset \text{Л.}$$

Эта формула логически эквивалентна формуле

$$\bigwedge_i A_i \supset \left(\bigwedge_j H_j \supset C \right).$$

Процедуры, позволяющие строить доказательства формул такого типа, называются «доказательствами посредством опровержения». Они помогают избегать менее перспективных направлений поиска. Системы доказательства теорем во многих приложениях включают формулы логики предикатов с переменными, связанными кванторами существования. В таких случаях доказательства позволяют находить конкретизации для этих переменных.

Например, интересно выяснить, можно ли формулу $\exists x C(x)$ логически вывести из гипотез и аксиом. Если да, то хотелось бы знать конкретизацию для x в терминах вывода. Попытка доказательства $\exists x C(x)$ из $\{H_j, A_i\}$ должна быть конструктивной. Проиллюстрируем подобное доказательство следующим примером.

3.1.20. Пример

Аксиом нет, гипотезы:

H_1 : таможенники возвращают всех, кто въехал в страну без паспорта,

H_2 : люди на машинах въехали в страну и были возвращены лишь другими людьми на машинах,
 H_3 : ни один человек на машине не имел паспорта.

Тезис (или заключение):

C : некоторые таможенники были на машинах.
 Рассмотрим пять следующих предикатов.

$E(x)$: x въехал в страну,
 $P(x)$: x с паспортом,
 $R(x, y)$: y возвращает x ,
 $M(x)$: x на машине,
 $D(x)$: x — таможенник.

Можно перевести гипотезы H_i и тезис C в термины логики предикатов:

$H_1 : \forall x ((E(x) \wedge \neg P(x)) \supset \exists y (R(x, y) \wedge D(y))),$
 $H_2 : \exists x ((M(x) \wedge E(x)) \wedge \forall y (R(x, y) \supset M(y))),$
 $H_3 : \forall x (M(x) \supset \neg P(x)),$
 $C : \exists x (M(x) \wedge D(x)).$

Запишем гипотезы и отрицание тезиса в сколемовой форме (§§ 1.2.7—1.2.8) и исключим импликации:

$H_1 : \neg E(x) \vee P(x) \vee (R(x, f(x)) \wedge D(f(x))),$
 $H_2 : M(a) \wedge E(a) \wedge (\neg R(a, y) \vee M(y)),$
 $H_3 : \neg P(x) \vee \neg M(x),$
 $\neg C : \neg M(x) \vee \neg D(x).$

При доказательстве теоремы в качестве правила вывода можно использовать неклаузальную резолюцию (§§ 1.1.15, 1.2.14). Вот одна из последовательностей операций, составляющих доказательство теоремы (обозначения из §§ 1.1.15 и 1.2.14):

$B_1 : P(a) \vee (R(a, f(a)) \wedge D(f(a)))$	$\perp_E \{(x, a)\} (H_2, H_1)$
$B_2 : \neg P(a)$	$\perp_M \{(x, a)\} (H_2, H_3)$
$B_3 : R(a, f(a)) \wedge D(f(a))$	$\perp_P (B_1, B_2)$
$B_4 : M(a) \wedge E(a) \wedge M(f(a))$	$\perp_R \{(y, f(a))\} (B_3, H_2)$
$B_5 : \neg D(f(a))$	$\perp_M \{(x, f(a))\} (B_4, \neg C)$
$B_6 : \perp$	$\perp_D (B_3, B_5)$

3.1.21. Рассуждения по поводу знаний

В большинстве встречающихся в ИИ систем относящиеся к области экспертизы знания делятся на две категории: «факты» и «правила» (§ 1.1.16). Факты — это данные (представленные предикатами), касающиеся области экспертизы. Например, данные о персонале некоего университета составляют множество фактов.

● Факт (1).

По-русски: Жак — профессор факультета информатики,

Логически: *Проф(Инфо, Жак_2)*.

● Факт (2).

По-русски: Мари — студентка математического факультета,

Логически: *Студ(Мат, Мари_4)*.

Правила — это данные, представленные с помощью импликации (или в иной эквивалентной логической форме). Они представляют собой обобщающие знания об области экспертизы.

● Правило (1).

По-русски: Если y — профессор факультета x и w — студент(ка) факультета z при $x \neq z$, то y может служить внешним экзаменатором для w ,

Логически: $\text{Проф}(x, y) \wedge \text{Студ}(z, w) \wedge \neg \text{Равно}(x, z) \supset \text{Экзам}(y, w)$.

Задача доказательства (обоснования) теоремы состоит в установлении выводимости из фактов и правил некой формулы («предложения-цели», «заключения»), представляющей некоторый вопрос.

● Предложение-цель (1).

По-русски: Может ли Жак служить внешним экзаменатором для Мари?

Логически: *Экзам(Жак_2, Мари_4)*.

Можно применять различные приемы для выработки стратегий доказательства теоремы.

3.1.22. Системы прямой дедукции

В системах *прямой дедукции* новые знания получают, применяя выводы к фактам и правилам. Алгоритм завершает работу при получении некоторого знания, эквивалентного цели (или непосредственно влекущего ее). Для иллюстрации системы прямой дедукции обратимся снова к примеру из § 3.1.21. Докажем теорему

$(\text{Факт}(1) \wedge \text{Факт}(2) \wedge \text{Правило}(1)) \supset \text{Цель}(1).$

● Этап (1):

Преобразуем $\text{Факт}(1)$ и $\text{Правило}(1)$ в дизъюнкты, чтобы применить затем метод резолюций (§§ 1.1.12, 1.2.14). С помощью резолюции выводим новое $\text{Правило}(2)$, используя обозначение

Факт(1) Правило(1):

Правило(2)

Факт(1):

$\text{Проф}(\text{Инфо}, \text{Жак_2})$

Правило(1):

$\neg \text{Проф}(x, y) \vee$

$\neg \text{Студ}(z, w) \vee$

$\text{Равно}(x, z) \vee \text{Экзам}(y, w)$

Правило(2): $\neg \text{Студ}(z, w) \vee \text{Равно}(\text{Инфо}, z) \vee$
 $\vee \text{Экзам}(\text{Жак_2}, w)$

● Этап (2): из Факта (2) и Правила (2) резолюцией выводим новый

Факт(3):

Факт(2):

$\text{Студ}(\text{Мат}, \text{Мари_4})$

Правило(2):

$\neg \text{Студ}(z, w) \vee \text{Равно}$
 $(\text{Инфо}, z) \vee \text{Экзам}$
 $(\text{Жак_2}, w)$

Факт(3): $\text{Экзам}(\text{Жак_2}, \text{Мари_4}) \vee$

$\vee \text{Равно}(\text{Инфо}, \text{Мат}) = \text{Л}$

(Отношение $\text{Равно}(\text{Инфо}, \text{Мат}) = \text{Л}$ должно быть явно указано в БД)

● Этап (3):

Факт (3) соответствует Цели (1). Следовательно, она подтверждена. Аналогично выведем утверждение Л из Факта (3) и отрицания Цели (1):

Факт (3): \neg Цель (1):

Экзам(Жак_2, Мари_4) \neg Экзам(Жак_2, Мари_4)

Л

Систему прямой дедукции можно толковать как систему, в которой применяется теорема о прямой дедукции (см. § 1.1.9): *Если F_1, \dots, F_n, G — логические выражения, то G является логическим следствием из F_1, \dots, F_n тогда и только тогда, когда логическое выражение $(F_1 \wedge \dots \wedge F_n \wedge \neg G)$ тождественно ложно, т. е. невыполнимо.* Правила вывода и стратегии, используемые в системах прямой дедукции, графически представимы И/ИЛИ-деревьями [83]. Эти деревья будут введены в § 5.1.10 в рамках описания стратегии, используемой языком логического программирования Пролог.

3.1.23. Системы обратной дедукции

В системах *обратной дедукции* выводы применяют к цели и к правилам, чтобы построить новые частичные цели. Алгоритм завершает работу, когда все частичные цели соответствуют фактам. Такую систему можно толковать с логической точки зрения как систему, в которой применяется теорема об обратной дедукции, которая гласит (см. § 1.1.9):

Если F_1, \dots, F_n, G — логические выражения, то G является логическим следствием из F_1, \dots, F_n тогда и только тогда, когда логическое выражение $(\neg F_1 \vee \dots \vee \neg F_n \vee G)$ тождественно истинно, т. е. общезначимо.

В системах обратной дедукции правила и цели преобразуют в конъюнкты, чтобы применить затем правило согласия (§ 1.2.14).

- Этап (1): из Цели (1) и отрицания Правила (1), используя правило согласия, выводим новую Цель (2):

Цель (1):*Экза́м (Жак_2, Мари_4)***Правило (1):**

$$\begin{aligned} & \text{Проф}(x, y) \wedge \\ & \wedge \text{Студ}(z, w) \wedge \neg \\ & \text{Равно}(x, z) \wedge \neg \text{Экза́м} \\ & (y, w) \end{aligned}$$

Цель (2): *Проф*(*x*, *Жак_2*) \wedge *Студ*(*z*, *Мари_4*) \wedge
 \neg *Равно*(*x*, *z*)

- Этап (2): Из Цели (2) и отрицания Факта (1) с помощью правила согласия выводим Цель (3).

Цель (2):

Проф(*x*, *Жак_2*) \wedge
Студ(*z*, *Мари_4*) \wedge
 \neg *Равно*(*x*, *y*)

Правило (1):
 \neg *Проф*(*Инфо*, *Жак_2*)

Цель (3): *Студ*(*z*, *Мари_4*) \wedge \neg *Равно*(*Инфо*, *z*)

- Этап (3): из Цели (3) и отрицания Факта (2) выводим теорему:

Цель (3):

Студ(*z*, *Мари_4*) \wedge
 \neg *Равно*(*Инфо*, *z*)

Правило (2):
 \neg *Студ*(*Мат*, *Мари_4*)

$$\neg \text{Равно}(\text{Инфо}, \text{Мат}) = \text{И}$$

3.2. Сетевое представление

3.2.1. Введение

В разд. 3.1 описано использование исчисления предикатов для представления знаний. Исчисление предикатов можно комбинировать с такими сравнительно эффективными механизмами вывода, как резолюция (§§ 1.1.12, 1.2.14). Таким образом, логические формализмы представления знаний охватывают проблему рассуждений достаточно изящно.

Недостаток логического формализма — его неструктурированность: например, для сбора всей информации по одному объекту (конкретизации) прихо-

дится пробегать все множество логических формул некой базы данных (БД). Графические представления, к которым мы переходим, служат глобализации и структурированию информации. Граф собирает вокруг одного узла всю информацию по некоторому объекту. Графические представления, такие, как *концептуальные графы* и *семантические сети*, позволяют визуализировать модель мира, которому принадлежит решаемая проблема. Такой тип представления интересуется в большей мере описанием мира проблем, чем правил, относящихся к самому этому миру.

Концептуальный граф представляет логическую формулу. Имена и аргументы предикатов представлены в нем соответственно двумя типами узлов. Дуги графа соединяют имена предикатов с их аргументами. Эти графы близки к моделям, используемым специалистами по БД. Одна из проблем при создании БД — учет структуры области экспертизы. Возьмем, к примеру, структуры, существующие в недрах некоего предприятия. Различные отделы, секции, бюро и субъекты, их заполняющие, образуют иерархическую структуру. Ее можно представить набором концептуальных графов.

Семантические сети представляют более сложные структуры. Такая сеть состоит из множества концептуальных графов, представляющих логические формулы. Она позволяет визуализировать множество отношений между концептуальными графами, ее составляющими. Ею описывается также универсум, в который погружены концептуальные графы (что является глобальной областью экспертизы).

Концептуальные графы и семантические сети составляют графическую версию исчисления предикатов.

3.2.2. Концептуальные графы

Логика предикатов — это язык, который можно интерпретировать в терминах области рассуждений (экспертизы): логические формулы представляют фразы метаязыка. Аргументы предикатов и логических функций существенно используются для представления

атрибутов, событий и состояний. Имена предикатов указывают способ увязывания этих понятий. В частности, эти имена представляют правила соединения, правила грамматики и процедуры.

Концептуальные графы содержат прямоугольники для представления аргументов и круги для имен предикатов. Круг соединяется стрелкой с прямоугольником, если они представляют соответственно имя и аргумент одного и того же предиката.

Предикаты могут иметь несколько аргументов. Следовательно, круги могут иметь несколько входящих и/или выходящих стрелок. Большинство предикатов, используемых для представления знаний, обладают двумя аргументами (бинарные предикаты). Тогда круги соединены с прямоугольниками двумя стрелками: входящей и выходящей. Проиллюстрируем эти понятия на примере.

3.2.3. Пример и терминология

Снова обратимся к примеру из § 3.1.6. На рис. 3.5 изображен концептуальный граф предиката *Посылка* (*Жак_2*, *Мари_4*, *Книга_22*), формализующего фразу «Жак посылает книгу Мари». На рис. 3.6 приведен концептуальный граф логического представления той же фразы бинарными предикатами, т. е. граф, соответствующий формуле

$$\begin{aligned} & \text{Отправитель}(\text{Посылка_8}, \text{Жак_2}) \wedge \\ & \text{Получатель}(\text{Посылка_8}, \text{Мари_4}) \wedge \\ & \text{Объект}(\text{Посылка_8}, \text{Книга_22}) \wedge \\ & \text{Элем}(\text{Посылка_8}, \text{посылки}). \end{aligned} \quad (3.5)$$

При графическом представлении бинарных предикатов круги иногда опускают, а имена предикатов указывают на стрелках. В § 3.1.6 было описано общее правило преобразования m -арного предиката ($m \geq 2$) вида

Предикатное_имя(*значение_1*, *значение_2*, ..., *значение_m*) в произведение m бинарных предикатов

$$\wedge \text{Функция_j}(\text{предикатное_имя}, \text{значение_j}).$$

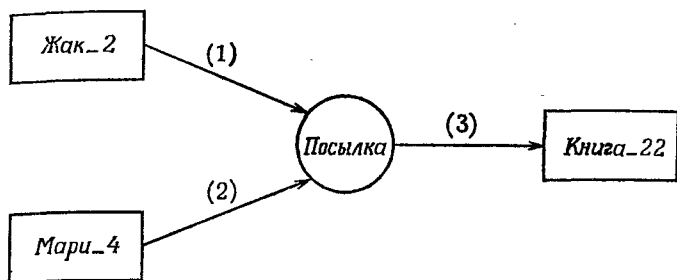


Рис. 3.5. Концептуальный граф.

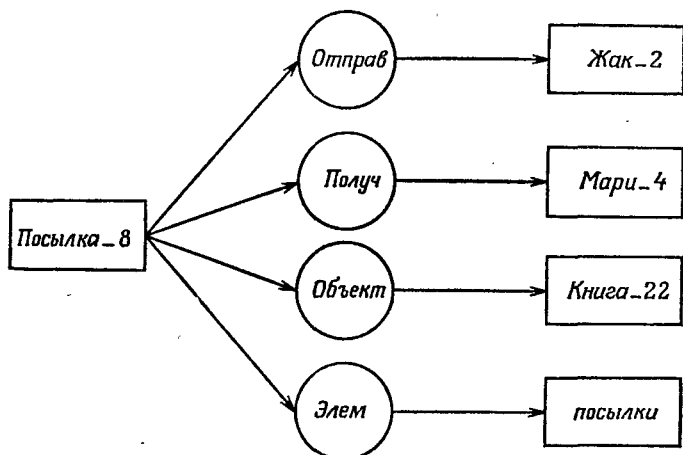


Рис. 3.6. Концептуальный граф логического представления бинарными предикатами.



Рис. 3.7. Концептуальный граф бинарного предиката.

Свяжем с каждым бинарным предикатом концептуальный граф — как показано на рис. 3.7. Стрелки предикатов направлены от первого аргумента

предиката (*предикатное_имя*) к имени предиката (*Функция_j*) и от имени предиката ко второму аргументу (*значение_j*). В записи, называемой по-английски slot-assertion (§ 3.1.6), имя предиката соответствует «имени слота», а значение — «значению слота». Вообще в концептуальных графах «значение слота» соответствует некоторому концепту.

Графическое представление m -арных предикатов основано на следующем соглашении. Стрелка, соответствующая m -му аргументу, направлена к прямоугольнику, представляющему этот аргумент. Все остальные стрелки направлены от прямоугольников к кругу, представляющему имя предиката. Стрелки помечены от 1 до m , чтобы явно показать соответствие между стрелкой и аргументом (рис. 3.5).

В дальнейшем нас интересует исключительно представление бинарных предикатов. Добавим несколько штрихов по поводу терминологии, широко используемой для концептуальных графов бинарных предикатов. Вообще имя бинарного предиката представляет некую функцию. Узел концептуального графа, указывающий это имя, называется *связывающим узлом*, ибо он связывает два концепта, представленных двумя аргументами бинарного предиката. Функция, представленная этим связывающим узлом, носит по той же причине название *концептуального отношения*. Термины «имя бинарного предиката» и «аргумент», представляющие соответственно функцию и концепт, часто замènяют на «связывающий узел» и «узел-концепт» в графическом представлении.

3.2.4. Семантические сети

В § 3.2.1 указывалось, что каждый концептуальный граф представляет одну логическую формулу. *Семантическая сеть* гораздо сложнее. Она представляет не только набор (соединение) формул, но также описывает их взаимосвязи и погружение их в контекст области рассуждений. Семантические сети получаются из концептуальных графов по правилам соединения, которые будут введены посредством примеров.

3.2.5. Правила конъюнкции и упрощения

Рассмотрим набор из трех фраз:

Фраза 1: Жак пишет книгу,

Фраза 2: Жак посылает эту книгу Мари,

Фраза 3: Мари читает книгу (которую Жак ей прислал).

Каждую из этих фраз можно представить либо формулой исчисления предикатов, либо концептуальным графом. Первый этап построения семантической сети основан на использовании двух следующих формальных правил получения концептуального графа g из двух графов g_1 и g_2 .

Правило конъюнкции. Если узел-концепт c_1 в g_1 идентичен узлу-концепту c_2 в g_2 , то g получается удалением c_2 и соединением с c_1 всех связывающих узлов, которые были связаны с c_2 в g_2 .

Правило упрощения. Если концептуальный граф g содержит два идентичных (соединенных с одними и теми же узлами-концептами) связывающих узла, то можно удалить один из них вместе со связанными с ним стрелками.

Применим правило конъюнкции к приведенному только что примеру. «Книга, которую написал Жак, которую он послал Мари и которую Мари читала» представляется конкретизацией *Книга_22*. Эта конкретизация «книги» появляется в концептуальных графах, представляющих рассмотренные выше фразы 1, 2 и 3. Группирование конкретизаций *Книга_22*, *Жак_2* и *Мари_4* порождает концептуальный граф, изображенный на рис. 3.8. На нем рамками выделены графы фраз 1, 2, 3 и граф группы фраз (1, 2, 3).

3.2.6. Представление контекста

Сам по себе концептуальный граф несет немного информации, тогда как включение в семантическую сеть позволяет связать его концепты и функциональные связи с областью рассуждений (экспертизы). Например, *Книга_22* — конкретизация слова «книга», которое представляет абстрактное понятие (или, иначе, *тип*). Отношение принадлежности типу представлено

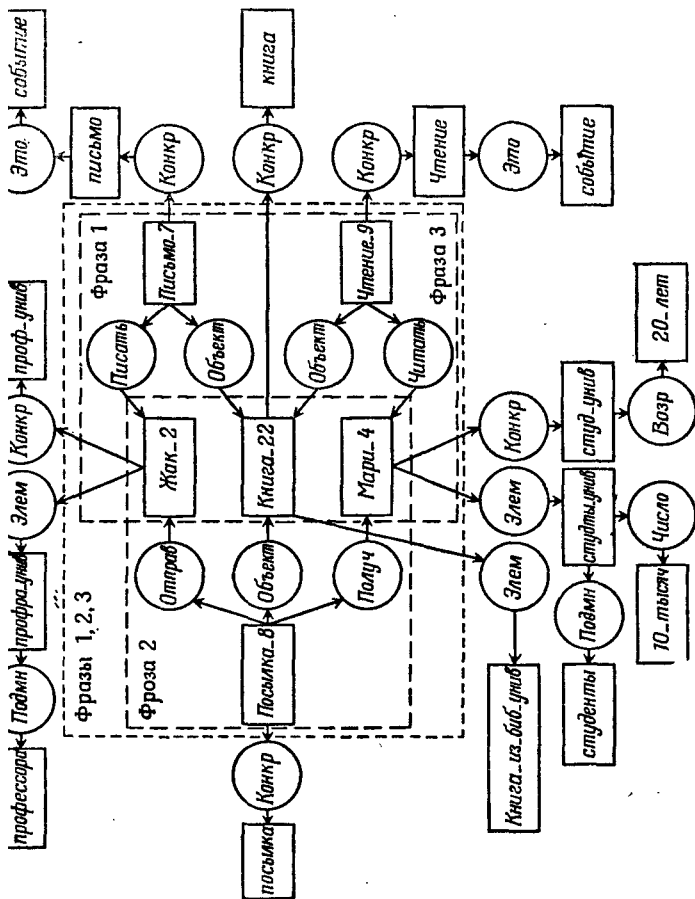


Рис. 3.8. Концептуальный граф и контекстное представление.

связывающим узлом (именем предиката) *Конкр* (от слова «конкретизация»).

Кроме факта, что *Книга_22* относится к типу «книга», мы хотим показать, что она принадлежит некоторому «множеству книг» (например, из библиотеки некоторого университета). Отношение принадлежности множеству представлено связывающим узлом (именем предиката) *Элем* (от слова «элемент»). На рис. 3.8 конкретизации *Жак_2*, *Книга_22* и *Мари_4* связаны с различными типами и множествами, которым они принадлежат. Отметим, что «тип» употребляется в абстрактном смысле (абстрактный тип). Утверждения о типах — *аналитические*, например: средний возраст студентов университета двадцать лет. Наоборот, свойства множеств — *синтетические*: число студентов университета равно десяти тысячам.

Используем связывающий узел, представляющий имя предиката *Подмн* для представления отношения между двумя множествами, из которых первое — подмножество второго. Как *Элем* объявляет о принадлежности элемента множеству, *Подмн* означает включение одного множества в другое. *Элем* и *Подмн* отражают принадлежность множеству.

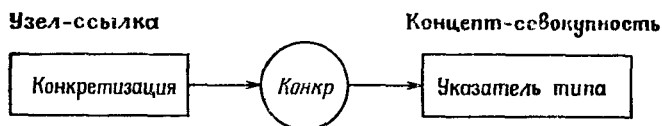
Связывающий узел *Это* используется для представления отношения между первым типом и более общим вторым. Напомним, что предикат *Конкр* когда-то использовался нами для представления принадлежности индивида типу. *Это* и *Конкр* отражают принадлежность типу.

Концепты в семантических сетях обладают различными свойствами, делящимися на аналитические (свойства «типа») и синтетические (свойства «множества»). Важно учитывать это различие при построении сетей, чтобы избежать некорректных выводов.

3.2.7. Представление «совокупность¹⁾-ссылка»

Существует графическое представление (эквивалентное рассмотрению в предыдущих параграфах) с

¹⁾ В оригинале: родовой (générique). — Прим. ред.



Представление полей совокупности и ссылки

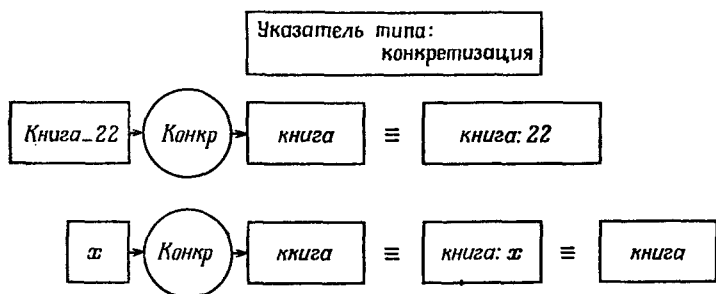


Рис. 3.9. Поля ссылки и совокупности.

узлами, представляющими типы и называемыми «узлами-совокупностями», или «родовыми узлами», и узлами, представляющими конкретизации типов и называемыми «узлами-индивидами» или «узлами-ссылками».

Семантическая сеть образуется последовательностями из трех узлов, соединенных так: за узлом-ссылкой (например, представляющим конкретизацию *Книга_22*) следует связывающий узел *Конкр*, сопровождаемый узлом-совокупностью (например, представляющим тип «книга»).

Используя метод Совы [102], эти три узла можно сгруппировать и представить одним «узлом-прямоугольником», состоящим из двух полей: из «поля совокупности», содержащего некоторый тип, и следующего за этим полем «поля ссылки», например, с конкретизацией типа из первого поля (рис. 3.9). Здесь поле ссылки (справа от двоеточия) представляет конкретизацию *Книга_22*. Помимо конкретизации его содержимым может быть переменная. Поле совокупности (слева от двоеточия) представляет тип. Представление [*книга* : *Книга_22*] (или, короче, [*книга* : 22]) означает вполне определенный объект типа

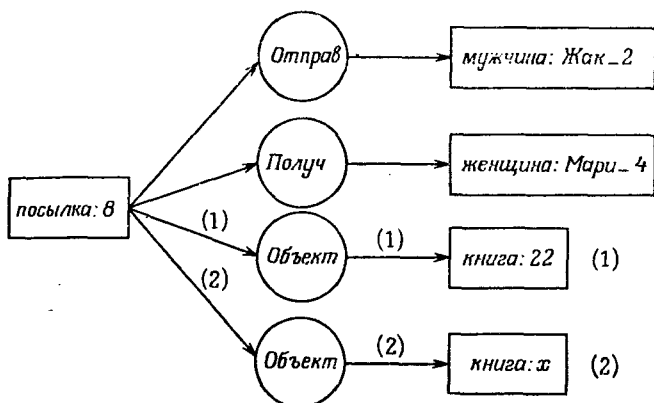


Рис. 3.10. Пример концептуального графа.

«книга». Представление [книга: x] означает просто какой-то объект типа «книга». Отметим, что наличие переменной в правом поле вообще необязательно. Например, можно заменить представление [книга: x] эквивалентным ему представлением [книга] (рис. 3.9).

3.2.8. Пример

Фразы

- Жак посылает (вполне определенную) книгу Мари,
- Жак посылает (какую-то) книгу Мари

представляются двумя графами, которые можно выделить на рис. 3.10: первая фраза — графом без элементов с (2), вторая фраза — графом без элементов с (1).

3.2.9. Пример введения кванторов

Введем графическое обозначение кванторов на примере трех фраз.

- (1): Жак посылает (какую-то) книгу каждой женщине,

- (2): Жак посылает всякую книгу каждой женщине,
- (3): Жак посылает (одну и ту же) книгу каждой женщине.

Запишем эти фразы в логике предикатов:

- (1): $\forall x \exists y \exists z [\text{Отправитель}(z, \text{Жак_2}) \wedge \text{Получатель}(z, x) \wedge \text{Объект}(z, y) \wedge \text{Конкр}(z, \text{посылка}) \wedge \text{Конкр}(y, \text{книга}) \wedge \text{Конкр}(x, \text{женщина})]$,
- (2): $\forall x \forall y \exists z [\text{Отправитель}(z, \text{Жак_2}) \wedge \text{Получатель}(z, x) \wedge \text{Объект}(z, y) \wedge \text{Конкр}(z, \text{посылка}) \wedge \text{Конкр}(y, \text{книга}) \wedge \text{Конкр}(x, \text{женщина})]$,
- (3): $\exists y \forall x \exists z [\text{Отправитель}(z, \text{Жак_2}) \wedge \text{Получатель}(z, x) \wedge \text{Объект}(z, y) \wedge \text{Конкр}(z, \text{посылка}) \wedge \text{Конкр}(y, \text{книга}) \wedge \text{Конкр}(x, \text{женщина})]$.

Перевод с логического языка на графический осуществляется по следующим правилам.

Прежде всего концептуальный граф делят на иерархическое множество зон, каждая из которых соответствует области действия одного или нескольких кванторов. Например, рассмотрим логическую форму фразы (1). Соответствующий граф изображен на рис. 3.11 (элементы с (2) удалить). Для графического изображения нужно ввести обозначение области действия квантора общности по переменной x . С этой целью установим иерархию прямоугольников концептуального графа. На рис. 3.11 и 3.12 жирно выделены прямоугольники наивысшего порядка. Они содержат конкретизации (*Конкр*) концепта совокупности «формула под квантором общности» (на рис. 3.11 и 3.12 этот концепт обозначен \forall *Квант_форм*). Каждая переменная под квантором общности в этих формулах представлена связывающим узлом со знаком \forall .

Читатель может проверить, что с учетом этих соглашений относительно записи рис. 3.11, без элементов с (2), изображает формулу (1), что он же, но

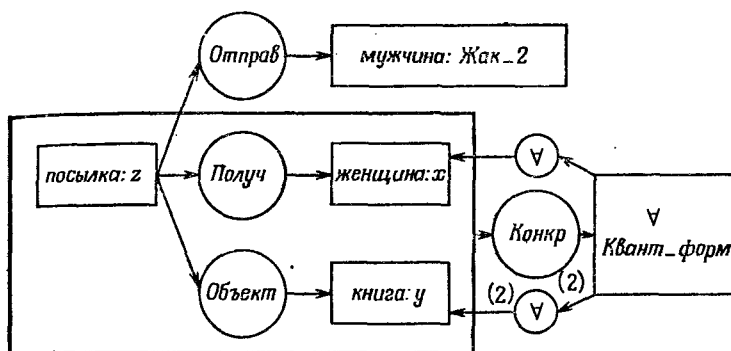


Рис. 3.11. Концептуальный граф (формулы 1 и 2) с квантором.

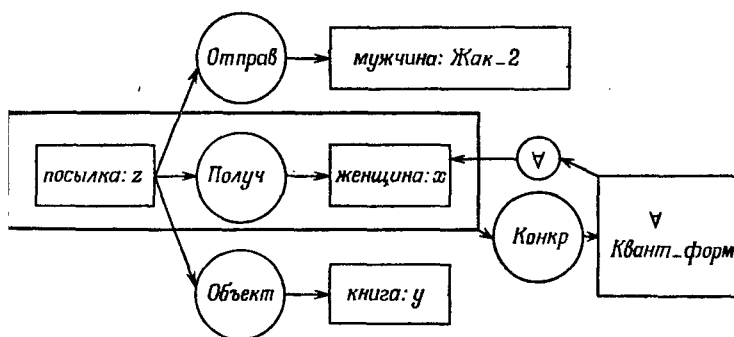


Рис. 3.12. Концептуальный граф (формулы 3) с квантором.

вместе с элементами, помеченными знаком (2), изображает формулу (2) и что рис. 3.12 представляет формулу (3).

3.2.10. Временные и модальные операторы

Используемые в логике знания глаголы (например, «полагать» и «знать») относятся к объектам, которые могут быть целыми фразами. Эти глаголы пронизывают иерархическую графическую структуру, в которой целые графы можно интерпретировать как узлы графов высшего порядка.

Фразу «Жак посылает книгу Мари» изображают классическим концептуальным графом в отличие от

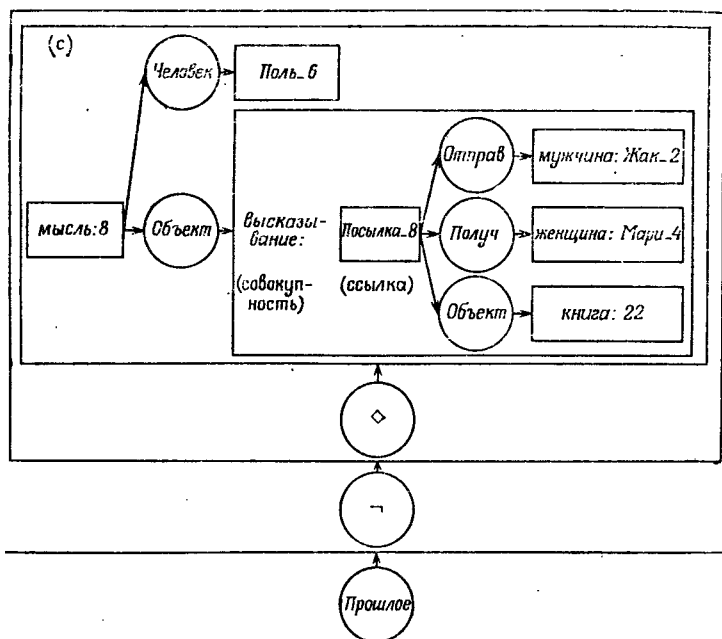


Рис. 3.13. Концептуальный граф логики знания.

фразы «Поль полагает, что Жак посылает книгу Мари», отраженной (с пометкой (с)) в центральной части графа на рис. 3.13. Из этого рисунка видно, что *Мысль_8* принадлежит типу *мысль*. Объект *Мысли_8* — концепт типа (точнее, с «меткой типа») *высказывание*. Ссылка *Мысли_8* — граф фразы «Жак посылает книгу Мари». Полагающий это человек — *Поль_6*.

Любую формулу модальной логики можно рассматривать в духе этого примера. В модальной логике всегда можно разделить пропозициональную часть фразы (выразимую в логике предикатов) от собственно модальности. При изображении концептуальных графов пропозициональная часть поля ссылки концепта обозначена типом *высказывание*. Такие модальности, как \Diamond , \Box , *полагает*, *знает*, суть концеп-

туальные отношения графа. На рис. 3.13 фраза «Не могло быть такого, чтобы Поль полагал, будто Жак посылает книгу Мари» представлена в виде концептуального графа.

3.2.11. Канонические графы

В концептуальном графе есть узлы-концепты и узлы-связи (связывающие узлы). Каждая входящая в связывающий узел стрелка (или выходящая из него) соединена с узлом-концептом. Однако некоторые комбинации узлов бессмысленны. Например, можно изобразить графом фразу «удалить последнее слово следующей строки» (какого-то текста). С позиций грамматики эта фраза синтаксически и семантически корректна. Напротив, фраза «удалить последнюю строку следующего слова» синтаксически корректна, но абсурдна (семантически некорректна). Чтобы исключить графы нереальных (невозможных) ситуаций области рассуждений, Сова [102] определил *канонические* (семантически корректные) *графы* разрешенных комбинаций слов. Каждый составляет представление о мире, выразимом каноническими графами, исходя из личного опыта.

Таким образом, возможный путь построения таких графов следующий: мозг вырабатывает систему концептов, базируясь на поступающих ощущениях, и так расставляет сформированные концепты, чтобы они отражали реальную ситуацию. Следовательно, можно считать, что мозг строит канонические графы, представляющие, вообще говоря, очевидности (другими словами, тавтологии исчисления предикатов). Новые канонические графы строятся из имеющихся по определенным правилам построения. Эти правила формируют порождающую грамматику для концептуальных графов точно так же, как это делается посредством правил переписывания Хомского в случае порождающих грамматик для синтаксических структур естественных языков (см. разд. 5.2 и [112]).

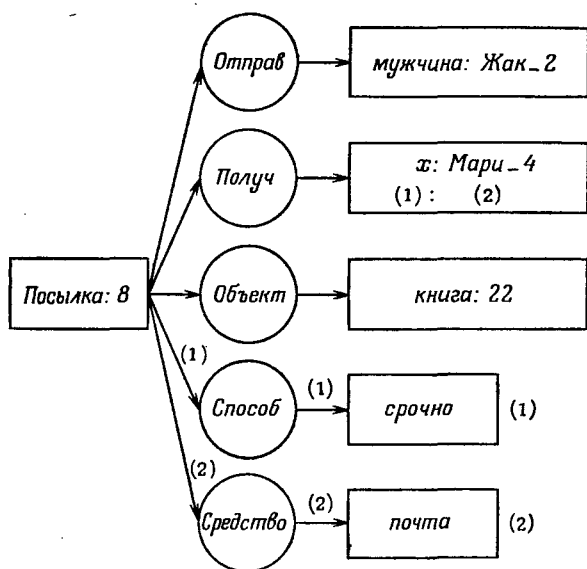


Рис. 3.14. «Жак срочно посылает книгу кому-то».

3.2.12. Правила построения

Сова [102] предложил четыре правила построения для конструирования нового канонического графа g из имеющихся графов g_1 и g_2 (которые могут совпадать):

Правило конъюнкции: определено в § 3.2.5.

Правило упрощения: определено в § 3.2.5.

Правило копирования: граф g есть копия графа g_1 .

Правило ограничения: Для любого концепта с концептуального графа g тип(c) можно заменить неким подтипом. Если c — концепт совокупности, то ее ссылку можно заменить индивидуальным указателем (какой-то конкретизацией). Такие замены разрешены лишь тогда, когда ссылка для c соответствует типу(c).

Проиллюстрируем эти правила на концептуальных графах, представленных рис. 3.14. Граф без элементов с (2) — для фразы «Жак срочно посылает книгу кому-то». Граф без элементов с (1) — для фразы

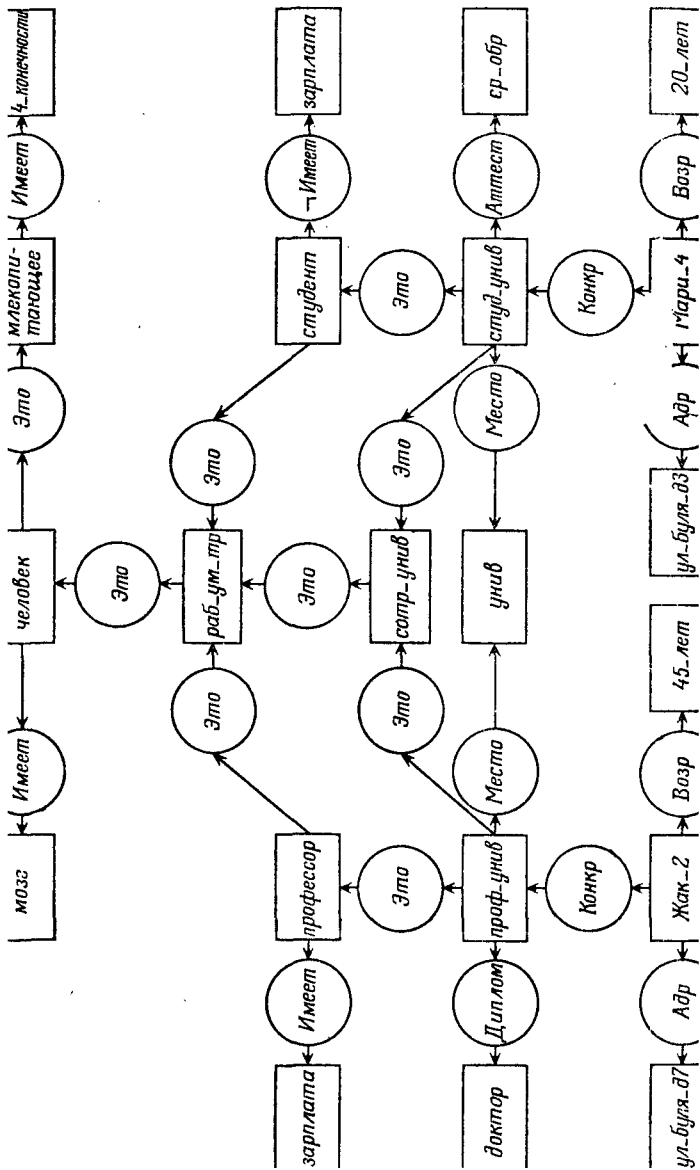
«Жак посылает почтой книгу Мари». Применение правила конъюнкции к этим двум графам дает весь граф, изображенный на рис. 3.14. Можно было бы далее применить к последнему графу правило ограничения, заменяя обозначенную x совокупность «кому-то» меньшей совокупностью «женщина». Однако прежде надо убедиться в свойстве «Мари — женщина». Предыдущие правила являются фактически *правилами сужения*: ограничение сужает концепты, а конъюнкция добавляет условия на графах. Аналогично можно определить правила расширения, которые представляют собой правила, обратные к правилам сужения (см. [102]). Вникать в подробности мы не будем.

3.2.13. Унаследованные свойства

Форма различных концептов, типов и множеств во многих приложениях иерархическая. Классификация животных по родам, видам, семействам, отрядам и т. д. — классический пример иерархии. Другой пример — множество слов в словаре. Книги, транспортные средства и постройки образуют иерархически упорядоченные множества. Можно иерархически классифицировать и абстрактные концепты действий, состояний, свойств, убеждений. Используемые в обычных рассуждениях классификации, как правило, сложнее биологических: индивид принадлежит, вообще говоря, не одному типу и не одному множеству. Однако в большинстве случаев для ИИ используются сравнительно простые иерархии.

Сначала проиллюстрируем иерархию типов на примере, а затем дадим формальное определение. Рассмотрим иерархию, изображенную на рис. 3.15. Если известно, что Жак (представленный конкретизацией Жак_2) — профессор университета, то можно сделать заключение о наличии у него докторской степени и о том, что он работает в университете. Информация «Жак — профессор университета» предоставила нам целую цепочку сведений о Жаке. *Унаследованными*¹⁾ называются такие свойства, которые

¹⁾ Или иначе — наследственными. — Прим. перев.



можно вывести подобным образом. Особых причин детально останавливаться на них нет. Ясно, что кое-какие характеристики профессоров университета можно описать, исходя из принадлежности к профессуре; некоторые характеристики профессуры выводятся из того факта, что она содержится в совокупности работников умственного труда, и, в конечном счете, в совокупности всех людей. Соображения эффективности побуждают нас не связывать с Жаком все общие свойства профессоров университета, вообще профессоров и, наконец, всех людей. Весьма общее свойство, как, например, «У Жака две руки и две ноги», сопоставляется узлу, характеризующему всех людей. Это дает возможность не провозглашать наличие указанного свойства отдельно для каждого индивида.

3.2.14. Решетки типов; иерархии типов

Рассматривая рис. 3.15, замечаем, что *иерархия типов* определяет частичный порядок на множестве меток (ярлыков) типов. Обозначим его символом \leq . Пусть t_1 и t_2 — метки типов. Если $t_1 \leq t_2$, то t_1 — *подтип* типа t_2 , а t_2 — *надтип* типа t_1 . Типы «профессор» и «студент» имеют много общих надтипов: «работник умственного труда», «человек», «высшее млекопитающее». В графе, изображенном на рис. 3.15, «работник умственного труда» — наименьший общий надтип для «профессора» и «студента». Типы «профессор» и «сотрудник университета» имеют общий подтип «профессор университета». В графе, приведенном на рис. 3.15, «профессор» и «студент» не имеют общего подтипа.

Для преобразования иерархической формы на рис. 3.15 в решетку (§ 1.1.9) надо ввести две особые метки типов соответственно в высшей и низшей точках иерархии — *универсальный тип* U (надтип всех типов) и *абсурдный тип* A (подтип всех типов). Иерархия типов даст *решетку типов* со всеми свойствами решетки:

- *любая пара меток t_1 и t_2 имеет наименьший общий надтип,*

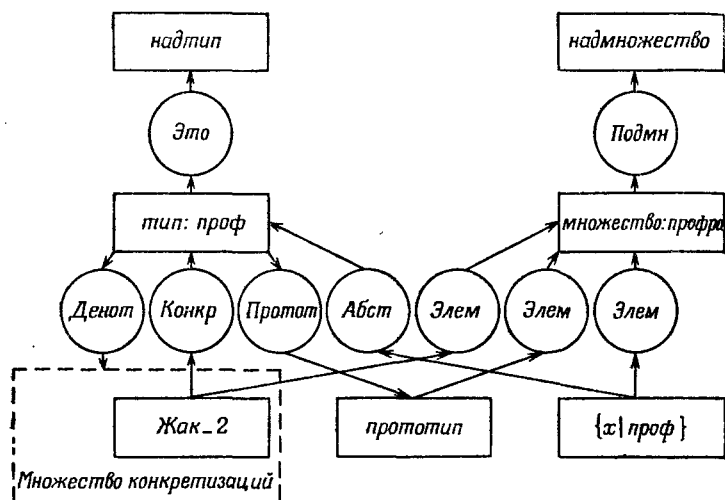


Рис. 3.16. Связывающие узлы и узлы-концепты.

- любая пара меток t_1 и t_2 имеет наибольший общий подтип,
- для любого типа t выполняется соотношение $A \leq t \leq U$.

Данное отношение порядка представляется функциональным отношением *Это*. *Конкр* означает принадлежность индивида типу (§ 3.2.6).

3.2.15. Решетки множеств и решетки типов

Можно также ввести отношение порядка для множеств (подмножество множества) с помощью предиката *Подмн* или связывающего узла с тем же именем. При одноэлементном подмножестве *Подмн* заменяется на *Элем* (§ 3.2.6). Иерархия типов позволила упорядоченно классифицировать свойства типов, а иерархия множеств — свойства множеств. Рис. 3.16 иллюстрирует различие составных частей двух иерархий. Введем на концептуальных графах (в духе классики) несколько функций и операторов, действующих над множеством типов. Прежде всего —

функция *тип*. Она отображает множество концептов на множество T , элементы которого называются *метками* (*ярлыками*) *типа*. Концепты c_1 и c_2 имеют один тип, если $\text{тип}(c_1) = \text{тип}(c_2)$. *Денотатом* типа t называется множество всех тех сущностей, которые являются конкретизациями некоего концепта типа t . Оператор, сопоставляющий типу t его денотат, обозначается через *Денот*. Тип *профессор_университета* есть подтип *профессора*. Следовательно, денотат *Денот(профессор_университета)* является подмножеством *Денот(профессор)*.

Каждой конкретизации (например, *Жак_2*) сопоставлено два вида узлов: узел описания типа конкретизации и узел описания множества, которому принадлежит эта конкретизация. Таким образом, имеем три вида узлов: узел, представляющий индивида (конкретизацию), который через промежуточные связывающие узлы соединен с узлами типа (*Конкр*) и множества (*Элем*). Естественно ввести четвертый вид: $\{x|\text{тип}\}$ — для произвольного индивида (x) определенного типа (*тип*).

Профессор x (о котором больше ничего не известно) обозначен $\{x|\text{проф}\}$. Этот узел соединен с узлом, представляющим множество профессоров, через связывающий узел *Элем*. С другой стороны, он соединен с узлом, представляющим тип профессора, через связывающий узел *Абст*. Оператор *Абст* определяет тип (абстрактный концепт) по описанию какого-либо элемента этого типа. Наконец, можно определить оператор «прототип_для», который можно считать приблизительно обратным оператору *Абст*. Оператор «прототип_для» (*Протот*) представляется связывающим узлом, который соединяет тип с описанием этого типа, даваемым «прототипом». Последний позволяет сжато описать мир, разделенный на типы объектов. Описание, даваемое прототипом, можно рассматривать как описание некоторого типичного представителя множества. Следовательно, прототип — разновидность мифической константы. Узел *прототип* отличается от узла $\{x|\text{тип}\}$, который доставляет множество общих свойств объектов одного типа или одного множества. В этом смысле операторы

Абст и *Протот* не являются взаимно обратными. Операторы и объекты, о которых шла речь, схематически изображены на рис. 3.16 (связывающими узлами и узлами-концептами).

Сова [102] доказал, что решетки типов не изоморфны решеткам множеств. Естественное соответствие между ними ни инъективно, ни сюръективно. Рассмотрим иерархию, изображенную на рис. 3.15. С абстрактным концептом «профессора университета» связано свойство обладания докторской степенью. С другой стороны, вполне вероятно, что каждый индивид из множества «профессоров университета» имеет докторскую степень. Следовательно, соответствующие иерархии типов и множеств, приведенные на рис. 3.15, изоморфны.

3.2.16. Определение типа посредством рода и различия

Фраза «Жак посылает книгу Мари» представлена графом на рис. 3.5 и логической формулой (3.5). Граф на рис. 3.10 представляет ту же фразу в обозначениях вида *совокупность* — *ссылка*; соответствующая запись ее логической формулой такова:

Отправитель (посылка : 8, мужчина : Жак_2) \wedge
Получатель (посылка : 8, женщина : Мари_4) \wedge (3.6)
Объект (посылка : 8, книга : 22).

В таком представлении каждое значение аргумента *типизировано*, т. е. ему приписана некая *метка типа*. Исходя из этого представления, введем некую канву для определения типов. Определение типа должно задавать множество необходимых и достаточных условий, характеризующих тот факт, что рассматриваемый объект является объектом типа.

Можно определить метки типов, используя так называемый аристотелев подход, — через *род* (genus) в *различие* (differentia). Тип определяется исходным типом *род* и высказыванием, называемым *различие* и отделяющим новый тип от исходного. Например, *Посылка* — «событие (род), которое происходит, когда два человека, отправитель и получатель, обеспечи-

тип *ресторан(x)* есть

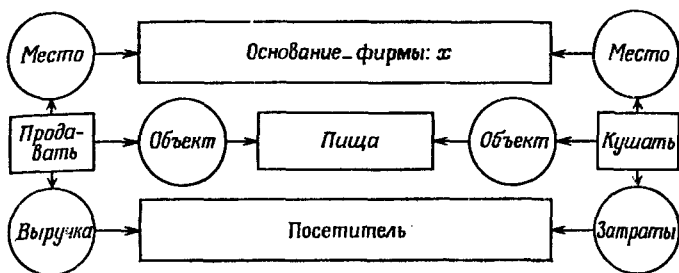


Рис. 3.17. Определение типа для ресторана.

вают перемещение предмета по почте (различие)». Формально:

тип *Посылка(x)* есть

$\text{Конкр}(x, \text{событие}) \wedge$ (род)

$\text{Отправитель}(x, \text{Человек_1}) \wedge$

$\text{Получатель}(x, \text{Человек_2}) \wedge$

$\text{Объект}(x, \text{Письмо}) \wedge$

$\text{Способ}(x, \text{Почта}) \wedge$

$\text{Конкр}(\text{Человек_1}, \text{человек}) \wedge$

$\text{Конкр}(\text{Человек_2}, \text{человек}) \wedge$

$\neg \text{Равно}(\text{Человек_1}, \text{Человек_2})$

(различие)

Определение типов можно формализовать с помощью λ -выражений (§ 3.1.18). Пусть t — метка типа, λxF — λ -выражение. Тогда *тип $t(x)$ есть F* , где тело F этого λ -выражения является различием метки t , а $t(x)$ — ее родом. На рис. 3.17 приведено определение типа ресторана [102] с использованием концептуального графа.

3.2.17. Прототипы

Определить какой-либо тип можно еще путем демонстрации нескольких примеров индивидов «некоего типа» и утверждая, что все, чем похожи индивиды, относится к обсуждаемому типу. Прототип — это конкретизация типа или, иначе, типовая конкретизация (§ 3.2.15). Метод прототипов описывает скорее

типичного индивида некоего класса, чем произвольного представителя этого класса с набором характерных свойств.

Прототип указывает свойства, истинные в типичном, но не обязательно в каждом отдельном случае. Он позволяет гибко определять класс объектов. Прототип содержит фиксированную часть (соответствующую тому общему, что есть у всех конкретизаций этого класса) и переменную часть (обязательную или нет, свою для каждой конкретизации). Формальное определение прототипа в терминах объектных представлений дано в разд. 3.3. Сова [102] определил прототип, используя λ -выражения (§ 3.2.18).

3.2.18. Схемы и схематические кластеры

Концептуальный граф служит для представления знаний. Его можно также использовать для рассуждений и вычислений. Для этого введем понятие схемы, увязав его с правдоподобными и имеющими смысл рассуждениями. *Схему* можно определить, вводя последовательно ограничения на концептуальный граф.

- В произвольном концептуальном графе не накладываются никакие ограничения на расстановку узлов.
- Канонические графы имеют семантические ограничения, представляя нечто «семантически корректное» или «понятное».
- Схемы включают «специфические знания» об области рассуждений (экспертизы), представляя всё правдоподобное.

Понятие схемы подразумевает не только определение, но и способ применения. Для иллюстрации этого аспекта сначала сравним определения схемы и типа. Формальное определение схемы дадим потом. Каждый концепт имеет ровно одно определение его типа, дающее необходимые и достаточные условия принадлежности конкретизаций определенному типу.

Каждому типу можно сопоставить несколько схем, каждая из которых будет представлять один из способов применения концепта данного типа. Это приво-

дит к понятию «множества схем» как возможного источника информации, эквивалентной определению типа. А сам концепт можно определить «множеством его возможных применений».

Для формализации описания концепта Патнем [85] ввел понятие *кластера* или *набора схем*. Каждая из них указывает способ применения данного концепта. Набор всех возможностей применения типа называется его *схематическим кластером*.

Каждую схему можно формализовать с помощью λ -выражений, действуя так, как в § 3.2.16. Пусть λxF — представляющее схему λ -выражение. Формальный параметр x будет представлять концепт того типа, который требуется определить, а тело F укажет один из способов применения этого типа. Здесь F — логическая формула или представляющий ее канонический граф.

Определим схематические кластеры с помощью λ -выражений. *Схематический кластер для типа t есть множество $\{\lambda x_1 F_1, \dots, \lambda x_p F_p\}$ λ -выражений, где каждый формальный параметр x_i принадлежит типу t и каждое выражение $\lambda x_i F_i$ — схема для типа t .*

В отличие от определений типа, выражающих необходимые условия, схема не обязательно истинна при всяком применении. При заданном схематическом кластере соответствующий концепт применим, если хотя бы одна схема из этого кластера окажется истинной для этого применения.

Теперь сравним определения понятия схемы и прототипа. Схемы показывают типичные способы использования концептов. Они не описывают типичных конкретизаций для этих концептов. Напротив, прототип — это типичная конкретизация (§ 3.2.17). Понятия схемы и схематического кластера позволяют более формализованно определить прототип.

Прототип p для типа t есть λ -выражение λxF со следующими свойствами:

- Формальный параметр x принадлежит типу t .
- Прототип p получается сочетанием (по правилу конъюнкции) одной или нескольких схем из схематических кластеров для t и ограничением

(по правилу ограничения) части или всех концептов этих схем только концептами-совокупностями (без индивидов).

При подходящих реализациях схемы соответствуют созвездиям [12], фреймам (кадрам) [74] и сценариям (предписаниям) [98]. Различные примеры схем для иллюстрации этих соответствий даны в разд. 3.3, где вводится объектное представление (в частности, фреймы). Примеры рассуждений с умолчаниями, использующих понятие фрейма, будут приведены в § 3.3.12.

3.2.19. Рассуждения, использующие семантические сети

В книге Совы [102] можно найти строгое описание графовых эквивалентов для правил вывода, таких, как *modus ropens* и правило обобщения. Ограничимся одним примером рассуждения, относящегося к данным об иерархически организованных фактах. Таким структурам присущи наследственные свойства.

Одна из ключевых проблем организации памяти — эффективное управление данными и запросами. При иерархической организации индивидуальные свойства связаны с именами индивидов. Например, на рис. 3.15 индивидуальные свойства Жака (возраст, адрес, семейное положение и т. д.) связаны с именем *Жак_2*. Кроме того, наследственное свойство связывает Жака с (абстрактным) концептом профессора университета. Выходящие из узла *Жак_2* стрелки представимы набором (конъюнкцией) бинарных предикатов:

$$\begin{aligned} & \text{Возр}(\text{Жак_2}, 45_лет) \wedge \\ & \text{Адрес}(\text{Жак_2}, \text{ул_буля_7}) \wedge \\ & \text{Сем_пол}(\text{Жак_2}, \text{холост}) \wedge \\ & \text{Конкр}(\text{Жак_2}, \text{проф_унив}). \end{aligned} \quad (3.7)$$

Эти предикаты указывают, что Жаку 45 лет, что он проживает по улице Буля дом 7, холост и принадлежит типу профессоров университета.

Различные имена предикатов, связанные с узлами-индивидами, образуют первое множество гипотез (некой теоремы, какой — уточним потом). Например, вопрос об адресе Жака выражается предикатом *Адрес(Жак_2, x)*. Этот предикат составит цель (заключение) теоремы, достоверные гипотезы которой — названные выше свойства Жака. Доказательство сводится к поиску конкретизации для *x*, позволяющей вывести названное заключение из гипотез. Проверка показывает, что таковой является *ул_буля_7*.

Иначе обстоит дело с вопросом: «Какой диплом у Жака?», ибо мы не находим гипотезы вида

Диплом(Жак_2, ...).

Но если воспользоваться связью *Конкр*, можно получить подходящую гипотезу. Связанная с узлом *проф_унив* часть графа представима логической формулой:

$$\begin{aligned} \text{Конкр}(x, \text{проф_унив}) \supset & (\text{Диплом}(x, \text{доктор}) \wedge \\ & \text{Место}(x, \text{унив}) \wedge \\ & \text{Это}(x, \text{проф}) \wedge \\ & \text{Это}(x, \text{согр_унив})) \end{aligned} \quad (3.8)$$

Если добавить эту логическую формулу к множеству (индивидуальных) гипотез, то можно, используя обычную технику доказательства теорем, вывести, что Жак имеет докторскую степень. Таким образом, новое множество гипотез и поставленный вопрос составляют формулировку логической теоремы. Исходя из этой формулировки и применяя обычный механизм доказательства, получаем ответ на вопрос.

Механизм доказательства теорем в иерархической среде можно смоделировать следующим образом. Вопрос (составляющий заключение или цель доказываемой теоремы) имеет следующую общую форму.

По-русски: Каково значение некоего индивида по отношению к некоторому свойству?

Логически: *Свойство_i(индивид_j, x)*.

Ожидаемый ответ — конкретизация, представляющая некоторое значение для *x*.

Сначала попытаемся доказать эту теорему, пользуясь лишь гипотезами, прямо связанными с узлом *индивид_j* в соответствующем графе. Эти гипотезы имеют следующую форму.

По-русски: Значение некоего индивида по отношению к некоторому свойству есть ...

Логически: *Свойство_i* (*индивид_j*, *значение_i*).

Если так доказать не удастся, используем связь *Конкр*, исходящую из узла, представляющего индивида и указывающего на новое множество гипотез, которое имеет следующий общий вид.

По-русски: Если индивид принадлежит типу *t*, то он имеет дополнительное множество значений по отношению к дополнительному множеству свойств.

Логически: $\forall x [\text{Конкр}(x, \text{тип}_t) \supset \bigwedge_i \text{Свойство}_i(x, \text{значение}_i)]$.

Из этой логической формулы и предиката *Конкр* (*индивид_j*, *тип_t*) выводим новое множество свойств относительно данного индивида. Если эти новые гипотезы позволяют доказать нашу теорему, то доказательство закончено. Если же нет, то связь *Это* укажет на тип повыше, который в свою очередь даст новое множество гипотез. Эта процедура с последовательными множествами гипотез заведомо эффективнее общего метода доказательства теорем, не использующего иерархической структуры данных. Процедура вывода схематически изображена на рис. 3.18.

3.2.20. Заключение

Основанное на семантических сетях представление знаний может заменить представление, базирующееся на логике предикатов. Мы доказали, что в отличие от логического представления сетевое дает сгруппированную вокруг конкретизаций информацию. Кроме того, видение этой информации более глобальное. Се-

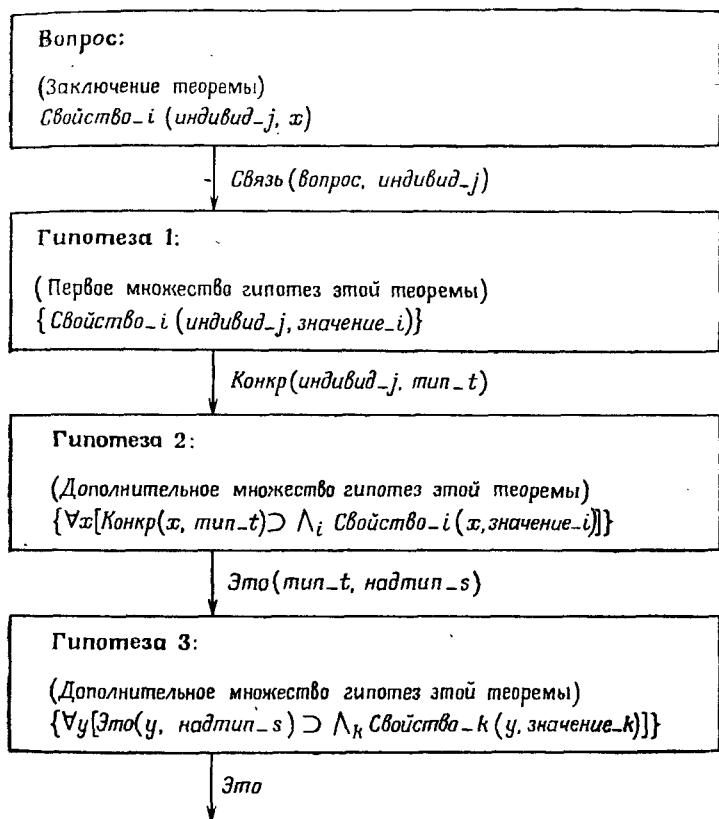


Рис. 3.18. Схема процедуры вывода.

мантические сети принадлежат к моделям, используемым специалистами по БД. Они отражают структуру некой области и реализацию этой структуры на ЭВМ.

В наше время семантические сети используются в системах распознавания естественных языков. Они вмонтированы также в более сложные формализмы, где служат для поддержки описания и классификации сущностей (экспертные системы). Достоинство концептуальных графов — в возможности прямого обобщения на модальные системы и другие формализмы, трудно представимые логикой первого порядка.

Семантические сети могут сосуществовать с другими обозначениями, а также служить для промежуточного представления. Графические обозначения широко используются в качестве промежуточных между естественным языком и каким-нибудь языком программирования.

3.3. Объектное представление

3.3.1. Введение

Из § 3.2.17 известно, что класс объектов может определяться одним типичным объектом. Объектное представление можно получить как из логического, так и из сетевого. При объектном представлении собирают из логических формул, содержащих одни и те же конкретизации, более крупные структуры, называемые *фреймами*. Последние строятся вокруг конкретизаций области рассуждения. Если нужен доступ к информации по одному из этих объектов, то обращаются к соответствующему фрейму и уже внутри блока находят свойства и факты относительно рассматриваемого объекта.

Бинарная версия исчисления предикатов (§§ 3.1.4, 3.1.6) привела к графическим обозначениям через семантические сети (см. разд. 3.2). Бинарный предикат можно представить тройкой вида

(Объект, атрибут_{*j*}, значение_{*j*}).

Собрав все тройки с данным объектом, получим *объектное представление* области рассуждений относительно этого объекта. Общая форма этого представления такова:

Объект, (атрибут_{*j*}, значение_{*j*}), ($j = 1, \dots, m$).

Значит, вместо построения различных независимых формул строим некую структуру с полной информацией об объекте (§ 3.2.3).

Эти объектные обозначения приняты в новом семействе языков программирования, особенно хорошо приспособленных к выражению знаний в ИИ — *объектно-ориентированных языков* (§ 3.3.7).

3.3.2. Сцепки

Предположим, что мы хотим представить фразы из примера § 3.2.5, т. е. фразы (в компактной форме) «Жак пишет книгу, посылает ее Мари, которая ее читает». В БД с этими фразами использовались конкретизации *Жак_2*, *Мари_4*, *Посылка_8* и *Книга_22* для ссылок в объектном языке на имена концептов метаязыка, упомянутые в этих фразах. Если расширить БД, то добавятся новые концепты и дополнительная информация о них.

Для использования знаний полезно собрать все факты о данном концепте в одно множество, называемое *сцепкой*¹⁾ (по-английски *unit*). В нашем элементарном примере сцепкам *Жак_2*, *Мари_4* и *Книга_22* будут соответствовать логические формулы:

Жак_2

Пишет (Жак_2, Книга_22)

Посылает (Жак_2, Мари_4, Книга_22)

Мари_4

Посылает (Жак_2, Мари_4, Книга_22)

Читает (Мари_4, Книга_22)

Книга_22

Пишет (Жак_2, Книга_22)

Посылает (Жак_2, Мари_4, Книга_22)

Читает (Мари_4, Книга_22)

3.3.3. Фреймы и слоты

Если выразить эти фразы бинарными предикатами то сцепки будут называться *фреймами*. Из § 3.1.6 известно, что тернарный предикат

Посылает(Жак_2, Мари_4, Книга_22)

преобразуется в произведение бинарных предикатов

Отправитель (Посылает, Жак_2) ∧

Получатель (Посылает, Мари_4) ∧

Объект (Посылает, Книга_22).

¹⁾ Другие названия — *единица*, *секция*. — Прим. перев.

Концепту «посылает» соответствует следующий фрейм:

ФРЕЙМ

Посылает		(объект)
Отправитель	Жак_2	(слот-1)
Получатель	Мари_4	(слот-2)
Объект	Книга_22	(слот-3)
(атрибуты или имена слотов)	(значения или значения слотов)	

Каждая пара (атрибут, значение) фрейма называется слотом или (имя-слота, значение-слота). Сам фрейм по-английски — *slot-and-filler notation*. В этих обозначениях различные слоты сгруппированы вокруг объекта, охарактеризованного фреймом.

В обозначениях *slot-assertion* (§ 3.1.6) соответствующие бинарным предикатам слоты используются как изолированные сущности без группировки в охватывающем их фрейме.

3.3.4. Явные фреймы

В § 3.1.7 показано, что часто бывает полезно представление знаний с явным указанием всех ссылок. Именно поэтому мы постулировали в нашем примере существование вполне определенной *Посылки_8*. Фраза «Жак посылает книгу Мари» бинарными предикатами представляется так:

Посылка_8

Элем	посылки
Отправитель	Жак_2
Получатель	Мари_4
Объект	Книга_22

Следовательно, в процессе выявления ссылок можно не только дать явные значения аргументов и имена предикатов, но также имена представляющих высказывания логических формул. Например, *Посылка_8* — имя высказывания *Посылает(Жак_2, Мари_4, Книга_22)*. Такой формализм называется явным фреймом (по-английски *case-frame*).

3.3.5. Функциональные фреймы

В § 3.1.8 показано, что представление бинарными предикатами легко можно выразить в функциональной форме. Отношения между *Посылка_8* и первоначальными аргументами для *посылает* можно выразить функциями на множестве *посылок*. Введем следующие функциональные обозначения для фреймов:

Посылка_8

<i>элемент</i>	: (элемент_из посылок)
<i>отправитель</i>	: Жак_2
<i>получатель</i>	: Мари_4
<i>объект</i>	: Книга_22

Форма «элемент-из» в слоте имеет имя «элемент» для указания того, что описанный фреймом объект является элементом некоторого множества (в нашем примере это множество *посылок*). Определенный таким способом фрейм называется *функциональным*.

3.3.6. V-квантификация

Посмотрим, как можно осуществлять V-квантификацию переменных при оперировании с фреймами. Фраза «Жак посылает книгу каждой женщине» (§ 3.2.9 и рис. 3.11) записывается бинарными предикатами в следующем виде:

$$\forall x \exists y \exists z [\text{Отправитель}(z, \text{Жак_2}) \wedge \text{Получатель}(z, x) \wedge \text{Объект}(z, y) \wedge \text{Элем}(z, \text{посылки}) \wedge \text{Конкр}(y, \text{книга}) \wedge \text{Конкр}(x, \text{женщина})].$$

Множество представленных переменной *z* *посылок* описано функциональным фреймом:

z (*x*)

<i>элемент</i>	: (элемент_из посылок)
<i>отправитель</i>	: Жак_2
<i>получатель</i>	: <i>x</i>
<i>объект</i>	: <i>y</i> (<i>x</i>)

Этот фрейм целиком образует область действия квантора общности. Важно отметить, что все представленные фреймами логические формулы записаны в

предваренной сколемовской форме (§§ 1.2.7—1.2.8): отрицания выражены явно, переменные стандартизованы по отдельности, переменные, которые были связаны кванторами существования, замещены и кванторы общности применяются ко всей формуле.

3.3.7. Рассуждения, использующие объектное представление

Объектные представления были введены фактически по прагматическим причинам. Эти представления часто связаны с вполне определенными классами проблем. *Объектно-ориентированные языки* (известнейший из них Смолток) характеризуются взаимопроникновением структур данных и процедур. Связанные с этими представлениями законы вывода, вообще говоря, лишены формальной строгости правил вывода логики предикатов. Концепция объектных представлений больше направлялась заботой об эффективности исчисления, чем стремлением к его полноте. Очевидно, можно было бы развить для них системы выводов, эквивалентные использованным в логике предикатов. Такой синтаксис был бы весьма громоздок, и его применение лишило бы объектное представление его прагматических достоинств. Вместо того чтобы покоиться на полном множестве законов дедукции, различные используемые в ИИ системы используют сокращенные версии классических законов вывода. Попозже мы дадим несколько примеров.

3.3.8. Паросочетание

Определить (аналогичную унификации в логике предикатов) *операцию паросочетания* (по-английски *matching operation*) настоятельно необходимо, если мы хотим использовать объектное представление в качестве БД для запросной системы (query system). До задания свойств операции паросочетания напомним, что объектное представление — альтернатива логическому формализму.

Нильсон [83] предложил следующее определение.

Два объектных представления паросочетательны тогда и только тогда, когда логическую формулу для

первого можно унифицировать с логической формулой для второго. Слово «унифицировать» означает здесь, что существуют подстановки для переменных, делающие логические формулы идентичными. См. § 1.2.13.

Если наша цель — построить «язык запросов» (query language) для извлечения информации из БД, то нужно более ограниченное определение паросочетания. Операция паросочетания вообще не должна быть симметричной в том смысле, что она связывает заключение (объект-цель) с гипотезами (объектами-фактами). Объект-цель «паросочетается» с объектом-фактом, если влекущая объект-цель логическая формула унифицируема с одним из сомножителей объекта-факта (представленного конъюнкцией гипотез и аксиом).

В терминах доказательства теорем это означает, что операция паросочетания может осуществляться тогда и только тогда, когда цель можно доказать из гипотез и аксиом (фактов).

Рассмотрим *фрейм-факт* (гипотезу) из § 3.3.5:

Посылка_8

<i>элемент</i>	: (<i>элемент_из посылок</i>)
<i>отправитель</i>	: <i>Жак_2</i>
<i>получатель</i>	: <i>Мари_4</i>
<i>объект</i>	: <i>Книга_22</i>

Можно установить паросочетание этого фрейма-факта с фреймом-целью из § 3.3.6:

z(y)

<i>элемент</i>	: (<i>элемент_из посылок</i>)
<i>отправитель</i>	: <i>Жак_2</i>
<i>получатель</i>	: <i>x</i>
<i>объект</i>	: <i>y(x)</i>

Фрейм-цель интерпретируется как вопрос: «Кому Жак послал книгу?» Паросочетание фрейма-цели и фрейма-факта приводит к подстановке $\{(z, \text{Посылка_8}), (x, \text{Мари_4}), (y, \text{Книга_22})\}$, которую можно интерпретировать как ответ на вопрос (определение подстановки см. в § 1.2.6).

3.3.9. Функциональные атрибуты

Атрибуты описывают множество характеристик, связанных с функциями функционального фрейма. Атрибут не обязательно является конкретизацией некоторых данных вроде *Жак_2*; с тем же успехом он может быть функциональным выражением. Рассмотрим фразу «Жак посылает книгу Мари, а Поль получает письмо от человека, которому Жак послал книгу». Ее первая часть выражается фреймом из § 3.3.5:

Посылка_8

<i>элемент</i>	: (<i>элемент_из посылок</i>)
<i>отправитель</i>	: <i>Жак_2</i>
<i>получатель</i>	: <i>Мари_4</i>
<i>объект</i>	: <i>Книга_22</i>

Вторая часть представляется фреймом, для которого некое функциональное выражение является атрибутом «отправителя»:

Посылка_9

<i>элемент</i>	: (<i>элемент_из посылок</i>)
<i>отправитель</i>	: <i>получатель (Посылка_8)</i>
<i>получатель</i>	: <i>Поль_6</i>
<i>объект</i>	: <i>Письмо_3</i>

Мари_4 и *получатель (Посылка_8)* — два разных способа представления одного и того же лица.

Если пытаемся установить паросочетание между фреймами, содержащими функциональные выражения в качестве атрибутов, то следует предположить, что эти выражения получают значения сразу, как только это будет возможно. Присваивание значений осуществляется посредством ссылки на объект, указанный аргументом этой функции.

Предположим, что поставлен вопрос: «Посылала ли Мари что-нибудь кому-нибудь?», представимый фреймом-целью:

z(x)

<i>элемент</i>	: (<i>элемент_из посылок</i>)
<i>отправитель</i>	: <i>Мари_4</i>
<i>получатель</i>	: <i>x</i>
<i>объект</i>	: <i>y</i>

Множество фреймов-фактов содержит *Посылку-8* и *Посылку-9*. Так как *получатель(Посылка-8)* может получить значение *Мари-4*, то фрейм *Посылка-9* получит *Мари-4* в качестве значения функции *отправитель*. Ответ на вопрос даст паросочетание фрейма *Посылка-9* с фреймом *z*.

С помощью подстановки $\{(x, \text{Поль-6}), (y, \text{Письмо-3})\}$ можно получить ответ «да, Мари отправила письмо Полю».

3.3.10. Автоматические рассуждения, использующие фреймы

Из § 3.3.9 известно, что атрибуты функций во фреймах не обязательно константы или конкретизации, но могут также быть функциональными выражениями. Сделаем еще шаг и рассмотрим еще одну возможность для атрибутов (представляющих множество связанных с объектами характеристик) — быть процедурами или подпрограммами. Это приводит нас к понятию *объектно-ориентированных языков*, предназначенных для автоматических рассуждений на основе объектного представления.

Язык KRL¹⁾ [8] — объектно-ориентированный, основанный на представлении знаний фреймами. В нем все знания описаны в терминах UNITS-KRL. Ограничимся двумя примерами ([92] с. 397—401):

[Путешествие UNIT Абстрактное
 <SELF. (Событие)>
 <способ (ИЛИ Авиа Авто Поезд)>
 <назначение (Город)>]

[Путешествие UNIT Абстрактное
 <SELF (Взаимодействие)>
 <визитер (Человек)>
 <визит (Множество из (Человек))>]

Первую часть можно интерпретировать следующим образом. «Путешествие — абстрактное понятие, связанное с самолетом, машиной или поездом и имеющее

¹⁾ «Язык представления знаний». — Прим. перев.

некий город пунктом назначения». Вторую часть можно интерпретировать аналогично. KRL материализует некоторый процесс рассуждений, управляемый правилом паросочетания (§§ 3.3.8—3.3.9).

3.3.11. Иерархические рассуждения, использующие фреймы

Операция паросочетания аналогична унификации из логики предикатов. Можно также связать объектное представление с более мощными правилами вывода, такими как теорема дедукции. Нильсон [83] и Сова [102] предложили общие операции дедукции, использующие объектный формализм и базирующиеся на логической импликации и правиле *modus ponens*.

Рассмотрим лишь частный случай одного нередко используемого закона дедукции, в соответствии с которым с помощью импликации приписываются свойства каждому элементу определенного множества. Речь идет о свойстве наследования, уже рассмотренном в § 3.2.13. Займемся иерархией, приведенной на рис. 3.15, где типы «профессор университета» и «профессор» заменены соответствующими множествами. Концептуальный граф, окружающий узел Жак_2, представляется объектно и логически (см. § 3.2.19) следующим образом:

Объектные обозначения

Жак_2

эле : (*эле_из проф_унив*)

возр : *45_лет*

адр : *ул_буля_7*

сем_пол : *холост*

Логические обозначения (Факт)

Эле (*Жак_2, проф_унив*) \wedge

Возр (*Жак_2, 45_лет*) \wedge

Адр (*Жак_2, ул_буля_7*) \wedge

Сем_пол (*Жак_2, холост*)

(3.9)

Мы видели, что вопросы наподобие «Сколько лет Жаку?» или «Кто проживает по улице Буля дом 7?»

представимы фреймом-целью или логическим заключением:

Объектные обозначения	Логические обозначения (Цель)
-----------------------	----------------------------------

● Жак_2

возр : x

Возр (Жак_2, x)

● y

адр : ул_буля_7

Адр (y, ул_буля_7)

Из §§ 3.3.8—3.3.9 мы знаем, как получить ответы на эти вопросы, используя операцию паросочетания (для объектных обозначений) или унификацию (для логических).

Далее рассмотрим вопрос «Какой диплом у Жака?», формализованный следующим образом:

Объектные обозначения	Логические обозначения (Цель)
-----------------------	----------------------------------

Жак_2

дипл : y

Дипл (Жак_2, y)

(3.10)

Нельзя получить немедленного ответа на этот вопрос. Придется обратиться к объектным и логическим обозначениям, описывающим свойства индивида x из множества преподавателей университета (см. рис. 3.15 и § 3.2.19):

Объектные обозначения	Логические обозначения (Правило)
-----------------------	-------------------------------------

{x | проф_унив}

дипл : доктор

место : унив

Элем (x, проф_унив) \supset

(Дипл (x, доктор) \wedge

Место (x, унив)) (3.11)

Осуществим паросочетание и унификацию целей (3.10) с правилами (3.11) для получения новой цели (3.12):

Объектные обозначения	Логические обозначения (Цель)
-----------------------	----------------------------------

Жак_2

элемент :

(элемент_из проф_унив)

Элем (Жак_2,
проф_унив)

(3.12)

Эта новая цель точно соответствует фразе «Жак — профессор университета», что является фактом. Ответ «Жак имеет степень доктора» также найден. Такой вид доказательства отвечает системе прямой дедукции (§ 3.1.22).

Если нельзя ответить на вопрос, рассматривая свойства индивида (*Жак_2*) или содержащего его множества (*проф_унив*), то обратимся к новому множеству (*проф*), подмножеством которого является прежнее. Подмножество профессоров университета связано с множеством профессоров следующим образом:

Объектные обозначения Логические обозначения

проф_унив

элемент: (*подмнож в проф*) *Подмн(проф_унив, проф)*
(3.13)

3.3.12. Рассуждения с умолчаниями

Изучавшиеся до сих пор представления уязвимы для следующего упрека. Они предполагают строгие рассуждения, тогда как не всегда можно опереться на формально точные знания. Используемые экспертами в большинстве областей правила лишь приблизительно точны и не всегда применимы. Многочисленные описания вида «все x имеют свойство P » надо считать лишь приблизительно истинными. Например, можно говорить, что «все птицы летают (за исключением страусов, пингвинов и т. д.)».

Вообще интересно использовать знания, вроде «все птицы летают», и подвергать рассуждения сомнению лишь в некоторых заранее известных исключительных случаях. Это называется *рассуждениями с умолчаниями*. Предложены многочисленные формализации для выражения «все x имеют свойство P , за исключением тех случаев, когда явно указано противное для некоторых конкретизаций переменной x ».

Изложим формализм, основанный на объектном представлении [83, с. 408] и допускающий построение рассуждения с умолчаниями. Рассматриваемый нами подход сохраняет определенную простоту рассужде-

ний. Он состоит в разрешении осуществлять исключения в рассуждениях, проведенных по правилу \forall -квантификации, действующему на формулы с импликацией. Утверждение вида «все преподаватели университета имеют степень доктора» можно поначалу сделать без всяких исключений. Из него выводим, что «Жак имеет степень доктора, если Жак преподает в университете». Если в дальнейшем выяснится, что Жак не имеет степени доктора, то мы аннулируем нашу дедукцию и так изменим универсальное утверждение о преподавателях университета, чтобы Жак оказался исключенным.

Способ, которым операция паросочетания использует наследственное свойство, может дать автоматический механизм обработки исключений такого рода. Управляющий операцией паросочетания механизм может использовать наследственное свойство для вывода некоего свойства объекта, только если специфическая информация об этом свойстве не появляется в соответствующем этому объекту фрейме. Например, мы интересуемся, какой диплом у Жака. Для ответа на вопрос сначала попробуем установить паросочетание фрейма-цели с фреймом-фактом. Если фрейм-факт по Жаку имеет вид

Жак_2

элемент : (элемент_из проф_унив)

дипл : магистр

то паросочетание вовлекает унификацию (*у, магистр*) и ответ таков: *магистр* (Жак имеет диплом *магистра*). Наоборот, если фрейм-факт по Жаку говорит лишь, что он профессор университета:

Жак_2

элемент : (элемент_из проф_унив)

то механизм паросочетания использует фрейм

{x | проф_унив}

дипл : доктор

и ответ — *доктор*.

3.3.13. Заключение

Логика предикатов применима для представления знаний, используемых некоторыми системами ИИ. Однако существуют специфические виды знаний, с трудом представимые языком логики предикатов. Поэтому мы ввели неклассические (модальную и временную) логики и альтернативные (сетевое и объектное) представления.

С другой стороны, мы указали различные методы рассуждений, связанных с логическим, сетевым и объектным представлениями. Кроме того, определили рассуждения с умолчаниями в рамках объектного представления.

Это подвело нас к введению других видов рассуждений (таких как рассуждения здравого смысла, нестрогие рассуждения по поводу предположений и модифицируемые рассуждения), формализация которых дается в гл. 4.

4. Логика и модифицируемые рассуждения

4.1. Многочисленные роли логики

4.1.1. Введение

В гл. 3 введены различные элементарные формы представления знаний: *логическая, сетевая и объектная*. Две последние часто являются альтернативой первой. Было показано, что их можно переписать и переинтерпретировать с помощью логического формализма. Таким образом, мы подошли к вопросу о реальной роли логики в методах представления знаний и рассуждений.

В действительности форма вклада, который математическая логика должна вносить в развитие методов представления знаний и рассуждений, остается предметом нескончаемых дебатов. В этой связи возможны различные, не исключающие друг друга, точки зрения:

- Логика может рассматриваться как средство, хорошо подходящее для представления знаний и рассуждений.
- Логика может рассматриваться как формализм для ссылок.
- Логика может рассматриваться как метод подтверждения рассуждений и семантического анализа представленных знаний.

В параграфах этого раздела будут развиты эти различные точки зрения на роль логики.

4.1.2. Логика как средство для представления знаний и рассуждений

Предмет формальной логики — корректные формы рассуждений. Она предоставляет различные средства формализации и анализа правильности дедуктивных

рассуждений. Эти средства рассмотрены в гл. 1: системы семантической оценки и системы вывода формул.

Методы формализации и обоснования рассуждений были привиты на различные языки, позволяющие представлять знания. Важнейшее свойство этих языков состоит в том, что они предоставляют пользователю строгий синтаксис. Второе свойство заключено в существовании средств связи с семантикой. Это позволяет установить однозначное соответствие между миром и его представлением в языке. Более того, они обеспечивают обоснование выводов, которые можно сделать из представленных знаний.

Таким образом, логическая система будет состоять из языка, формальной семантики и системы вывода¹⁾. Эти логические средства можно применять довольно непосредственно. Логические языки служат опорой для выражения знаний, которые также представлены декларативно в виде логических выражений. Рассуждение определяется как операция доказательства общезначимости или выполнимости логического утверждения. Напомним, что логическая формула общезначима, если все ее интерпретации являются моделями, и выполнима, если допускает хотя бы одну модель (§§ 1.1.6, 1.2.5). Рассуждения осуществляются окольным путем, посредством манипулирования дедуктивными компонентами рассматриваемой логической системы. Например, такими компонентами являются классические аксиоматические системы (§§ 2.1.3, 2.1.8), системы натурального вывода (§§ 2.1.7, 2.1.9) или семантика рассматриваемой логической системы. Так как процесс доказательства должен быть автоматизирован, то это очень часто реализуется методами поиска, причем производится систематическая манипуляция с дедуктивной компонентой.

Этот способ действий ведет к методу *логического программирования* [58], [63], который можно прямо использовать для представления знаний и рассужде-

¹⁾ Это верно для большинства обычных логических систем, но некоторые из названных средств могут отсутствовать у отдельных систем. Например, интенциональные логики Монтегю [76] лишены системы вывода.

ний. Наиболее распространенной системой такого сорта является язык Пролог [19], [18] — соединение метода резолюций (§§ 1.1.12, 1.2.14), языка хорновских дизъюнктов (§ 1.1.6), метода поиска вглубь типа обратного вывода (§ 3.1.23) и использование отрицания для получения противоречия [17].

Подобные системы будут описаны в гл. 5 и 6. Здесь мы ограничимся анализом их пригодности для представления знаний и рассуждений. Он имеет три аспекта: эпистемический, дедуктивный и связанный с алгоритмической эффективностью.

● Эпистемический аспект

Поскольку конечная цель — представление знаний, основным критерием адекватности используемого логического языка является его выразительность. Системы ИИ чаще всего ограничиваются применением логических языков порядков 0 и 1: логики высказываний и логики предикатов. Логика предикатов служит эталоном выразительности для альтернативных систем [44]. Кстати, она достаточно выразительна для решения многих проблем представления знаний в ИИ, но не универсальна. Некоторые знания формализуемы лишь в логических языках высших порядков. Например, квантификация формул не представима в логике порядка 1.

Была отмечена трудность формализации некоторых знаний в обычных логических языках. Типичные примеры — пространственно-временные отношения, для которых лучше приспособлены специфические логики, в частности модальные (§ 3.1.11).

С другой стороны, основные логические системы не оснащены средствами структурирования и агрегирования знаний. Применение систем классификации и типизации объектов через механизм ассоциированного доступа прямо не предусмотрено. Иногда ищется компромисс через соединение чисто логических методов с объектным и сетевым представлениями (разд. 3.2, 3.3). Системы классификации и механизмы доступа объектного формализма используются для иерархического таксономического¹⁾ представления термов, на

¹⁾ Или, иначе, классификационного. — *Прим. перев.*

которые ссылаются логические утверждения (§§ 3.2.19, 3.3.11). Логический язык дополняющим образом используется для представления позитивных знаний об этих терминах. Реализованный в системе KRYPTON [9] союз между логической и сетевой системами опирается не только на эпистемический аспект, но и на аспект эффективности. Чтобы оттенить сказанное, заметим, что недавние применения систем логического программирования (вроде распространенной системы Пролога LOGIN [2]) направлены на внесение в логику систем классификации с наследованием свойств (§ 3.2.13).

● Дедуктивный аспект

Формальная логика связана с формализацией и обоснованием корректных рассуждений. Последние также называются *общезначимыми*: их правильность несомненна при всех интерпретациях. Дедуктивные системы логики специально приспособлены для формализации этого класса рассуждений. А распространяется ли их полезность на другие классы?

Рассуждения, которые желательно моделировать в приложениях ИИ, не все общезначимы. Часто они приблизительны и неопределенны по сути или от неполноты, или неопределенности предпосылок. Выведенные из неопределенных рассуждений заключения должны допускать возможность отказа от них, если предпосылки, приведшие к принятию предположения о возможности этих заключений, больше не подтверждаются или если новая информация блокировала эту дедукцию. Дедуктивные системы логики не позволяют прямо формализовать модифицируемые рассуждения. Симптом этой ограниченности — свойство *монотонности* всех обычных логических систем: множество теорем такой системы лишь растет с увеличением множества основных аксиом. Различные логические системы, формализующие модифицируемые рассуждения, будут введены и подробно описаны в последующих разделах этой главы.

● Аспект эффективности

Эффективность важна во всех практических применениях дедуктивных принципов логики. Особое внимание следует обратить на стратегию проведения преобразований в дедуктивной системе. Ей грозит комбинаторный взрыв пространства поиска, могущего подчас стать бесконечным. Существенное неудобство — неразрешимость логических систем, наблюдаемая с тех пор, как она поразила логику предикатов (§ 2.2.11).

4.1.3. Логика как формализм ссылок

Если можно считать логику средством представления знаний и рассуждений (возможно, в сочетании с более адекватными альтернативными системами), то можно признать за ней и другие роли. Можно интерпретировать логику как формализм ссылок, полезный на разных уровнях в определении других эффективных методов представления. Прежде всего, она может служить их строгому определению. Переписыванием языков и законов в логический формализм можно уточнить семантику этих методов. Логика предикатов широко используется для этого [44], но не всегда дает достаточно выразительную систему. Например, системы множественного наследования с исключениями (§ 3.3.12) позволяют выразить немонотонные формы рассуждений. Их нельзя прямо переписать в логику предикатов, но можно строго определить в более развитых логических языках [110].

Так как многочисленные формализмы можно переписать на логический язык, то последний — эталон выразительности. Альтернативный формализм можно строго определить путем переписывания его языка и законов в рамках логики. Сам он, вроде бы, и не нужен. Однако объективно уступающие логике предикатов методы представления бывают привлекательны приростом некой эффективной выразительности. С Паскаля можно все перевести на язык ассемблера. Тем не менее Паскаль обладает в некоторых отношениях превосходящей эффективной выразительностью. Так же как и некий формализм, сводимый к логике предикатов.

Особенно это относится к сетевым (разд. 3.2) и объектным (разд. 3.3) представлениям. Ранее мы продемонстрировали перевод *m*-арных предикатов в бинарные и их интерпретации в сетевом и объектном представлениях. Таким образом, эти представления «эквивалентны» логическому в смысле возможности взаимного преобразования. Между тем было показано, что сетевое и объектное представления дают «структурный» образ мира, чего логическое представление само не делает.

Логика может также выполнять роль блюстителя логических принципов и правил во всех системах, кроме явно оговоренных исключений. Соблюдение этих принципов в данном формализме можно проверить переписыванием их в логический или простым сравнением с ним.

4.1.4. Неизбежность логики

Иногда утверждают, что некоторые проблемы представления знаний и рассуждений решаемы лишь с помощью логических языков и ассоциированных дедуктивных систем (как бы то ни было с ролью логики как системы ссылок или средства точного определения других формализмов). В частности, благодаря точному определению принципов применения операторов и связок (вроде конъюнкции, дизъюнкции, отрицания, равенства, кванторов существования и общности) логика позволяет задать некоторые часто полезные парадигмы рассуждений [77].

Например, логика предикатов с равенством (§ 2.1.10) дает возможность:

- выразить, что нечто обладает определенным свойством, не указывая, что именно (роль \exists -квантификации),
- выразить, что каждый элемент некоего класса обладает определенным свойством, без указания, что представляет из себя каждый такой элемент (роль \forall -квантификации),
- выразить, что хотя бы одно из двух утверждений истинно, не говоря, какое именно (роль дизъюнкции),

- явно сказать, что нечто ложно (роль отрицания),
- утверждать или оставлять неустановленным тот факт, что два различных выражения означают один и тот же объект (роль равенства).

Даже если бы дело дошло до задумывания нелогического формализма, опирающегося на эти функциональные средства выразимости и дедуктивности, то естественно возник бы вопрос о возможности обретения вновь некой разновидности логики.

Эти парадигмы полезны и подчас необходимы при решении многих проблем (в частности, требующих рассуждений в условиях неполной информации). Вот пример [77]. Три блока *A*, *B* и *C* расположены в линию (рис. 4.1). Известно, что *A* зеленый и *C* синий. Цвет *B* неизвестен. Находится ли зеленый блок рядом с незеленым? Ответ «да»: если *B* зеленый, то он рядом с незеленым *C*. Если *B* незеленый, то он рядом с зеленым *A*.

Согласно Муру, три трудно познаваемых в нелогическом формализме логических фактора вплетены в рассуждения для обретения возможности:

- увидеть истинность \exists -квантифицированного высказывания, не зная, какой объект делает его истинным,
- распознать подтверждаемость некоего высказывания, либо его отрицания,
- рассматривать ряд случаев по отдельности.

4.1.5. Анализ знаний и рассуждений

Иные авторы приписывают логике в основном функции семантического анализа знаний и обоснования выводов [43], [82].

Представить знания — это значит выразить в некотором формализме имеющийся у нас образ мира.

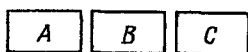


Рис. 4.1. Пример.

Соответствие между миром и его представлением устанавливается семантическим анализом. Такой анализ имеет целью определить объекты представления и уточнить образ мира, определяемый представлением. Следовательно, оно должно позволить осуществлять анализ истинности высказываний о мире. Иначе говоря, для плодотворности представления нужно, чтобы оно могло быть предметом анализа, использующего информацию из этого представления для выявления того, что свойственно миру, а что нет. С этой точки зрения обоснованный вывод или дедукция «подтверждаются» видением мира, который определен семантическим анализом представления.

Сказать, что Р. Рейган все еще жив и является президентом США, — это значит вообразить мир, где истинно, что Р. Рейган жив, но и равным образом ложно, что он мертв и что другой человек является президентом США, если известно, что у власти может находиться лишь один президент.

Семантический анализ представленного в некотором формализме знания должен позволять определить, что в этом воображаемом мире влечет истину, а что — ложь. Даже если анализ облечен другими аспектами, подобная операция относится *по определению* к компетенции логики и делает особенно полезным обращение к теории моделей [43], [77], [82]: «...нелогический анализ знаний — такая же чушь, как программирование без программистов...» [82, с. 17].

Итак, логику можно считать преимущественно средством анализа знаний и рассуждений. В свою очередь это вызывает другие рассматривания.

4.1.6. Заключение

Роль логики в проблематике представления знаний и рассуждений многообразна.

- Логику можно прямо использовать для представления знаний и рассуждений.
- Она может пригодиться для ссылок и как эталон выразительности, модель компетенции, гарант элементарных логических принципов. Мо-

жет помочь в точном определении альтернативных методов.

- Она определяет принципы и законы, незаменимые при решении многих проблем.
- Она позволяет анализировать смысл некоего представления знаний и обоснованность выводов.
- В этом отношении она является преимущественно средством анализа знаний и рассуждений как таковых.

4.2. Логика и модифицируемые рассуждения

4.2.1. Формализация модифицируемых рассуждений

Классическая логика формализует строго корректные рассуждения. Моделирование встречающихся в ИИ рассуждений не должно ограничиваться формализацией непогрешимого интеллекта. Наш интеллект часто способен вырабатывать разумные рассуждения в условиях неопределенности. Имея дело с неполной, неточной или изменчивой информацией, наши рассуждения часто предположительны, всего лишь правдоподобны и должны подвергаться пересмотру. Рассмотрим пример. «Зная, что большинство птиц может летать и что Тити — птица, я заключаю, что Тити может летать». Этот вывод кажется приемлемым. Между тем он не является абсолютно корректным и общезначимым, ибо не учитывает возможных исключений. Следовательно, он неточен и подлежит пересмотру. Если уточнено, что Тити — страус, то утверждение «Тити может летать» отвергается.

Априори ясно, что логическая система для формализации модифицируемых рассуждений должна быть немонотонной. Численность теорем, которые можно получить, может уменьшаться при росте числа основных аксиом или предпосылок.

4.2.2. Классическая логика и общезначимые рассуждения

Дедуктивные системы классической логики ограничиваются формализацией общезначимых рассуждений

(разд. 2.1). Следовательно, как таковые они не подходят для формализации нестрогих и модифицируемых рассуждений. Фундаментальные свойства формальных систем дедукции ясно свидетельствуют об этой ограниченности.

Формальная система дедукции классической логики состоит из множества схем аксиом и правил вывода (разд. 2.1). Она позволяет делать заключения из предпосылок. Таким образом, между формулами определено отношение выводимости \vdash , которое [35]:

- рефлексивно: $\{p_1, \dots, p_n, q\} \vdash q$,
- монотонно: если $\{p_1, \dots, p_n\} \vdash q$, то $\{p_1, \dots, p_n, r\} \vdash q$,
- транзитивно: если $\{p_1, \dots, p_n\} \vdash r$ и $\{p_1, \dots, p_n, r\} \vdash q$, то $\{p_1, \dots, p_n\} \vdash q$.

Здесь p_1, \dots, p_n, q, r — формулы рассматриваемого логического языка. Эти фундаментальные свойства формализуют требования к общезначимым рассуждениям:

- Вывод заключения, идентичного одной из посылок, есть общезначимая операция.
- Полученный результат не опровергается дальнейшими.
- Промежуточные результаты можно использовать для установления общезначимости заключения.

Формальные системы дедукции классической логики предстают как алгебраические системы переписывания с учетом этих требований. Свойство монотонности препятствует прямой формализации модифицируемых рассуждений. Следовательно, с чисто синтаксической точки зрения построение немонотонной системы вывода делает необходимым ослабление свойств дедуктивных систем классической логики. Классическое определение отношения семантического следования непригодно для формализации модифицируемых рассуждений.

Семантическая характеристика логической системы состоит в приписывании семантических значений выражениям языка посредством *интерпретации*. Если последняя истинна, то она есть *модель* (§ 1.2.5). Эта характеристика осуществляется посредством приписывания семантических значений основным выражениям языка и определения правил семантической оценки¹⁾ более сложных выражений. Например, таков композиционный принцип, который был описан в §§ 1.1.4, 3.1.10.

Отношение семантического следования \models (§ 1.1.6) между множеством посылок A и заключением p классически определяется так:

$A \models p$, если любая модель для A является моделью для p . В этом определении слово «любая» влечет, что из A можно вывести лишь «точные» («общезначимые») следствия.

Модифицируемое рассуждение не является в классическом смысле общезначимым. Вывести p из множества посылок A и отказаться от p , как только информация q будет добавлена к A , означает, что допустимо вывести p , в то время как существует модель для $A \cup \{q\}$ (которая тем более является моделью для A), не подтверждающая p .

Для построения немонотонной логики нужно определить отношение вывода, позволяющее получать заключения, которые подтверждаются не во всех моделях для посылок.

Неадекватность систем дедукции классической логики для формализации модифицируемых рассуждений объясняется также тем, что их правила вывода являются лишь *позволяющими* [74]. Они всегда имеют вид: « q — теорема, если p_1, p_2, \dots, p_n — теоремы». Это позволяет лишь получать новые теоремы, но не отказываться от ранее полученных теорем. Моделирующая модифицируемые рассуждения система должна была бы содержать также *ограничивающие* правила вида: « q — теорема, если p_1, p_2, \dots, p_n — не теоремы».

¹⁾ Илн, иначе, означивания. — *Дрим. перев.*

4.2.3. Характеристики немонотонных логик

Немонотонные логики должны иметь системы вывода для моделирования модифицируемых (следовательно, необщезначимых в классическом смысле) рассуждений. Формализующая эти рассуждения система вывода должна давать «правдоподобные» формулы. С семантической точки зрения это сводится к выводу выполнимых вместе с посылками формул (в смысле их подтверждения хотя бы в одной модели). В этом предположении моделируют разумного субъекта, заключения которого выполнимы вместе с множеством исходных сведений. Например, заключение «Тити летает» не является общезначимым следствием из множества двух посылок: «Большинство птиц летает» и «Тити — птица». Оно просто выполнимо с этим множеством. Следовательно, заключение принадлежит к возможно выполнимому на основе двух своих посылок образу мира.

С другой стороны, кажется разумным потребовать, чтобы невыполнимые в совокупности заключения не выводились бы все вместе. Даже от способного лишь на правдоподобные заключения субъекта можно потребовать согласованности утверждений. Недопустимо ему утверждать одновременно «Тити летает» и «Тити не летает».

Требование выполнимости связано с модифицируемостью. При поступлении новой информации предположения могут стать невыполнимыми с новым множеством посылок и будут отвергнуты. Узнав, что Тити — страус, и зная, что страусы не летают, мы отвергнем утверждение: «Тити летает».

Немонотонная логика даст возможность выводить формулы, выполнимые как сами по себе, так и с данным множеством посылок. В этом контексте нужно выяснить ряд логических вопросов.

● Каков статут для формул, которые могут быть выведены?

Они больше не тавтологии и не общезначимы (в классическом смысле) по отношению к своим посылкам, но лишь выполнимы с ними (§§ 1.1.6, 1.2.5).

● Какова структура множеств выведенных формул?

В классической дедуктивной системе растущее множество образуют все заключения, выводимые из множества посылок. Множество посылок можно увеличить лишь вместе с увеличением множества заключений. Немонотонная система не обязана обладать этим свойством. Из нее можно получить, используя правила вывода, различные несовместные множества формул. Отдельно взятое множество заключений существенно зависит от порядка применения правил вывода. Если выведена всего лишь правдоподобная формула, то следует запретить дальнейшее выведение других правдоподобных (но невыполнимых вместе с первой) формул. Например, если выведено «Тити летает», то нельзя выводить «Тити не летает». И наоборот. Таким образом, в зависимости от порядка применения правил вывода можно получить два разных множества формул.

● Как избежать наличия нескольких несовместных множеств возможных заключений?

Можно ослабить монотонность (§ 4.2.2) до *ограниченной монотонности* [35]:

если $\{p_1, \dots, p_n\} \vdash r$ и $\{p_1, \dots, p_n\} \vdash q$, то $\{p_1, \dots, p_n, r\} \vdash q$

(где \vdash есть символ отношения выводимости в рассматриваемой системе).

Заметим, что ограниченная монотонность не позволяет моделировать взаимно несовместимые альтернативы рассуждений.

● Как охарактеризовать множества возможных заключений?

Когда формулы сначала выводят, а потом отвергают, утрачивается простая итеративная структура классических аксиоматических систем (§ 2.1.2), позволяющая строить и перечислять множества возможных заключений.

Охарактеризуем множества выводимых формул с помощью «неподвижных точек» соответствующих операций. Неподвижная точка представляет собой устойчивое множество предположений, из которого нельзя вывести никакую новую выполнимую формулу. Этот метод непосредственно применяется в *немонотонных логиках* Мак-Дермотта (разд. 4.5). В *логиках умолчаний* Рейтера (разд. 4.3) он предстает в виде *расширений с умолчаниями*, а в автоэпистемических логиках Столнекера и Мура (разд. 4.6) — в виде *устойчивых расширений*, полных и легальных множеств заключений идеально разумного субъекта, выведенных из множества посылок.

● **Что станет с понятиями теоремы, общезначимого вывода и доказательства?**

Мнения расходятся. В классической логике теорема — это тавтология и результат общезначимых выводов. В теории формальных языков теорема понимается шире, как утверждение, выводимое с помощью системы подстановок термов (которая не обязательно дает общезначимые выводы). В немонотонной логике за «теоремы» можно принять формулы, присутствующие во *всех* выводимых устойчивых множествах утверждений. Можно также поинтересоваться строением *одного* из этих множеств. Тогда будут представлять интерес и все утверждения, которые могут выражать картину мира, нарисованную неким субъектом с использованием исходных предположений и с соблюдением свойства выполнимости. В последнем случае *доказательство* сводится к установлению существования для данной формулы устойчивого выполнимого множества предположений. Оно должно быть выводимым из рассматриваемых посылок и содержать нашу формулу.

Что будет с классическими метатеоремами?

Без монотонности большинство метатеорем классической логики станут необщезначимыми. Например, метатеорема дедукции (§ 2.1.4) неверна в немонотонных логиках Мак-Дермотта (разд. 4.5).

4.2.4. Зацикливание правил немонотонного вывода

Если немонотонная система должна *сама обеспечить* отказ от своих выводов, то ее правила вывода должны быть *модифицируемыми*. Их применение может динамически блокироваться. Для этого они оснащаются условиями применения, проверка которых динамически изменяется вместе с множеством посылок. Эти правила будем называть *правилами немонотонного вывода*. Их условия (называемые *предусловиями*) позволяют проверять (до вывода) выполнимость некоего утверждения вместе с уже выведенными утверждениями в этой системе из существующего множества посылок.

Рассмотрим правило «если x птица, то x летает». Сделаем его правую часть выводимой, когда она всего лишь выполнима вместе с посылками и другими (уже выведенными) формулами. Для этого преобразуем наше правило в «если x птица и если *выполним* вывод, что x летает, то x летает». Условие выполнимости само может ставиться в форме невыводимости некоего утверждения: «Если выполним вывод, что x летает» можно заменить на «если нельзя вывести, что x не летает».

Проверка такого рода предусловий может динамически изменяться вместе с множеством формул, уже выведенных системой, успешно сокращая множество утверждений, выполнимых при данном состоянии системы.

Как интерпретировать условия *выполнимости* и *выводимости*, используемые в этих предусловиях? В самом деле, интерпретация этих условий должна апеллировать к отношению выводимости, заданному системой, которая содержит эти правила с предусловиями. Интерпретация понятия выводимости, входящего в предусловие «нельзя сделать вывод, что x не летает», должна учитывать то, что можно вывести с помощью оснащенных этим предусловием правил.

Есть опасность зацикливания в определении и интерпретации отношения выводимости. Для проверки невыводимости некоего утверждения модифицируемые правила должны апеллировать к содержащей их

системе вывода. Таким образом, для проверки предусловия «нельзя сделать вывод, что x не летает» придется рассматривать все правила вывода системы, включая немонотонные.

Следовательно, нельзя определить независимые отношения выводимости, чтобы задать проверку условий применения модифицируемых правил, с одной стороны, и охарактеризовать оснащенную правилами с предусловиями систему вывода, с другой стороны.

Рассмотрим эту проблему ближе. Пусть p — высказывание, A — множество высказываний, считающихся посылками. Определим примитив *UNLESS* [97], [59] следующим образом:

- *UNLESS*(p) истинно тогда и только тогда, когда p недоказуемо с использованием множества посылок A в логике высказываний.

Этот примитив можно применять в определении правил немонотонного вывода перед разрешением получить новое утверждение, исходя из предположения о невыводимости некоторых других утверждений.

Для этого применим *UNLESS* к высказываниям данного языка как оператор истинности. Дедуктивная система логики высказываний, пополненная новыми правилами с этим оператором и примененная к множеству посылок A , немонотонна. Но она не обладает желаемыми свойствами. В частности, противоречия уже изученному, она не запрещает вывести одновременно p и *UNLESS*(p), ибо *UNLESS* не зависит от нового отношения выводимости (как хотелось бы), но лишь от дедуктивной системы логики высказываний.

Действительно, вообразим немонотонное правило

$$UNLESS(q) | \sim p.$$

Предполагая недоказуемость p из множества посылок A в логике высказываний, выводим *UNLESS*(p), опираясь прямо на определение *UNLESS*. С другой стороны, если q недоказуемо из множества посылок A в логике высказываний, то получаем *UNLESS*(q). Из последнего по правилу немонотонного вывода следует p .

Итак, годная для проверки условия немонотонного правила система выводы должна содержать это правило.

4.2.5. Полирасширяемость немонотонной системы

Проиллюстрируем проблему полирасширяемости немонотонной системы. Для этого предположим, что *UNLESS* корректно определен и *зависит* не только от исчисления высказываний, но и от всех других правил и аксиом данной системы. Последние могут и сами использовать примитив *UNLESS*.

Пусть \mathcal{L}_p — пополненный оператором *UNLESS* язык исчисления высказываний. A — множество $\{a, a \wedge \text{UNLESS}(b) \supset c, a \wedge \text{UNLESS}(c) \supset b\}$, где a, b и c — высказывания из \mathcal{L}_p . Очевидно, либо b , либо c выводимы из A , но не оба сразу. Действительно, если b невыводимо в этой системе, то имеем *UNLESS*(b). Откуда можно вывести c . Но тогда нельзя вывести *UNLESS*(c). Совершенно симметрично, если невыводимо c , то выводимо b . Так можно получить различные несовместные множества заключений из одного множества посылок.

4.2.6. Различные формы немонотонных рассуждений

Природа модифицируемых рассуждений различна. Причины модификации знаний самые разные. Для выбора наиболее адекватных методов моделирования надо понимать явления и гипотезы, участвующие в рассуждениях.

Выделим два класса модифицируемых рассуждений.

- Рассуждения, модифицируемые из-за неопределенности и гипотетичности: *Вообще объекты типа X имеют свойство P . Если A — объект типа X , то я делаю вывод, что A (по-видимому) обладает свойством P .* Пример: «Если Тити — птица, то я вывожу, что (по-видимому) Тити летает».

Знание, которое позволяет выводить подобные заключения, принадлежит к типу «большинство птиц летает» или «типичная птица летает». Это рассуждение неточно и дополнительная информация может привести к его модификации.

- Рассуждения, модифицируемые по интроспективной¹⁾ природе:

Исходя из состояния моих знаний, я могу сделать вывод, что...

Пример: «Мне ничего не известно о старшем брате, и отсюда я делаю вывод, что у меня нет старшего брата».

Утверждение о том, что у меня нет старшего брата, сделано не потому, что «правдоподобно», что его у меня нет. Механизм рассуждения иной. Он интроспективен и основан на предположении о том, что все знания, имеющиеся по этому вопросу, таковы: «если бы у меня был старший брат, то я бы об этом знал».

Можно потребовать от этих рассуждений, чтобы они осуществлялись «определенным образом», в предположении наличия и корректности всей соответствующей информации, полагая при этом, что все, что не дано, ложно. Иначе говоря, можно потребовать от рассуждений «общезначимости» относительно этого состояния знаний [80]. Модифицируемый характер рассуждений проистекает из зависимости от состояния знаний. Оно присуще рассуждающему субъекту и может изменяться.

Попутно заметим, что многие модифицируемые от неопределенности рассуждения моделируемы интроспективно. Но различие между двумя классами модифицируемых логик важно для понимания областей применимости разных немонотонных логик. Последние перечислены ниже и описаны далее в этой главе.

- Логика умолчаний (разд. 4.3) — это логические системы, в которых немонотонность обусловлена необщезначимостью правил вывода, присутствующих в области применения. Правила выражают

¹⁾ Зависящей от текущего состояния знаний субъекта. — Прим. перев.

знания типа «большинство птиц летает» в виде: «если вывод, что птица может летать, является выполнимым, то выводимо, что она летает». Таким образом, эти приемы вывода позволяют выразить правила с исключениями, не перечисляя исключений.

- Немонотонные логики Мак-Дермотта (разд. 4.5). Их назначение — предложить универсальную (независимую от области применения) аксиоматическую систему оценки «выполнимых» множеств утверждений, выводимых из множества посылок.
- Автоэпистемические логики (разд. 4.6) — это реконструкции немонотонных логик Мак-Дермотта. Они моделируют чисто интроспективные рассуждения. Идеально разумный субъект рассуждает на основе своих предположений. Рассуждения модифицируемы, ибо зависят от изменчивого состояния знаний.
- Методы, основанные на принципе замкнутого мира и принципе ограничения. Предполагается наличие в системе всей информации по данной проблеме. Соответствующие методы развиваются и обосновываются *теорией моделей* (разд. 2.2). Они определяют (часто неявным образом) правила вывода истинных выражений из посылок в некоторых моделях. Эти методы будут представлены во втором томе.

4.3. Логика умолчаний

4.3.1. Введение

Логика умолчаний¹⁾ введены и развиты Рейтером [88] для формализации рассуждений, являющихся всего лишь выполнимыми. При неполной информации мы вынуждены получать всего лишь правдоподобные предположительные заключения. Иногда мы считаем абсолютно общими правила, которые правильны

¹⁾ Другое название — *логики типичного*. — Прим. перев.

в большинстве случаев, но допускают некие исключения.

Если Тити птица, то выводимо, что Тити летает. Не все птицы летают. Но можно без особого на то разрешения заключить «Тити летает», если это не запрещено. Так как «Тити летает» выполнимо вместе с моими представлениями, то я заключаю «Тити летает» (ибо это наиболее естественно). Это *рассуждения с умолчаниями*.

Логика умолчаний позволяют формализовать такие рассуждения в виде правил вывода, называемых *умолчаниями*:

$$\frac{\alpha : M\beta}{\gamma}$$

Интуитивный смысл таков. Если мы верим в α и если β выполнимо вместе со всем, во что мы верим, то можно верить и γ .

Итак, правило «птицы вообще летают» выразимо в виде:

$$\frac{\text{Птица}(x) : M \text{Летает}(x)}{\text{Летает}(x)}$$

Интуитивно: если x птица и если выполнимо « x летает», то выводимо « x летает». Общее правило с исключениями гласит, что *типичные* птицы летают. Это правило умолчания позволяет обрабатывать исключения без их предварительной идентификации.

Система логики умолчаний представляется *теорией с умолчаниями* (или подробнее: *с правилами с умолчаниями*), состоящей из некоторого множества особо выделенных формул и правил вывода. В ней содержатся формулы логики предикатов, представляющие основную информацию о системе, обрабатываемую в соответствии с имеющимися аксиомами. Содержатся также правила умолчаний, отражающие различные утверждения, касающиеся исключений.

Для такой системы существует несколько (нуль или больше) множеств выводимых предположений. Эти множества представляют различные картины мира, которые можно вообразить, исходя из теории с умолчаниями.

4.3.2. Теории с умолчаниями

Обозначим через \mathcal{L} язык предикатов первого порядка (разд. 2.2).

Правило умолчания (сокращенно: *умолчание*)

\mathcal{D} — это выражение вида

$$\frac{\alpha(x) : M\beta_1(x), \dots, M\beta_m(x)}{\gamma(x)},$$

где

- $\alpha(x)$, $\beta_1(x)$, ..., $\beta_m(x)$ и $\gamma(x)$ — формулы языка \mathcal{L} , свободные переменные у которых выбраны среди $x = (x_1, \dots, x_n)$,
- $\alpha(x)$ называется *требованием* умолчания \mathcal{D} , $\beta_i(x)$ — *обоснованием* умолчания \mathcal{D} , $i = 1, \dots, m$.
- $\gamma(x)$ — *следствием* умолчания \mathcal{D} ,
- M — некий символ метаязыка.

Умолчание \mathcal{D} называется *замкнутым* тогда и только тогда, когда $\alpha(x)$, $\beta_1(x)$, ..., $\beta_m(x)$ и $\gamma(x)$ не содержат свободных переменных. При этом можно использовать более простые обозначения: α , β_1 , ..., β_m и γ соответственно.

Свободные переменные умолчания считаются \forall -квантифицированными. Область действия этих кванторов простирается на все члены умолчания. Незамкнутое умолчание называется *открытым*. Оно представляет общую схему вывода. Его конкретизацией является замкнутое умолчание, полученное заменой всех свободных переменных открытого умолчания на константы языка \mathcal{L} (с соблюдением неявного закона об области действия свободных переменных умолчания).

Теория с умолчаниями Δ — это пара (D, F) , где

- D — множество умолчаний,
- F — множество замкнутых формул из \mathcal{L} .

Она называется *замкнутой* тогда и только тогда, когда все умолчания из D замкнуты.

4.3.3. Примеры применения умолчаний

Для начала рассмотрим пример представления неполных знаний (этот пример заимствован из [88]).

Пусть неполные знания представлены двумя умолчаниями:

- По-русски: "Человек обычно живет вместе со своей семьей"
- Логически: $(\text{Семья } (x, y) \wedge (\text{Живет } (y) = z) : \text{М Живет } (x) = z)$

$$\frac{\text{Живет } (x) = z}{\text{Живет } (x) = z}$$
- По-русски: "Человек обычно живет в одном городе с нанимателем"
- Логически: $\text{Наниматель } (x, y) \wedge (\text{Город } (y) = z) : \text{М } (\text{Живет } (x) = z)$

$$\frac{\text{Живет } (x) = z}{\text{Живет } (x) = z}$$

Предположим, что супруг Мари живет в Брюсселе, а ее наниматель — в Париже. Два наших правила вывода дают два невыполнимых совместно заключения. Если сперва применить первое правило, то нужно воспрепятствовать всем выводам, приводящим к «Мари живет в Париже». «Мари живет в Брюсселе» и «Мари живет в Париже» — два (невыполнимых совместно) *расширения* данной теории с вышеуказанными умолчаниями.

Следующий пример [32] иллюстрирует возможность использования умолчаний в иерархических структурах с исключениями. Утверждения:

- 1) моллюски являются раковинными,
 - 2) головоногие — моллюсками, но не раковинными,
 - 3) наутилусы — головоногими и раковинными
- представимы теорией Δ с двумя умолчаниями и тремя формулами.

- Умолчание D_1 :
$$\frac{\text{Мо } (x) : \text{М } (\text{Ра } (x) \wedge \neg \text{Го } (x))}{\text{Ра } (x)}$$

По-русски: Если x — моллюск и выполнимо « x раковинный и не головоногий», то x раковинный.

- Умолчание D_2 :
$$\frac{\text{Го } (x) : \text{М } (\neg \text{Ра } (x) \wedge \neg \text{На } (x))}{\neg \text{Ра } (x)}$$

По-русски: Если x головоногий и выполнимо « x не раковинный и не наутилус», то x не раковинный.

- Формула $F_1 : \forall x (Ha(x) \supset Go(x))$

По-русски: Наутилусы являются головоногими.

- Формула $F_2 : \forall x (Go(x) \supset Mo(x))$

По-русски: Головоногие являются моллюсками.

- Формула $F_3 : \forall x (Ha(x) \supset Pa(x))$

По-русски: Наутилусы являются раковинными.

Если x наутилус, то эти три формулы дают: « x — головоногий, моллюск и раковинный». Этот вывод определяет единственное расширение данной теории объединением теории Δ и утверждения « x наутилус».

Если x головоногий и не наутилус, то из формулы F_2 следует: « x моллюск», а из правила D_2 следует: « x не раковинный». Этот вывод определяет единственное расширение данной теории — объединение Δ и утверждения « x головоногий и не наутилус». Формальное определение расширения см. в § 4.3.4.

4.3.4. Расширения теорий с умолчаниями

Теория с умолчаниями $\Delta = (D, F)$ подразумевает некоторое (нулевое или большее) число множеств предположений, которые выводимы с использованием множества формул F , и удовлетворяет свойству выполнимости. Эти множества предположений называются *расширениями* данной теории с умолчаниями.

Расширения теории с умолчаниями явно определены здесь лишь для замкнутых теорий. Открытую теорию можно преобразовать эффективным образом в замкнутую, заменяя каждое открытое умолчание множеством всех его конкретизаций, получаемых посредством применения открытых умолчаний к эрбрановой области (§ 1.2.9) данной теории. Используя это преобразование, полученные для замкнутых теорий с умолчаниями результаты можно распространить на открытые теории (когда они конечны). Заметим, что теории, содержащие функциональные символы ненулевой ариальности, имеют бесконечную эрбранову область.

Прежде чем приступать к формализации, охарактеризуем интуитивно свойства, которыми должно

обладать расширением замкнутой теории с умолчаниями. Расширение — это надмножество основных сведений системы, включающее все выводимое (по правилам классической логики и/или логики умолчаний).

Пусть X — подмножество из \mathcal{L} , $Th_{\mathcal{L}}(X)$ — множество замкнутых формул, общезначимо выводимых из X по классическим правилам вывода из \mathcal{L} :

$$Th_{\mathcal{L}}(X) = \{\omega \mid \omega \in \mathcal{L}, \omega \text{ замкнута и } X \vdash \omega\}.$$

Пусть $\Delta = (D, F)$ — теория с умолчаниями, S — подмножество в \mathcal{L} . Обозначим через $\Gamma(S)$ наименьшее подмножество в \mathcal{L} , удовлетворяющее следующим трем условиям:

- $F \subseteq \Gamma(S)$,
- $Th_{\mathcal{L}}(\Gamma(S)) = \Gamma(S)$,
- если $\frac{\alpha : M\beta_1, \dots, M\beta_m}{\gamma} \in D$, $\alpha \in \Gamma(S)$
и $\neg \beta_1, \dots, \neg \beta_m \notin S$, то $\gamma \in \Gamma(S)$.

Множество формул $E \subseteq \mathcal{L}$ является *расширением* для Δ тогда и только тогда, когда $\Gamma(E) = E$ (т. е. E — неподвижная точка оператора Γ).

Расширение E можно охарактеризовать так. Строим последовательность формул E_i , полагая $E_0 = F$ и

$$E_{i+1} = Th_{\mathcal{L}}(E_i) \cup \left\{ \gamma \mid \frac{\alpha : M\beta_1, \dots, M\beta_m}{\gamma} \in D \text{ или } \alpha \in E_i \text{ и } \neg \beta_1, \dots, \neg \beta_m \notin E_i \right\}$$

для $i = 0, 1, 2, \dots$. Множество E есть расширение для Δ тогда и только тогда, когда

$$E = \bigcup_{i=0}^{\infty} E_i.$$

4.3.5. Примеры расширений теорий с умолчаниями

Теория с умолчаниями иногда позволяет вывести несколько расширений из одного множества посылок.

- Пример 1 [70].

Пусть $\Delta = (D, F)$, где $D = \left\{ \frac{:MA}{\neg P}, \frac{:MP}{\neg Q}, \frac{:MQ}{\neg S} \right\}$
и $F = \emptyset$.

Эта теория обладает расширением

$$E = Th_{\mathcal{L}}(\{\neg P, \neg S\}).$$

● Пример 2.

Пусть $\Delta = (D, F)$, где $D = \left\{ \frac{:MA}{\neg A} \right\}$ и $F = \emptyset$.

Эта теория не имеет расширений.

● Пример 3 [70].

Пусть $\Delta = (D, F)$, где $D = \left\{ \frac{:MA}{\neg B}, \frac{:MB}{\neg A} \right\}$ и $F = \emptyset$.

У этой теории два расширения: $E_1 = Th_{\mathcal{L}}(\{\neg A\})$ и $E_2 = Th_{\mathcal{L}}(\{\neg B\})$.

● Пример 4 [88].

Пусть $\Delta = (D, F)$, где $D = \left\{ \frac{A : M\exists xP(x)}{\exists xP(x)}, \frac{:MA}{A}, \frac{:M\neg A}{\neg A} \right\}$ и $F = \emptyset$.

Расширений два:

$$E_1 = Th_{\mathcal{L}}(\{\neg A\}) \text{ и } E_2 = Th_{\mathcal{L}}(\{A, \exists xP(x)\}).$$

Например, первым умолчанием из D можно формализовать «большинство профессоров университета имеет степень доктора»:

$$\frac{\text{Проф_унив}(x) : M(\exists y \text{ Конкр}(y, \text{доктор}) \wedge \text{Имеет}(x, y))}{\exists y \text{ Конкр}(y, \text{доктор}) \wedge \text{Имеет}(x, y)}$$

4.3.6. Нормальные теории

Как показывает пример 2 из § 4.3.5, некоторые теории с умолчаниями не обладают расширениями. Существование расширений гарантировано, если следствие и обоснование одного и того же умолчания совпадают [88]. Такие теории называются *нормальными*. Они

состоят из *нормальных умолчаний*:

$$\frac{\alpha(x) : M\beta(x)}{\beta(x)}.$$

Кроме интересного свойства допускать хотя бы одно расширение нормальная теория с умолчаниями обладает свойством *полумонотонности*: если увеличить множество умолчаний, то полученная теория допускает расширение, включающее какое-то расширение исходной теории [88]. Практически важное следствие этого свойства состоит в возможности построения такой теории доказательств, в которой использованные умолчания проявляются локальным образом.

4.3.7. Теория доказательств для нормальных теорий

Можно ли построить теорию доказательств для логик умолчаний? В частности, для замкнутых нормальных теорий? Точнее, даны замкнутая нормальная теория Δ и замкнутая формула f из \mathcal{L} . Существует ли метод проверки наличия для Δ расширения E , содержащего f ?

Для получения ответа Рейтер [88] определил *доказательство в теории с умолчаниями* следующим образом. Пусть $\Delta = (D, F)$ — замкнутая нормальная теория и f — замкнутая формула из \mathcal{L} . Конечная последовательность D_0, \dots, D_k конечных подмножеств из D есть доказательство для f в Δ тогда и только тогда, когда

1. $F \cup \{KC(D_0)\} \vdash f$,
2. $F \cup \{KC(D_i)\} \vdash KT(D_{i-1})$ для $i = 1, 2, \dots, k$,
3. $D_k = \emptyset$,
4. $F \cup \{KC(D_i) \mid 0 \leq i \leq k\}$ выполнимо,

где $KC(D_i)$ — конъюнкция следствий и $KT(D_i)$ — конъюнкция требований умолчаний из D_i .

Итак, доказательство есть последовательность подмножеств умолчаний. Его можно интерпретировать следующим образом. Первое подмножество (D_k) выбирается пустым. Последовательно строятся

D_{k-1}, \dots, D_1, D_0 . Множество основных аксиом с добавленной к нему конъюнкцией следствий из всех умолчаний D_0 должно обеспечивать доказательство f классическим образом. Из построения подмножества D_{i-1} вытекает, что множество F (с добавленной к нему конъюнкцией следствий из D_i) должно позволять доказывать классическим образом требования из D_{i-1} и, следовательно, гарантировать применимость умолчаний из D_{i-1} . Глобальная применимость всех умолчаний устанавливается проверкой выполнимости объединения F и конъюнкций следствий всех использованных умолчаний.

Заметим, что в определении не говорится, как строить подмножества D_i , и не приводится разрешающей процедуры для используемого отношения доказательства (из классической теории предикатов первого порядка). Более того, оно предполагает проверку выполнимости некой формулы из \mathcal{L} , тогда как множество замкнутых выполнимых формул из \mathcal{L} не является рекурсивно перечислимим.

Метод Рейтера [88] сочетается со свойством полноты. Пусть f — замкнутая формула из \mathcal{L} . Нормальная теория Δ (замкнутая и выполняемая) обладает расширением E (содержащим f) тогда и только тогда, когда f обладает доказательством в Δ .

К сожалению, во всей общности проблема проверки существования расширения с данной замкнутой формулой не полуразрешима (§ 2.2.6). Это не столь неожиданно. Тем не менее в некоторых случаях она поддается практическому решению (особенно когда ограничиваются рамками высказываний).

Можно определить особые теории с умолчаниями, разрешимые с помощью метода полного доказательства. Например, Бенар, Кинью и Кинтон [6] применяют метод насыщения, позволяющий реализовывать метод полного доказательства для разрешимых теорий, содержащих только так называемые «свободные» умолчания:

$$\frac{: MP(x) \supset R(x)}{P(x) \supset R(x)}$$

4.3.8. Полунормальные теории

Нормальные умолчания выглядят достаточно привлекательными при представлении многих форм рассуждений. Свойства формальных систем, составляющих нормальные теории, особенно выигрышны в силу следующих обстоятельств.

Следствие и обоснование нормального умолчания совпадают. Следовательно, нормальные умолчания неприменимы, когда ложность их следствий доказана. Эти умолчания не могут вводить невыполнимости, опровергать обоснования из других ранее примененных нормальных умолчаний и своих собственных. Итак, нормальные теории полумонотонны, всегда обладают хотя бы одним расширением и представляют довольно простую теорию доказательств.

Между тем в ходе применения нормальных умолчаний могут возникать неприятные осложнения. В частности, взаимодействие различных нормальных правил может приводить к нежелательным заключениям [89].

Поэтому иногда необходимо *блокировать транзитивность* между умолчаниями. Например, рассмотрим нормальную теорию $\Delta = (D, F)$, где D содержит два нормальных умолчания:

«обычно студент университета является взрослым»,
или в символьном виде:

$$\frac{C(x) : MB(x)}{B(x)}$$

«обычно на работу берут взрослых», или в символьном виде:

$$\frac{B(x) : MP(x)}{P(x)}$$

и где F — множество из одного элемента $\{C(\text{Эрик})\}$.

Правила из D позволяют по умолчанию вывести, что студента университета взяли на работу. Это нежелательный вывод. Осложнение предотвращается с помощью третьего нормального умолчания: «обычно студента не берут на работу». Таким образом, получается нормальная теория $\Delta' = (D', F)$ где D' — мно-

жество умолчаний:

$$\left\{ \frac{C(x) : MB(x)}{B(x)}, \frac{B(x) : MP(x)}{P(x)}, \frac{C(x) : M \neg P(x)}{\neg P(x)} \right\}.$$

Для данного студента множество D' может дать два расширения теории Δ' (соответствующих различной занятости студента):

$Th_{\mathcal{L}}(\{C(\text{Эрик}), B(\text{Эрик}), \neg P(\text{Эрик})\})$ и

$Th_{\mathcal{L}}(\{C(\text{Эрик}), B(\text{Эрик}), P(\text{Эрик})\})$.

Тем не менее разумно потребовать блокирования транзитивности между двумя первыми правилами из D . Следовало бы также выделить априори расширение, соответствующее ситуации, когда студента не принимают на работу. Это можно осуществить модификацией второго правила: чтобы оно не могло быть применено в исключительном случае — «взрослый является студентом».

Итак, для теории $\Delta'' = (D'', F)$, где D'' есть

$$\left\{ \frac{C(x) : MB(x)}{B(x)}, \frac{C(x) : M \neg P(x)}{\neg P(x)}, \frac{B(x) : M(P(x) \wedge \neg C(x))}{P(x)} \right\},$$

получаем желаемое расширение

$Th_{\mathcal{L}}(\{C(\text{Эрик}), B(\text{Эрик}), \neg P(\text{Эрик})\})$.

Третье умолчание D'' полунормальное [89], т. е. имеет вид

$$\frac{\alpha(x) : M(\beta(x) \wedge \gamma(x))}{\beta(x)}.$$

Таким образом, полунормальное умолчание явно управляется дополнительным условием в обосновании. Теория с полунормальными умолчаниями (полунормальная теория) не обязательно обладает расширением. Она теряет некоторые достоинства нормальных теорий, в частности полумонотонность.

4.3.9. Наследственные системы с исключениями

Системы сетевого и объектного представлений часто позволяют выразить наследование свойств с исклю-

чениями и встроить механизмы вывода, связанные с этими методами (§ 3.3.12). Поведение такой системы редко бывает корректным и полностью охарактеризованным. Соответствующие методы довольно плохо освоены [110], [29].

Если наследственные свойства между классами и подклассами можно относительно легко охарактеризовать в классической логике (§ 3.3.11), то исследование исключений требует ухищрений. Логика умолчаний очерчивает естественные рамки для формализации систем представления знаний и рассуждений с исключениями.

Между тем прямое обращение к теории с умолчаниями (в частности, к нормальной) простым не является. Наследственная система обеспечивает передачу свойств по транзитивности (§ 3.2.19). Когда к системе добавляются исключения, транзитивность свойств должна допускать возможность блокирования.

В литературе предложены различные формализмы систем представления с механизмами наследования свойств.

- Можно использовать полунормальные умолчания [27], [29]. Исключения для наследования свойств явно перечислены в умолчаниях. Этерингтон [29] определил подкласс полунормальных умолчаний (связанных отношением зависимости), для которых соответствующие полунормальные теории всегда обладают хотя бы одним расширением. Он предложил процедуру эффективного построения полунормальных теорий.
- Можно использовать нормальные умолчания с неявным порядком, подчиненным иерархии моделируемой структуры [110]. Можно не упоминать явно исключения в умолчаниях.
- Можно использовать таксономические теории с умолчаниями (не являющиеся ни нормальными, ни полунормальными). Они обладают единственными расширениями [33], [34].

4.4. Модальные логики знания и веры

4.4.1. Введение

Первейшей функцией модальной логики является формализация модальностей «возможность» и «необходимость». Другое ее применение — моделирование и анализ парадигм «знание» и «вера». Для этого логические системы используют формальные языки с модальными операторами «веры» и «знания». Системы сочетают различные схемы аксиом и правила вывода для формализации свойств этих операторов. Модальные системы снабжены специфической семантикой. Различные модальные логики, которые мы рассмотрим, являются расширениями логики первого порядка. В частности, они заимствуют оттуда аксиомы, правила вывода и теоремы.

4.4.2. Некоторые элементарные модальные системы

Ограничимся неквантифицированными частями модальных языков с синтаксисом из § 3.1.14. Пусть \mathcal{L} — модальный язык высказываний, p и q — метаварьируемые, представляющие формулы в языке \mathcal{L} . Модальные операторы в \mathcal{L} обозначим символами L и M (в модальной логике необходимости и возможности это \Box и \Diamond). Операторы общности L и существования M двойственны: $L \equiv \neg M \neg$.

В логиках веры и знания оператор L принимает соответственно значения «предполагается» и «известно». Значения оператора M — соответственно «противоположное не предполагается» и «противоположное не известно».

Нормальная модальная система — это четверка, состоящая из:

- Множества всех теорем логики высказываний (разд. 1.1), область действия которых распространена на формулы модального языка высказываний \mathcal{L} .

- Схемы аксиомы дистрибутивности

$$L(p \supset q) \supset (Lp \supset Lq),$$

обозначаемой буквой K . В соответствии с толкованием модальности «необходимость» схема K утверждает, что «если необходимо, что p влечет q , то из необходимости p вытекает необходимость q ».

- *Правила modus ponens*

$$\frac{p \quad p \supset q}{q}$$

- *Модального правила вывода необходимости*

$$\frac{p}{Lp}$$

(« p необходимо истинно» при условии, что « p истинно»).

Можно получить модальные системы и поизощреннее — обогащая нормальную модальную систему различными схемами аксиом, вроде нижеследующих:

- *Схема аксиомы знания*

$$Lp \supset p.$$

Ее обозначают буквой T . По определению знание — информация. Таким образом, схема T утверждает: «то, что известно, — верно». Эту схему добавляют к нормальной модальной системе, чтобы оператор L означал «известно». Напротив, схемы T не будет в системе аксиом, формализующих «предположение», ибо оно может быть ошибочным.

Нормальная модальная система, пополненная схемой T , перенимает имя от двух входящих в нее схем модальных аксиом: она обозначается через KT (иногда просто через \mathcal{T}).

- *Схема аксиомы позитивной интроспекции:*

$$Lp \supset LLp.$$

Она обозначается цифрой 4. Когда модальный оператор L означает «известно», то схема 4 утверждает: «если мне известно p , то я знаю, что известно p ». Если же модальный оператор L означает «предполагается», то схема 4 гласит: «если я предполагаю, что p подтверж-

дается, то я предполагаю, что я предполагаю, что p подтверждается».

Описанная схемой 4 возможность интроспекции нужна для формализации совершенного интроспективного интеллекта. Нормальная модальная система, пополненная схемами аксиом T и 4, обозначается $KT\ 4$ (или, в более классической манере, $S4$).

● *Схема аксиомы негативной интроспекции*

$$Mp \supset LMp.$$

Она обозначается цифрой 5 и в логиках знания и веры формализует совершенную негативную интроспекцию. Схема 5 эквивалентна такой: $\neg Lp \supset L\neg Lp$. Когда оператор L означает «известно», то схема 5 утверждает: «если я не знаю, что p подтверждается, то я знаю, что я не знаю, что p подтверждается». Если же оператор L означает «предполагается», то она гласит: «если я не предполагаю, что p подтверждается, то я предполагаю, что я не предполагаю, что p подтверждается».

Это свойство, конечно, чрезвычайно обременительное. Оно выражает совершенное понимание пределов нашего знания или веры. Нормальная модальная система, пополненная схемами аксиом T , 4 и 5, обозначается $KT45$ (или, в более классической манере, $S5$).

Выбор модальной системы зависит от моделируемого понятия. Если хочется охарактеризовать знания разумного субъекта, обладающего совершенной способностью к логической интроспекции относительно того, что «известно» и что «неизвестно», то следует выбрать модальную систему $S5$ (разд. 4.5). Если же желательно моделировать предположения идеально разумного субъекта (некоторые предположения которого могут оказаться ошибочными, но который обладает совершенной способностью к логической интроспекции относительно того, что он предполагает и чего не предполагает), то лучше выбрать систему

К45, называемую также *слабой S5-системой* (разд. 4.6).

Каждая из этих различных модальных систем индуцирует присущее ей синтаксическое отношение выводимости. Оно обозначается \vdash_S , где S — имя рассматриваемой модальной системы.

4.4.3. Семантика возможных миров

Модальность увеличивает выразительность классической логики и позволяет выявлять некоторые понятия с помощью специфических операторов. Семантический анализ модального выражения зависит от параметров, неявно вносимых этими операторами. Например, выражения на языке временной логики используют модальные операторы с неявной переменной, отражающей временную эволюцию (§ 3.1.13). Формула $\Box p$ эквивалентна формуле $\forall t : p(t)$. Семантический анализ этой формулы должен осуществляться с учетом неявно подразумеваемого параметра t в модальном операторе \Box .

Для этого Крипке [60] ввел метод специфического семантического анализа: *семантику возможных миров*. Модальная формула будет оцениваться в лоне некоего «универсума» различных «возможных миров» (§ 3.1.17). Точнее, анализ истинности некой модальной формулы зависит от рассматриваемого возможного мира. В примере с временной логикой различные возможные миры представляют состояние мира в различных его конкретизациях. Некое «отношение доступности» свяжет эти возможные миры между собой и укажет последовательность различных моментов, в которые рассматривается мир. Опишем кратко эту семантику.

Универсум W есть множество возможных миров, связанных *отношением доступности* R . Пусть a и b — два мира из универсума W , тогда факт доступности мира b после мира a обозначается aRb . Пара (W, R) называется *структурой*. Свойства отношения R индуцируют различные схемы модальных аксиом, *общезначимых* в рассматриваемой логике. Оценка \mathcal{V} — это отображение из $W \times \mathcal{L}$ в $\{И, Л\}$, которое для каж-

дого мира w из универсума W сопоставляет каждой пропозициональной константе из \mathcal{L} определенное значение истинности. Тройка (W, R, \mathcal{V}) называется *моделью*.

Для семантической оценки формул из \mathcal{L} желательно иметь в виду конкретный мир из универсума W вместе с рассматриваемой оценкой \mathcal{V} . Рекурсивно определяют отношение семантического следования \models между моделями и формулами языка. Запись $(W, R, \mathcal{V}) \models_w f$ означает, что f истинно в мире w для модели (W, R, \mathcal{V}) . Основные правила следующие:

- $(W, R, \mathcal{V}) \models_w \text{И}$,
- $(W, R, \mathcal{V}) \not\models_w \text{Л}$,
- $(W, R, \mathcal{V}) \models_w f$, если $\mathcal{V}(w, f) = \text{И}$ и f — пропозициональная константа из \mathcal{L} ,
- $(W, R, \mathcal{V}) \models_w f \supset g$, если $(W, R, \mathcal{V}) \models_w f$ только тогда, когда $(W, R, \mathcal{V}) \models_w g$,
- $(W, R, \mathcal{V}) \models_w Lf$, если для любого мира x универсума W при wRx имеем $(W, R, \mathcal{V}) \models_x f$.

Смысл последнего правила: формула Lf подтверждается в мире w для некоей модели (W, R, \mathcal{V}) , если формула f подтверждается во всех мирах универсума W , доступных из мира w . Оно учитывает желательные значения модального оператора L . Действительно, в логике необходимого формула Lp представляет необходимость формулы p : « p необходимо в данном мире» интуитивно означает подтверждаемость p во всех мирах, доступных из данного. С другой стороны, в логике знания формула Lp означает « p известно» и, следовательно (интуитивно), что p подтверждается во всех возможных мирах, какие только можно вообразить на основе некоего множества знаний и предположений.

Из двойственности $M \equiv \neg L \neg$ непосредственно вытекает правило

- $(W, R, \mathcal{V}) \models_w Mf$, если существует мир x из универсума W , такой что wRx и $(W, R, \mathcal{V}) \models_x f$.

Прежде чем заняться анализом истинности модальной формулы в каждом из возможных миров, введем ряд понятий.

- Формула f из \mathcal{L} общезначима в модели (W, R, \mathcal{V}) тогда и только тогда, когда f подтверждается во всех мирах этой модели, т. е. если $\mathcal{V}(w, f) = \text{И}$ для всех миров w из W , или, в символьной записи $(W, R, \mathcal{V}) \models f$.
- Формула f из \mathcal{L} общезначима в структуре (W, R) тогда и только тогда, когда f общезначима в любой модели (W, R, \mathcal{V}) . Символьная запись выглядит так: $(W, R) \models f$.
- Формула f из \mathcal{L} общезначима тогда и только тогда, когда f общезначима в любой структуре (W, R) . Символьная запись такова: $\models f$.

Укажем на связь между семантическим отношением \models и синтаксическим отношением \vdash . Существует соответствие между выбором схем основных аксиом данной формальной системы и свойствами отношения доступности между возможными мирами семантической характеристики этой системы.

Можно показать, что конкретизации

- схемы аксиомы дистрибутивности, т. е. схемы K , общезначимы,
- схемы аксиомы знания (схемы T) общезначимы в любой структуре с рефлексивным отношением доступности R ,
- схемы аксиомы позитивной интроспекции (схемы 4) общезначимы в любой структуре с транзитивным отношением доступности R ,
- схемы аксиомы негативной интроспекции (схемы 5) общезначимы в любой структуре с евклидовым¹⁾ отношением доступности R .

Из факта « f — формула \mathcal{L} » доказуемо вытекают утверждения:

- $\vdash_K f$ тогда и только тогда, когда f общезначима в любой структуре,
- $\vdash_{KT} f$ тогда и только тогда, когда f общезначима в любой структуре с рефлексивным R ,

¹⁾ Отношение R называется евклидовым, если из aRb и aRc вытекает bRc .

- $\vdash_{KT4f} f$ тогда и только тогда, когда f общезначима в любой структуре с рефлексивным и транзитивным R ,
- $\vdash_{KT5f} f$ тогда и только тогда, когда f общезначима в любой структуре с рефлексивным и евклидовым R .

Структура возможных миров семантически характеризует различные модальные системы в зависимости от свойств отношения доступности.

Например, если выбрать систему $KT45$ (она же $S5$) для аксиоматизации свойств модального оператора L , то соответствующая семантическая характеристика будет состоять из множества возможных миров, связанных между собой отношением доступности R , являющимся отношением эквивалентности. Как только что указывалось, R должно быть рефлексивно, транзитивно и евклидово (то есть отношением эквивалентности).

4.5. Немонотонные логики Мак-Дермотта

4.5.1. Введение

Немонотонные логики Мак-Дермотта и Дойла [70], [71] отличаются от логики умолчаний Рейтера (разд. 4.3) в основном по следующим трем пунктам:

- Рамки этих логик такие же, как и у модальных систем необходимости и возможности.
- Предлагаемые логические системы не формализуют множество немонотонных правил, присущих данной области применения. Системы Мак-Дермотта и Дойла являются *универсальными* аксиоматическими системами.
- Интересуются не построением отдельного расширения теории, а *присутствующими во всех ее расширениях формулами*.

Мак-Дермотт и Дойл предложили изящный метод, позволяющий избежать закливания при задании правил немонотонного вывода (см. § 4.2.4). Они

предложили неконструктивную характеристику устойчивых множеств взаимно выполнимых формул, немонокотонно выводимых из некоего набора посылок.

Эти множества суть решения некоторого уравнения, являющиеся неподвижными точками и связанные с отношением выводимости, определяемым данной немонотонной системой. Соответствующая система может рассматриваться как классическая модальная аксиоматическая система, пополненная правилом вывода *выполнимых* утверждений.

В настоящем разделе представим эту систему с критических позиций. Для начала уточним формальный язык системы вывода, предложенной Мак-Дермоттом (прежде чем представить и прокомментировать ее первое описание). Вторая версия аксиоматического описания этой системы позволит исключить заикливание в определении немонотонного правила¹⁾. В заключение раздела будут очерчены достоинства и недостатки данного подхода.

4.5.2. Язык логики Мак-Дермотта

Строим *модальный язык первого порядка* \mathcal{L} . Он является основанием немонотонной аксиоматической системы, подлежащей определению (см. также §§ 1.1.3 и 1.2.3).

- Пусть \mathcal{C} , \mathcal{V} , \mathcal{F} и \mathcal{P} — множества символов индивидных констант, переменных, функций и предикатов соответственно.
- Множество термов из \mathcal{L} определяется по индукции следующим образом. Терм из \mathcal{L} — либо константа из \mathcal{C} , либо переменная из \mathcal{V} , либо выражение $f_n(t_1, \dots, t_n)$, где f_n — n -местная функция из \mathcal{F} , t_1, \dots, t_n — термы из \mathcal{L} .
- Атомарная формула из \mathcal{L} — это выражение вида $P_n(t_1, \dots, t_n)$, где P_n — n -местный предикат из \mathcal{P} , а t_1, \dots, t_n — термы из \mathcal{L} .

¹⁾ Мы ограничимся подробным рассмотрением второй немонотонной логики Мак-Дермотта [71]; которая лучше первой, оказавшейся слишком слабой.

- Множество формул из \mathcal{L} определяется по индукции: формула из \mathcal{L} — либо атомарная формула из \mathcal{L} , либо выражение $(\neg p)$, либо выражение $(p \supset q)$, либо выражение Mr , либо выражение $(\forall v)r$. Здесь p и q — формулы из \mathcal{L} , M — модальный оператор, v — переменная из \mathcal{V} .

Будем использовать обозначения $(p \vee q)$ для $((\neg p) \supset q)$, $(p \wedge q)$ для $\neg((\neg p) \vee (\neg q))$, $(p \equiv q)$ для $((p \supset q) \wedge (q \supset p))$, $(\exists v)r$ для $\neg((\forall v)(\neg r))$ и Lp для $\neg M\neg p$, а также обычные соглашения по использованию и опусканию скобок.

Относящаяся к высказываниям часть языка \mathcal{L} будет в разд. 4.6 взята в качестве языка автоэпистемической логики.

4.5.3. Пример немонотонной аксиоматической системы

Эта немонотонная аксиоматическая система содержит три вида элементов:

- «нелогические сведения», являющиеся формулами из \mathcal{L} со статусом дополнительных аксиом,
- схемы логических аксиом,
- логические правила вывода.

Совокупность схем логических аксиом будет состоять из схем формул, аксиоматизирующих логику предикатов (см. разд. 1.2), а также из обсуждавшихся в разд. 4.4 схем модальных аксиом.

Множество правил вывода будет содержать (кроме обычных правил: *modus ponens*, *универсального обобщения* и *модальной необходимости*) специфическое правило *немонотонного вывода*.

Главная ценность исследований, проведенных Мак-Дермоттом, заключена в этом изящно сформулированном специфическом правиле. Какую модальную систему выбрать? Этот вопрос является поводом для дискуссии. К ней мы перейдем в разд. 4.6 — после представления автоэпистемической логики.

Пусть L и M — двойственные модальные операторы. Рассмотрим следующие схемы и правила, где p, q и r — произвольные формулы из \mathcal{L} .

1. Схемы классических аксиом ([53], § 2.1.9):

- (a) $p \supset (q \supset p)$,
- (b) $(p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))$,
- (c) $(\neg q \supset \neg p) \supset ((\neg q \supset p) \supset q)$,
- (d) $(\forall v) p \supset p_{v/t}$ (где терм t свободен для v в p),
- (e) $(\forall v) (p \supset q) \supset (p \supset (\forall v) q)$, если v не фигурирует в p и не связана в q .

2. Схемы модальных аксиом (разд. 4.4):

- (a) схема аксиомы знания: $Lp \supset p$,
- (b) схема аксиомы дистрибутивности: $L(p \supset q) \supset (Lp \supset Lq)$,
- (c) схема Баркан: $(\forall v) Lp \supset L(\forall v) p$,
- (d) схема позитивной интроспекции: $Lp \supset LLp$,
- (e) схема негативной интроспекции: $Mp \supset LMp$.

3. Правила вывода:

- (a) *modus ponens*: $p, p \supset q \vdash q$,
- (b) правило универсального обобщения: $p \vdash (\forall v) p$,
- (c) правило необходимости: $p \vdash Lp$,
- (d) правило немонотонного вывода: "нельзя вывести $\neg p'' \vdash Mp$."

Выбирая разные подмножества из списка схем модальных аксиом, получаем различные системы немонотонного вывода. Система, включающая лишь две первые схемы, будет называться *немонотонной \mathcal{T} -системой*. Системы, содержащие соответственно четыре первые модальные схемы и всю их совокупность, назовем *немонотонной $S4$ -системой* и *немонотонной $S5$ -системой*. Эти названия вполне соответствуют названиям модальных классических систем (разд. 4.4). Если из перечисленных немонотонных систем удалить правило немонотонного вывода, то получатся модальные классические системы \mathcal{T} , $S4$ и $S5$. Сформулированное выше правило немонотонного вывода приемлемым, вообще говоря, не является. Оно закликивает определение отношения выводимости (§ 4.2.4). До

изложения способа решения этой проблемы отметим, что первоначальная формулировка правила немонотонного вывода хорошо высвечивает тройственную роль, предположительно им выполняемую:

1. Оно позволяет выводить, что некоторое утверждение возможно, т.е. выполнимо с точки зрения логики. Применяя его, можно вывести формулы вида Mr . Искомое значение для формулы Mr есть « r возможно» или же « r выполнимо». Для осуществимости этого надо, чтобы $\neg r$ не было выводимо, что обеспечивается проверкой условия правила немонотонного вывода.

Между тем для эффективной реализации этого приписанного модальному оператору M значения надо, чтобы оно обладало свойствами модальных операторов, выраженными схемами модальных аксиом этой системы.

2. Оно косвенно позволяет принять как «истинные» всего лишь выполнимые формулы. (Например, если Mr выведено по правилу немонотонного вывода и если в данной системе содержится дополнительная аксиома $MP \supset r$, то по *modus ponens* можно вывести r .) Применение этого метода означает также переход от констатации выполнимости некоего выражения к его утверждению. Данной системе присуща в некотором смысле способность воспринимать как истинные формулы с установленной выполнимостью.
3. Оно придает данной системе немонотонный характер. Например, если эта система позволяла вывести формулу Mr (или вообще формулу q , вытекающую из Mr) и если добавлена новая аксиома, позволяющая в этой системе вывести $\neg r$, то формулу q надо отвергнуть.

Для решения проблемы заикливания в определении правила немонотонного вывода, а также проблемы характеристики множеств выводимых формул опишем неподвижные точки отображения системы вывода в множество посылок A . Интуитивно эти не-

подвижные точки представляют собой такие множества формул, что никакую дополнительную (т. е. не входящую в рассматриваемое множество) формулу нельзя вывести с соблюдением свойства выполнимости.

Возьмем одну из аксиоматических систем вывода \mathcal{D} — немонотонную \mathcal{T} -систему, либо немонотонную $S4$ -систему, либо немонотонную $S5$ -систему. Обозначим через соответствующую модальную классическую систему, получаемую удалением из \mathcal{D} немонотонного правила (т. е. S есть либо \mathcal{T} , либо $S4$, либо $S5$). Определим сначала $Th_S(A)$ как множество формул из \mathcal{L} , выводимых в системе S из множества дополнительных аксиом A , т. е.

$$Th_S(A) = \{p \in \mathcal{L} \mid A \vdash_S p\}.$$

Итак, речь идет о множестве *модальных теорем*, монотонно доказуемых в модальной классической системе S с использованием множества посылок A .

Пусть B — подмножество формул из \mathcal{L} . Обозначим через $Hyp_A(B)$ множество формул, *предположительных* относительно множества B (т. е. множество формул, выполнимых вместе с формулами из множества B , но не доказуемых в модальной системе S с использованием множества посылок A).

$$Hyp_A(B) = \{Mq \mid q \in \mathcal{L} \text{ (} q \text{ не имеет свободных переменных) и}$$

$$\neg q \notin B\} \setminus Th_S(A).$$

Нас интересуют такие множества B формул из \mathcal{L} , которые являются *неподвижными* точками оператора $Th_S(A \cup Hyp_A(\))$, т. е. решениями рекуррентного уравнения

$$B = Th_S(A \cup Hyp_A(B)).$$

Правая часть этого уравнения есть множество всех модальных следствий, выводимых из объединения множества посылок A и формул, предположительных относительно искомого множества B . Множество B является неподвижной точкой этого уравнения и называется *Немонотонным* _{A} . Значит, оно *устойчиво* и

состоит из всех модальных следствий, выводимых из объединения множества дополнительных аксиом A и множества формул, предположительных относительно этого *Немонотонного* _{A} множества.

Таким образом, решения приведенного выше рекуррентного уравнения дают (неконструктивно определяемые) *максимальные множества формул, выводимых с использованием множества посылок A и с сохранением свойства выполнимости*. Эти множества содержат все логические следствия из множества посылок A , а также все предположительные относительно них формулы.

Обозначим через $TH_S(A)$ множество *теорем*, получаемых в результате применения к множеству дополнительных аксиом A системы немонотонного вывода \mathcal{D} , причем применение это осуществляется в соответствии со следующим соотношением:

$$TH_S(A) = \mathcal{L} \cap (\cap \text{Немонотонное}_A).$$

Таким образом, статус теорем придается формулам, принадлежащим *всем* неподвижным точкам из \mathcal{D} . При отсутствии неподвижных точек $TH_S(A)$ определяется как множество всех формул из \mathcal{L} . В этом еще одно отличие от логики умолчаний, где «теоремами» были формулы из отдельной неподвижной точки (расширения) некоего множества умолчаний (см. разд. 4.3).

Отношение немонотонной выводимости, соответствующее системе вывода \mathcal{D} , обозначается через $|\sim$ и определяется следующим образом. Пусть Q_1 и Q_2 подмножества из \mathcal{L} . Имеем $Q_1 | \sim_S Q_2$ тогда и только тогда, когда $Q_2 \subseteq TH_S(Q_1)$. Множество Q_2 формул становится с данного момента множеством *теорем* для множества посылок Q_1 , если и только если любая формула из Q_2 принадлежит всем неподвижным точкам множества Q_1 .

Как говорилось в разд. 4.2, эта логическая система позволяет построить различные множества заключений (неподвижных точек) посредством варьирования порядка применения выводов. При этом будет ноль, одна или больше неподвижных точек.

4.5.4. Примеры

Для иллюстрации изложенных выше понятий и положений приведем два примера, взятых из [70].

- Пусть $A = \{Mp \supset \neg q, \quad Mq \supset \neg p\}$, где p и q — пропозициональные константы. У этого множества посылок две неподвижных точки — F_1 и F_2 . Неподвижная точка F_1 содержит $\neg p$, но не включает $\neg q$. Аналогично F_2 содержит $\neg q$, но не включает $\neg p$. Действительно, если F_1 не содержит $\neg q$, то она содержит Mq , откуда по правилу *modus ponens* с дополнительной аксиомой $Mq \supset \neg p$ вытекает, что она содержит $\neg p$. Подобное рассуждение можно применить и к F_2 .
- Пусть $A = \{Mp \supset \neg p\}$. Неподвижных точек здесь нет. Действительно, если бы неподвижная точка не содержала $\neg p$, то она содержала бы Mp . Следовательно, она содержала бы $\neg p$ в противоречии с предположением. Если бы она содержала $\neg p$, то содержала бы и Mp . Поэтому формула $Mp \supset \neg p$ была бы выполнимой. Приходим к противоречию, ибо $\neg p$ и Mp не могут одновременно фигурировать в одном выполнимом множестве.

4.5.5. Ценность логики Мак-Дермотта

Основная ценность немонотонной логики Мак-Дермотта заключена в методе неподвижной точки, используемом для характеристики устойчивых множеств заключений немонотонной системы, а также в применении модальной логики для формализации модифицируемых рассуждений.

Впрочем, выбор подходящей для рассмотрения модальной системы остается проблематичным. Желая сопоставить модальному оператору M значение *быть выполнимым*, Мак-Дермотт заметил, что наиболее приемлемой модальной логикой является та, которая соответствует немонотонной $S5$ -системе. Однако эта

логическая система обнаруживает неожиданное свойство [71]:

если $A \mid \sim_{S5} p$, то $A \vdash_{S5} p$.

Иначе говоря, нет теоремы в *немонотонной* $S5$ -системе, которая не была бы теоремой в соответствующей монотонной классической системе $S5$. Мак-Дермотт подверг тогда сомнению полезность *немонотонной* $S5$ -системы и посоветовал (не приводя абсолютно убедительных аргументов) выбрать для формализации немонотонности *немонотонную* $S4$ -систему.

В следующем разделе мы покажем, как можно перестроить логику Мак-Дермотта, привлекая моделирование идеально разумного субъекта, интроспективно рассуждающего об исходном множестве предположений. Тогда удовлетворительно решится проблема выбора модальной системы.

4.6. Автоэпистемические логики

4.6.1. Введение

Автоэпистемические логики имеют своим предметом формализацию *интроспективных* и *идеально разумных* рассуждений об исходном множестве предположений. Они позволяют осуществить формализацию выражений вида: «если я не предполагаю, что p подтверждается, то подтверждается q ».

Под идеально разумными понимаются рассуждения, идеализированные в двух аспектах: можно выводить только ожидаемые логические следствия из исходного множества предположений и все эти логические следствия надо принять во внимание. Таким образом, для рассуждений нужны неограниченные ресурсы.

Подобные рассуждения немонотонны, ибо множество основных предположений субъекта может со временем меняться, что чревато противоречиями для некоторых выводов. Такими интроспективными рассуждениями можно моделировать многочисленные виды модифицируемых рассуждений (§ 4.2.6).

Модальные логики знания и веры (разд. 4.4) подходят для формализации такого вида рассуждений. Они позволяют определить и аксиоматизировать различные эпистемические понятия (предположение, знание, обоснованное знание и т. д.) и их свойства.

Столнекер [103] и особенно Мур [78], [79], [80] развили основанный на модальной логике метод для формализации немонотонных, интроспективных и идеально разумных рассуждений.

Эту *автоэпистемическую* логику можно рассматривать как результат реконструкции немонотонной логики Мак-Дермотта (см. разд. 4.5), состоящей в замене парадигм *выводимости* и *выполнимости* на формализацию *интроспективных способностей рассуждений*. В этом контексте модальный оператор M и двойственный ему оператор L формализуют соответственно «обратное не предполагается» и «предполагается» (вместо выполнимого и выводимого).

Отметим, наконец, что автоэпистемическая логика и логики умолчаний (см. разд. 4.3) имеют одинаковую формальную выразительность [57], хотя их области применения могут быть различными.

4.6.2. Язык и семантика

Рассматриваемый здесь язык \mathcal{L}_p получается ограничением модального языка \mathcal{L} , описанного в § 4.5.2, на свою пропозициональную компоненту и использованием модального оператора L (двойственного M) для построения модальных формул. Формула вида Lq интерпретируется следующим образом: «предполагается, что q подтверждается».

Назовем *теорией* множество формул языка \mathcal{L}_p , *автоэпистемической теорией* — подмножество T из \mathcal{L}_p , представляющее какое-то *полное* и *легальное* множество предположений, которое *идеально разумный* субъект может построить на основе множества A исходных предположений. Для начала уточним, каковы семантические свойства, которыми должно обладать подобное множество формул T .

Для этого введем следующие определения [79].

- *Интерпретация высказываний* автоэпистемической теории T приписывает значения истинности формулам множества T . Приписывание подчиняется классическим правилам оценки сложных формул логики высказываний. Оно придает произвольное значение истинности пропозициональным константам и формулам вида Lp («предполагается p »).
- *Модель высказываний* автоэпистемической теории T — это интерпретация высказываний из T , в которой подтверждаются все формулы теории T .
- *Автоэпистемическая интерпретация* автоэпистемической теории T — это интерпретация высказываний из T , для которой всякая формула вида Lp подтверждается тогда и только тогда, когда p принадлежит T . Таким образом, при автоэпистемической интерпретации формула Lp («предполагается p ») подтверждается в том и только в том случае, если p принадлежит множеству T предположений данного субъекта.
- *Автоэпистемическая модель* автоэпистемической теории T — это автоэпистемическая интерпретация, в которой подтверждается всякая формула из T .

Привьем на эти семантические рассмотрения понятия *полноты* и *легальности* (вместо *выполнимости*), приспособленные к автоэпистемичности [79].

- Автоэпистемическая теория T семантически *полна* тогда и только тогда, когда она содержит все формулы, подтверждающиеся во всех автоэпистемических моделях T . (Интуитивно: T полна тогда и только тогда, когда T содержит все формулы, которые данному субъекту семантически позволено вывести в предположении истинности всех его гипотез.)
- Автоэпистемическая теория T *легальна* относительно множества A основных предположений тогда и только тогда, когда любая автоэписте-

мическая интерпретация T , являющаяся моделью A , является также и моделью теории T . (Интуитивно: это позволяет гарантировать, что предположения некоего субъекта, составляющие автоэпистемическую теорию, истинны тогда и только тогда, когда истинны основные предположения из множества A .)

4.6.3. Характеризация синтаксиса

Посмотрим, как дать синтаксическую характеристику автоэпистемическим теориям T , обладающим семантическими свойствами полноты и легальности относительно множества A исходных предположений.

Ввиду трудности конструктивной характеристики множеств выводимости формул немонотонной системы (проблема уже обсуждалась в §§4.2.3—4.2.4) дадим неконструктивное определение автоэпистемических теорий T , таких, что их максимальные и легальные множества предположений идеально разумный субъект в состоянии построить на основе множества A исходных предположений.

В этом ключе будем говорить, что автоэпистемическая теория T *устойчива*, если T есть множество формул из \mathcal{L}_p , удовлетворяющее следующим условиям:

1. Если $\{p_1, \dots, p_n\} \subseteq T$ и $\{p_1, \dots, p_n\} \vdash q$ (где \vdash представляет отношение выводимости в логике высказываний), то $q \in T$.
2. Если $p \in T$, то $Lp \in T$.
3. Если $p \notin T$, то $\neg Lp \in T$.

Первое правило утверждает, что мыслящий субъект предполагает все логические следствия и уже предполагаемого (это обязательно, если требуем идеальной разумности субъекта). Второе правило гарантирует, что формула «предполагается p » принадлежит множеству T предположений этого субъекта, если p — предположение этого субъекта. Последнее правило утверждает, что формула « p не предполагается» фигурирует в множестве T предположений того же субъекта, если формулы p там нет.

Двойственным образом определяется следующее понятие: автоэпистемическая теория T называется *теорией, основанной* на множестве дополнительных аксиом A , если все формулы из T фигурируют среди тавтологических следствий из множества $A \cup \{Lp \mid p \in T\} \cup \{\neg Lp \mid p \notin T\}$.

Справедливы следующие утверждения [79]:

- Автоэпистемическая теория T семантически *полна* тогда и только тогда, когда она *устойчива*.
- Автоэпистемическая теория T *легальна* относительно множества A основных предположений тогда и только тогда, когда она *основана* на A .

Наконец, назовем *устойчивым расширением* множества исходных предположений A множество предположений T , которое устойчиво и основано на A . Устойчивые расширения являются максимальными легальными множествами предположений, которые идеально разумный субъект в состоянии вообразить на основе исходного множества предположений.

4.6.4. Анализ немонотонной логики

Построенная Мак-Дермоттом теория немонотонного вывода обнаруживает некоторые странные особенности, а именно: каждая теорема из *немонотонной* $S5$ -системы является теоремой из *монотонной* системы $S5$ (§ 4.5.5). Этот факт можно объяснить, привлекая автоэпистемическую логику. Мак-Дермотт рассматривает неподвижные точки T системы вывода своей немонотонной логики, приложенной к множеству A дополнительных аксиом. Эти точки можно сравнить с максимальными легальными множествами предположений идеально разумного субъекта.

Определение Мак-Дермотта, ограниченное на пропозициональную компоненту, в сущности эквивалентно следующему.

- T является неподвижной точкой относительно A тогда и только тогда, когда T есть множество модальных следствий (в системе $S = \mathcal{F}$, $S4$ или $S5$) из $A \cup \{\neg Lp \mid p \notin T\}$.

Правило немонотонного вывода позволяет вывести формулу вида Mr , если формула $\neg p$ не принадлежит рассматриваемой неподвижной точке T . Иначе говоря, опираясь на отношение двойственности между модальными операторами M и L , можно с помощью этого правила осуществить вывод формулы вида $\neg Lp$, если формула p не принадлежит рассматриваемой неподвижной точке. Но тогда в логике Мак-Дермотта неподвижная точка состоит из всех модальных следствий из объединения множества формул Lp с указанной характеристикой и множества A дополнительных аксиом.

Напротив, *устойчивое расширение* T определяется следующим образом:

- T является устойчивым расширением A тогда и только тогда, когда T есть множество тавтологических следствий из $A \cup \{Lp \mid p \in T\} \cup \{\neg Lp \mid p \notin T\}$.

Легко заметить, что в определении неподвижной точки, данном Мак-Дермоттом, в совокупности аксиом нет (!) множества $\{Lp \mid p \in T\}$. Мур [79] описывает природу этой неполноты следующим образом: *«при авто-эпистемическом видении модального оператора L мыслящий субъект, использующий немонотонную логику Мак-Дермотта, всеведущ относительно всего им не предполагаемого, но может полностью игнорировать им предполагаемое»*. В самом деле, он не включает в число аксиом формулы вида «я предполагаю, что p подтверждается», которые квалифицируют положительные предположения субъекта.

Теперь относительно того, что любая теорема немонотонной $S5$ -системы является теоремой в монотонной системе $S5$. Это можно объяснить следующим образом. Немонотонная $S5$ -система содержит схему аксиомы знания, т. е. $Lp \supset p$ (см. § 4.4.2). Для авто-эпистемического анализа немонотонности эта схема кажется слишком обременительной, так как она утверждает: «все, что предполагается, истинно». Это было бы приемлемо в логике знания, но не в логике веры.

Если мы хотим построить немонотонную систему, то использование логики знания представляется неподходящим. Все выведенное не имеет больше статуса общезначимого в классическом смысле, ибо есть возможность модификации. Это кажется трудно совместимым со схемой аксиомы, требующей истинности всего предполагаемого. Поскольку ничто не позволяет выделить в системе Мак-Дермотта общезначимые утверждения и утверждения со статусом всего лишь выполнимых формул, эту схему аксиом, видимо, надо исключить.

С другой стороны, немонотонная $S5$ -система содержит также схему аксиомы $Mr \supset LMr$, применяя которую совместно со схемой аксиомы знания, можно обосновать предположение в любой формуле [79].

Следовательно, нет такой формулы, которая отсутствовала бы во всех неподвижных точках множества вспомогательных аксиом, получаемого с помощью немонотонной $S5$ -системы. В частности, не существует теоремы вида $\neg Lp$ (эквивалентной $M \neg p$) ни в какой теории, базирующейся на немонотонной $S5$ -системе, ибо такая теорема имела бы обоснование, построенное с помощью правила немонотонного вывода из логики Мак-Дермотта. Мур [79] следующим образом толкует этот факт: «идеально разумный субъект, который предполагается не совершающим ошибок, склонен предполагать всё таким образом, что внешний наблюдатель не может сделать никаких выводов относительно того, что этот субъект не предполагает».

Предлагается и еще один путь: не отвергать схему аксиомы $Mr \supset LMr$, чтобы тем самым получить немонотонную $S4$ -систему, а лучше отбросить схему аксиомы знания $Lp \supset p$, что приведет к немонотонной системе, основанной на модальной слабой $S5$ -системе¹⁾ (§ 4.4.2).

Впрочем, Мур показал, что система вывода, содержащая произвольное подмножество схем модальных аксиом, присущих слабой $S5$ -системе, дает всегда одни и те же устойчивые расширения (независимо от

¹⁾ Эта система иногда обозначается через $K45$ [15].

выбора указанного подмножества). Таким образом, автоэпистемическая логика может обойтись без явно-го упоминания схем модальных аксиом.

4.6.5. Семантика возможных миров

Описанная в § 4.6.2 семантика автоэпистемической логики имеет то достоинство, что используя ее, можно характеризовать предположения субъекта, не ависимо от того, разумен он или нет. Между тем она оказалась неудобной для использования, будучи неконструктивной в том смысле, что в ней нет правил, позволяющих оценивать предположения субъекта о сложных формулах исходя из его предположений и/или непредположений о составных частях формул. Это положение вещей вполне приемлемо для случая неразумного субъекта, ибо нельзя априори установить никакой связи между его предположениями. Напротив, предположения разумного субъекта подчиняются неким отношениям, из которых можно попытаться извлечь пользу.

С этой целью Мур [78] предложил альтернативную семантическую характеристику своей автоэпистемической логики. Эта новая семантика, основанная на понятии *возможных миров*, позволяет построить конечные модели для автоэпистемических теорий. Она дает возможность доказать существование полных и легальных относительно некоторого множества посылок автоэпистемических теорий, что было бы трудно осуществить с помощью первоначально указанной семантической характеристики.

Основным результатом, на котором базируется эта новая характеристика, является следующее утверждение:

- T есть множество формул, подтверждающихся во всех мирах S_5 -полной структуры (т. е. структуры (см. § 4.4.3) относительно системы S_5 , для которой любой возможный мир доступен из другого, неважно какого, возможного мира) тогда и только тогда, когда T является устойчивой автоэпистемической теорией [78].

Таким образом, любая автоэпистемическая интерпретация \mathcal{I} устойчивой автоэпистемической теории T может характеризоваться структурой \mathcal{K} типа $S5$ и некой оценкой \mathcal{V} . Структура возможных миров специфицирует предположения идеально разумного субъекта, тогда как оценка определяет то, что действительно подтверждается в реальном мире.

Точнее, автоэпистемическая интерпретация \mathcal{I} автоэпистемической теории T есть пара $\mathcal{I} = (\mathcal{K}, \mathcal{V})$, где

- \mathcal{K} — $S5$ -полная структура (представленная множеством своих возможных миров, каждый из которых символизирован множеством подтверждающихся позитивных и негативных пропозициональных констант),
- \mathcal{V} оценка истинности в реальном мире для пропозициональных констант из \mathcal{L}_p .

Автоэпистемическая теория T является множеством всех формул, истинных во всех мирах из \mathcal{K} .

Рассмотрим пример автоэпистемической интерпретации $\mathcal{I} = (\mathcal{K}, \mathcal{V})$ с $\mathcal{K} = \{\{q, p\}, \{q, \neg p\}\}$ и $\mathcal{V} = \{p, q\}$. $S5$ -полная структура \mathcal{K} составлена из двух возможных миров. В первом формулы q и p истинны. Во втором истинны формулы q и $\neg p$. Оценка \mathcal{V} указывает, что p и q истинны в реальном мире. \mathcal{I} — автоэпистемическая интерпретация автоэпистемической теории T , содержащей все формулы, истинные во всех мирах структуры \mathcal{K} из \mathcal{I} . В данном случае T сводится к единственной формуле q .

\mathcal{I} называется автоэпистемической моделью теории T , составленной из множества формул, истинных во всех мирах из \mathcal{K} , когда любая формула из T подтверждается в \mathcal{I} .

Второй результат дает средство проверки, является ли автоэпистемическая интерпретация \mathcal{I} для T автоэпистемической моделью для T .

- Если $\mathcal{I} = (\mathcal{K}, \mathcal{V})$ — автоэпистемическая интерпретация для T , то \mathcal{I} является автоэпистемической моделью для T тогда и только тогда, когда \mathcal{V} — элемент из \mathcal{K} , что означает выполнимость

оценки \mathcal{V} вместе с данной оценкой, задаваемой в одном из возможных миров структуры \mathcal{K} (т. е. что реальный мир — один из миров, совместимых с предположениями данного субъекта) [78].

В нашем примере \mathcal{I} — автоэпистемическая модель для T , ибо оценка \mathcal{V} реального мира выполняема вместе с оценкой, заданной в первом возможном мире структуры \mathcal{K} .

Эти две теоремы интересны тем, что они индуцируют метод проверки принадлежности формулы к устойчивому расширению исходного множества посылок.

Напомним, что *устойчивое расширение* T множества A основных предположений (посылок) есть *устойчивое* множество предположений, *основанное* на A (§ 4.6.4).

В силу первой теоремы, автоэпистемическая теория T устойчива, если ее можно представить $S5$ -полной структурой возможных миров. Чтобы автоэпистемическая теория T была *устойчивым расширением*, нужно еще, чтобы T была *основана* на множестве A исходных предположений (следовательно, чтобы T была *легальна* относительно A). Иначе говоря, любая автоэпистемическая модель посылок A должна также быть и моделью для T . В силу второй теоремы, оценка реального мира каждой из этих автоэпистемических моделей посылок должна быть выполнимой вместе с оценкой в одном из возможных миров структуры \mathcal{K} для автоэпистемической интерпретации.

Таким образом, можно предложить разрешающую процедуру для автоэпистемической логики [80]. Приведем один простой ее вариант.

Пусть A — множество посылок.

1. Строим все оценки, возможные для пропозициональных констант, появляющихся в A . Они будут характеризовать $S5$ -полные структуры возможных миров языка для A .
2. Выбираем структуры возможных миров, для которых любая формула из A подтверждается во всех мирах.

3. Для каждой из этих структур \mathcal{K} строим все автоэпистемические интерпретации $(\mathcal{K}, \mathcal{V})$, соответствующие всем оценкам \mathcal{V} пропозициональных констант, которые появляются в A .
4. Проверяем для каждой автоэпистемической интерпретации $(\mathcal{K}, \mathcal{V})$ выполнение или невыполнение утверждения «любая формула из A истинна в $(\mathcal{K}, \mathcal{V})$ тогда и только тогда, когда \mathcal{V} принадлежит \mathcal{K} ». В случае выполнения \mathcal{K} характеризует некое устойчивое расширение для A .
5. Чтобы проверить принадлежность данной формулы устойчивому расширению, представленному посредством \mathcal{K} , выясняем, подтверждается ли эта формула в \mathcal{K} .

4.6.6. Пример

Пусть множество исходных предположений $A = \{\neg Lp \supset q\}$. Оно содержит только одну формулу, означающую «если я не предполагаю p , то q подтверждается». Докажем, что:

- идеально разумный субъект, имеющий множество исходных предположений A , состоящее из одного элемента, получит устойчивое расширение T , содержащее высказывание q , но не p ,
- не может существовать ни устойчивого расширения, содержащего p , но не q , ни устойчивого расширения, содержащего одновременно p и q .

Предположим, что какое-то устойчивое расширение T для A содержит q , но не p . В этом случае $S5$ -полной структурой, связанной с множеством предположений T , будет $\mathcal{K} = \{\{q, p\}, \{q, \neg p\}\}$. (Эта структура отражает два возможных мира: в первом подтверждаются q и p , во втором — q и $\neg p$. Истинные формулы во всех мирах этой структуры соответствуют формулам, предполагаемым данным субъектом. Таким образом, T — устойчивая автоэпистемическая теория.)

Рассмотрим все автоэпистемические интерпретации для T . Они составлены из структуры возможных

миров, отражающей предположения данного субъекта, и из произвольной оценки \mathcal{V} истинности в реальном мире. В рассматриваемом языке лишь две пропозициональные константы; значит, оценок — четыре: $\{p, q\}$, $\{p, \neg q\}$, $\{\neg p, q\}$, $\{\neg p, \neg q\}$.

Из четырех автоэпистемических интерпретаций $\mathcal{I} = (\mathcal{K}, \mathcal{V})$ только первая и третья превращают в истинное основное предположение $\neg Lp \supset q$. Так как оценка \mathcal{V} для каждой из этих интерпретаций совпадает с оценкой возможного мира данной структуры, то теория T основана на множестве посылок A (и, следовательно, является устойчивым расширением A).

Предположим, что T содержит p , но не q . Структура возможных миров будет тогда $\mathcal{K} = \{\{p, q\}, \{p, \neg q\}\}$. Рассмотрим оценку $\mathcal{V} = \{\neg p, q\}$. Она служит основой автоэпистемической интерпретации $\mathcal{I} = (\mathcal{K}, \mathcal{V})$ теории T , подтверждающей A . Так как \mathcal{V} не соответствует оценке какого бы то ни было мира этой структуры, то T не может быть устойчивым расширением A .

Предположим, что T содержит p и q . Единственным возможным является мир $\{p, q\}$. Так как существует автоэпистемическая интерпретация, подтверждающая A и заданная этим возможным миром и оценкой $\mathcal{V} = \{\neg p, \neg q\}$, для которой \mathcal{V} не соответствует единственному возможному миру данной структуры, то T не может быть устойчивым расширением A .

Вообразим, что субъект увеличил свое множество основных предположений за счет присоединения формулы p . Множество посылок A теперь такое: $\{p, \neg Lp \supset q\}$. Осуществим тот же анализ, что и выше.

Предположим, что T содержит q , но не p . Тогда теория T наверняка не является устойчивым расширением A , ибо T даже не надмножество A .

Предположим, что T содержит p , но не q . Структура возможных миров, связанная с T , будет такой: $\{\{p, q\}, \{p, \neg q\}\}$. Автоэпистемическими интерпретациями для T , подтверждающими эти две формулы из A , будут лишь те, которые заданы оценками

$\mathcal{U} = \{p, q\}$ и $\mathcal{U} = \{p, \neg q\}$. Так как каждая из этих оценок соответствует оценке одного из возможных миров этой структуры, то T является устойчивым расширением A .

Предположим, что T содержит p и q . Соответствующая структура возможных миров есть $\{\{p, q\}\}$. Так как автоэпистемическая интерпретация $(\{\{p, q\}\}, \{p, \neg q\})$ подтверждает основные предположения без того, чтобы ее оценка $\{p, \neg q\}$ соответствовала оценке единственно возможного мира этой структуры, то T не может быть устойчивым расширением A .

4.6.7. Области применения

Области применения автоэпистемической логики весьма разнообразны. Она позволяет характеризовать заключения, ожидаемые от системы, способной к совершенной интроспекции. Это особенно полезно, когда запрашивают базу данных или базу знаний об их собственных пределах знаний (см., например, альтернативную формализацию в [61]).

Между тем *эффективное* применение автоэпистемической логики ограничено возможностями языка высказываний и громоздкостью разрешающей процедуры.

Подчеркиваем, что автоэпистемическая логика формализует *идеально разумные* рассуждения и является, таким образом, моделью способности к совершенному интроспективному рассуждению. В некоторых приложениях оказывается, что свойства формализуемого знания не требуют применения *полных* и *выполнимых* логических способностей. В свете подобных приложений автоэпистемическая логика выступает в роли такой модели компетенции, к которой могут сходиться используемые модели [38].

5. Формальные грамматики и логическое программирование

5.1. Формальные грамматики и логика

5.1.1. Введение

Русская фраза есть конечная последовательность русских слов. Какова бы ни была грамматическая форма этих слов, они считаются неделимыми элементами. Кроме того, множество этих слов конечно. Следовательно, фраза есть конечная последовательность элементов данного конечного множества. Однако разрешены не все сочетания слов: надо, чтобы сочетания были корректны (в соответствии с некоторым синтаксисом) и имели смысл (в соответствии с некой семантикой).

Мы способны говорить или писать корректные и даже осмысленные фразы, не выслушивая или не прочитывая предварительно уже выраженные нами фразы. Таким образом, существует механизм построения фраз по мере наших надобностей (механизм порождения). С другой стороны, существует механизм признания корректными и понимания впервые встреченных фраз (механизм распознавания). Если бы можно было точно описать этот механизм, имелся бы общий эффективный метод отбора русских фраз среди произвольных последовательностей слов. Это была бы некая теория русского языка.

Для естественных языков и поныне не удалось вполне формализовать обсуждаемый механизм, но построены его приближения. Самое известное, «грамматики структуры фразы», принадлежит Хомскому [16]. Среди них наиболее употребительны КС-грамматики (по-английски context-free), особенно для определения синтаксиса языков программирования. Подход таков:

- имеется словарь T ,
- некоторые последовательности слов (элементы

замыкания T^* словаря T) не являются «правильно составленными»,

- мы хотим описать язык L , удовлетворяющий условию $L \subset T^*$, как множество правильно составленных фраз.

Используем «правила переписывания» («продукции»). Например, *фраза составлена из подлежащего, сказуемого и определения. Подлежащее образовано местоимением и существительным*, и т. д. Пишем формулы вида:

- **Фраза-Подлежащее, Сказуемое, Определение.**
- **Подлежащее-Местоимение, Существительное.**

Далее задаем словарь (список существительных и т. д.) этого языка.

КС-грамматики образуют разрешимые формальные системы (§ 2.2.6). Это значит, что если язык L определен КС-грамматикой, то можно построить общий алгоритм распознавания фраз языка L .

Такой подход применяют при формализации логического языка. В логике синтаксис языка обычно определяется по инструкции. Это определение порождает правильно составленные выражения (формулы) языка. Сначала задается список исходных (базисных) выражений по категориям (см. § 1.1.3 — язык высказываний, § 1.2.3 — язык предикатов). Эти исходные выражения соответствуют словам словаря различных синтаксических категорий (таких, как глагол, прилагательное, существительное) естественного языка, а также терминальным символам и синтаксическим категориям формальных грамматик и языков, вводимых в этой главе.

В логике принято выделять множество «правил построения», устанавливающих способы сочетания различных синтаксических категорий для образования более сложных выражений. Эти правила применяются рекурсивно и дают рекурсивное определение бесконечного множества формул, составляющих язык (порожденный по правилам из базисных выражений). Правила построения соответствуют правилам грамматики естественного языка (вроде такого правила:

«подлежащее, за ним сказуемое и определение — есть фраза») и правилам (продукции) формальных грамматик.

В этом разделе мы введем понятия, связанные с формальными грамматиками и языками. Будет показано, что фразы этих языков можно считать приближениями (в определенном далее смысле) фраз естественного языка. Продукции формальных грамматик можно интерпретировать как логическую формализацию правил грамматики естественного языка, а фразы — как подмножество множества логических формул. Подмножество это мы получим, используя синтаксис исчисления высказываний или синтаксис исчисления предикатов и задавая множество базисных выражений (словарь) и правила построения.

Итак, формальные языки и грамматики будут преподнесены с одной стороны как некий результат, получаемый при формализации естественных языков и грамматик, а с другой — как частный случай логических языков и их синтаксисов.

Замечание. Мы используем следующие обозначения. Если Σ — множество (алфавит или словарь), то Σ^* — замыкание множества Σ (или, иначе, свободный моноид, порожденный Σ), т. е. множество всех конечных последовательностей (наборов, слов), составленных из элементов множества Σ , включая пустую последовательность. Σ^+ — это положительное замыкание множества Σ или, иначе, свободная полугруппа, порожденная Σ , т. е. множество непустых последовательностей элементов из Σ .

5.1.2. КС-грамматики

Обычный способ определения языка, будь то естественный (естественная речь) или язык программирования, состоит в задании множества *правил (продукций)*, образующих грамматику. Эти правила описывают те последовательности слов, которые считаются корректными (допустимыми) фразами языка. Грамматика позволяет осуществлять грамматический анализ фразы, а значит, и явно описать ее структуру.

Проявление интереса к КС-грамматикам было обусловлено стремлением формализовать естествен-

ные языки. КС-грамматики состоят из правил. Пример:

- | | |
|----------------------------------|---|
| (1) фраза | → глагол, группа_сущ |
| (2) группа_сущ | → местоимение, прилагательное, существительное |
| (3) группа_сущ | → местоимение, прилагательное, существительное, предлог, группа_сущ |
| (4) глагол | → "печатать" |
| (5) глагол | → "стереть" |
| (6) местоимение | → "этот" |
| (7) местоимение | → "эта" |
| (8) прилагательное ¹⁾ | → "вторая" |
| (9) прилагательное | → "первый" |
| (10) прилагательное | → "последний" |
| (11) существительное | → "пароль" |
| (12) существительное | → "строка" |
| (13) предлог | → "в" |

Правило «фраза → глагол, группа_сущ» гласит: «для формирования фразы взять слово синтаксической категории *глагол* и поставить за ним последовательность слов синтаксической категории *группа_сущ*».

Множество из правил (2) и (3) указывает два возможных способа формирования «группы_сущ». В (3) использована рекурсия. «Базовый случай» рекурсии (2) позволяет завершить определение. Правило *глагол* → «стереть» указывает на принадлежность слова из словаря к соответствующей синтаксической категории. Эта грамматика подтверждает грамматическую корректность последовательности (цепочки) слов [*стереть, этот, последний, пароль, в, этой, второй, строке*]. Действительно, эту фразу можно построить из продукций:

¹⁾ Рассматриваемые формальные грамматики беднее русской: среднего рода нет, числительные включены в прилагательные, а пары предлог-местоимение в местоимения. Будь средний род, вместо «пароль» писали бы просто «слово». Это относится и к последующим примерам. — Прим. перев.

<i>фраза</i>	→ "стереть", <i>группа_сущ</i>
<i>группа_сущ</i>	→ "этот", "последний", "пароль", "из", <i>группа_сущ</i>
<i>группа_сущ</i>	→ "эта", "вторая", "строка"

Помимо краткости представление языка грамматическими правилами очень гибко. Введение в язык нового множества фраз осуществляется путем добавления новых продукций к грамматике. Если к грамматике, описанной выше, добавим продукции:

- | | |
|-----------------------------|--------------|
| (14) <i>местоимение</i> | → "некий" |
| (15) <i>местоимение</i> | → "в этом" |
| (16) <i>прилагательное</i> | → "третий" |
| (17) <i>существительное</i> | → "символ" |
| (18) <i>существительное</i> | → "страница" |

то к языку присоединятся новые допустимые фразы, например:

[стереть, некий, первый, символ]
[стереть, этот, первый, символ, в этом, третьем, пароле]

5.1.3. Формальное определение КС-грамматики

Формализуем интуитивное понятие КС-грамматики. КС-грамматика представляет собой четверку $G = (V, T, P, S)$, где

- V — конечное множество *переменных*, часто называемых *синтаксическими категориями* или *нетерминальными символами*,
- T — конечное множество (непересекающееся с V) *терминальных символов (слов)*,
- P — конечное множество *правил (продукций)* вида $A \rightarrow \alpha$, где A — элемент из V и α — последовательность (цепочка) символов множества $V \cup T$,
- S — особая переменная, называемая «начальной переменной» или «исходным символом».

В нашем примере:

- $V = \{\text{группа_сущ, глагол, местоимение, прилагательное, существительное, предлог, фраза}\},$
- $T = \{\text{печатать, стереть, этот, эта, некий, первый, вторая, третий, последний, символ, пароль, строка, страница, в, в этом}\},$
- $S = \text{фраза}.$

Язык строится с помощью грамматики и интерпретации каждой из его продукций как *правила переписывания*¹⁾. Продукция состоит из *заголовка* (слева от символа \rightarrow) и *тела* (справа от \rightarrow). Переписать формулу — это значит «заменить в формуле заголовок продукции ее телом». Продукции грамматики из § 5.1.2 можно интерпретировать следующим образом:

- (1) переписать [фраза] на [глагол, группа_сущ],
- (5) переписать [глагол, группа_сущ] на [«стереть», группа_сущ]
- (2) переписать [«стереть», группа_сущ] на [«стереть», местоимение, прилагательное, существительное],
- (6), (10), (11) переписать [«стереть», местоимение, прилагательное, существительное] на [«стереть», «этот», «последний», «пароль»].

На множестве $(V \cup T)^*$ введем отношения²⁾ \Rightarrow и \Rightarrow^* . Если $A \rightarrow \beta$ — продукция из P , α и γ — последовательности из множества $(V \cup T)^*$, то $\alpha A \gamma \Rightarrow \alpha \beta \gamma$. Иначе это можно сформулировать так: «продукция $A \rightarrow \beta$ применяется к последовательности $\alpha A \gamma$ для получения последовательности $\alpha \beta \gamma$ » или « $\alpha \beta \gamma$ получается непосредственно из $\alpha A \gamma$ в соответствии с

¹⁾ Иное название — *правило подстановки*. — Прим. перев.

²⁾ Соответственно «*непосредственное следование*» и «*следование*». — Прим. перев.

грамматикой G ». Две последовательности связаны отношением \Rightarrow , когда вторая получается из первой применением некой продукции.

Пусть $\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \dots, \alpha_{m-1} \Rightarrow \alpha_m$,

где $\alpha_1, \alpha_2, \dots, \alpha_m$ — последовательности из $(V \cup T)^*$. Эта последовательность отношений \Rightarrow записывается короче: $\alpha_1 \stackrel{*}{\Rightarrow} \alpha_m$. Таким образом, отношение $\stackrel{*}{\Rightarrow}$ — это рефлексивное и транзитивное замыкание отношения \Rightarrow .

Последовательность слов можно считать синтаксически корректной фразой (по отношению к правилам грамматики), интерпретируя каждую продукцию как правило *обратного переписывания*: «заменить в формуле тело продукции ее заголовком». Только что отмеченная двойственность порождения и распознавания является классической в теории автоматов и языков (разд. 5.2).

Язык L , порождаемый (распознаваемый) КС-грамматикой, есть множество последовательностей (слов), которые:

- состоят лишь из терминальных символов,
- можно породить, начиная с S .

Формально:

$$L = \{w \mid w \in T^* \text{ и } S \stackrel{*}{\Rightarrow} w\}.$$

Это множество образует *КС-язык, порождаемый грамматикой G* .

Следующие последовательности являются фразами языка, порождаемого КС-грамматикой из § 5.1.2.

[стереть, этот, второй, пароль],
 [печатать, эту, вторую, строку],
 [стереть, этот, второй, пароль в, этой, третьей, строке],
 [стереть, эту, вторую, строку, в, этой, третьей, строке].

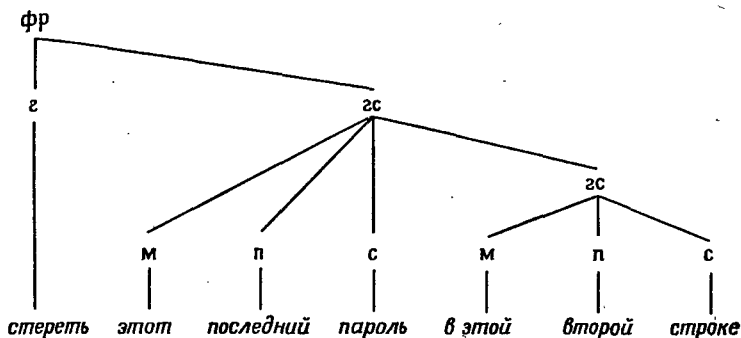


Рис. 5.1. Синтаксическое дерево.

Полезно представлять порождаемые продукциями фразы в виде *синтаксических* (или *грамматических*) *деревьев*. Каждый узел синтаксического дерева помечен либо переменной, либо терминальным символом грамматики. На рис. 5.1 изображено грамматическое дерево фразы «стереть этот последний пароль в этой второй строке». Вершина дерева называется *корнем* и помечена начальным символом грамматики. *Листья* (или *висячие узлы*) дерева помечены терминальными символами. Порождаемая фраза соответствует последовательности терминальных символов, сопоставленных листьям, причем листья проходятся слева направо. Все промежуточные узлы представляют продукции, используемые при построении правильных выражений, начинающихся с символа, приписанного корню. Таким образом, синтаксическое дерево описывает процесс построения правильного выражения. Если внутренний узел помечен символом A , а следующие за ним имеют метки X_1, X_2, \dots, X_p , то это означает, что при построении правильного выражения была применена продукция $A \rightarrow X_1, X_2, \dots, X_p$.

Дерево на рис. 5.1 представляет последовательность продукций, примененных для построения фразы, начиная с символа *фраза*. Его еще можно записать в виде выражения со скобками (как список):

$\phi_r (z(\text{стереть}), zc(m(\text{этот}), n(\text{последний}), c(\text{пароль}), zc(m(\text{в этой}), n(\text{второй}), c(\text{строке})))$.

Этот список состоит из метки $фр$, за которой идут списки для следующих за ней узлов (здесь это узлы $г$ и $гс$). Например, подсписок ($гс(м(в этой), n(второй), с(строке))$) представляет поддерево группы существительного $гс$.

Если переменные формальной грамматики соотнесены синтаксическим категориям грамматики естественного языка, то синтаксическое дерево отражает грамматический анализ фразы. Дерево на рис. 5.1 дробит фразу на глагол и две вложенные группы существительных, каждая из которых содержит местоимение, прилагательное и существительное. Грамматический анализ является основой семантического анализа.

Для КС-грамматики $G = (V, T, P, S)$ синтаксическое дерево удовлетворяет следующим условиям:

- каждый узел помечен символом из $V \cup T$,
- корень помечен символом S ,
- внутренние узлы помечены элементами из V ,
- листья помечены элементами из T ,
- если узел n и следующие за ним узлы n_1, n_2, \dots, n_p — помечены соответственно символами A, X_1, X_2, \dots, X_p , то $A \rightarrow X_1, X_2, \dots, X_p$ является продукцией из P , где категории X_1, \dots, X_p упорядочены так же (слева направо), как узлы n_1, n_2, \dots, n_p в рассматриваемом дереве.

Синтаксическое дерево позволяет легко описать построение фразы в грамматике G . Некоторые фразы можно анализировать многими различными способами. Фраза *двусмысленна*, если ей может соответствовать несколько синтаксических деревьев.

5.1.4. КС-грамматика и хорновские дизъюнкты

В § 1.1.17 множество хорновских дизъюнктов интерпретировалось как КС-грамматика. Напомним, что хорновский дизъюнкт — это дизъюнкция литер, в которой не более одной позитивной литеры, например:

$$s \vee \neg p \vee \neg q \vee \neg r.$$

Запишем эту формулу в импликативной форме:

$$s : - p, q, r$$

т. е. « s истинно, если p , q и r истинны». Из этой записи видна аналогия между приведенным хорновским дизъюнктом и нижеследующим правилом грамматики:

$$s \rightarrow p, q, r$$

(т. е. « s правильно построена, если p , q и r правильно построены»). Действительно, проинтерпретируем эту продукцию как логическую формулу; символ переписывания \rightarrow соответствует импликации « $:-$ » (\supset), а запятые — конъюнкциям. Элементы множества $V \cup T$ этой грамматики интерпретируются как атомы логики высказываний. Однако есть важное отличие: в продукции фиксирован порядок элементов (тела), а в дизъюнкте нет. Например (см. § 5.1.2), смысл продукций

фраза \rightarrow *глагол*, *группа_сущ*

фраза \rightarrow *группа_сущ*, *глагол*

различен: в соответствии с первой продукцией фраза состоит из глагола, за которым следует группа существительного, а вторая продукция говорит о том, что во фразе сначала идет группа существительного, а за ней — глагол. В логике обе эти продукции интерпретируются одним дизъюнктом ¹⁾:

$$\text{фраза} \vee \neg \text{глагол} \vee \neg \text{группа_сущ}$$

Чтобы сделать логический формализм эквивалентным грамматическому, нужно ввести связку «следует за», присущую правилам грамматики. Формализовать эту связку можно с помощью понятия «списка», рекурсивное определение которого выглядит так: *список* состоит из первого элемента (*головы*) и следующего за ним, возможно пустого списка (*хвоста*). Список можно записывать в виде последовательности

¹⁾ Ибо конъюнкция (обозначения запятой выше) и дизъюнкция (присутствующая ниже) коммутативны (см. группу формул (1.4) в § 1.1.9). — *Прим. перев.*

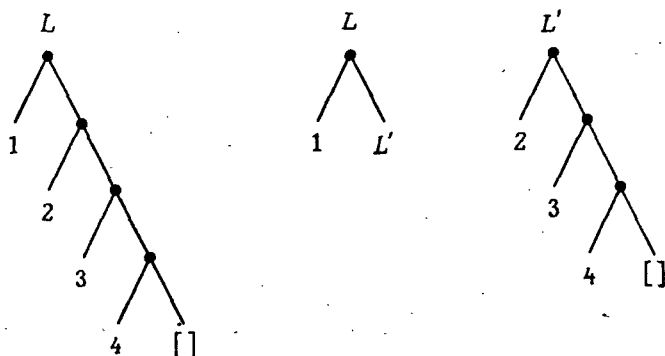
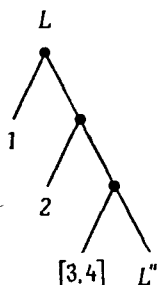


Рис. 5.2. Построение списков.

Рис. 5.3. Построение списка $L = [1, 2, [3, 4] | L'']$.

символов (элементов списка), помещенных в квадратные скобки и разделенных запятыми, например $L = [1, 2, 3, 4]$.

Используются и другие обозначения. Пусть $|$ обозначает символ построения списка из головы и хвоста. На рис. 5.2 список $[1, 2, 3, 4]$ представлен в виде $L = [1 | L']$, где $L' = [2, 3, 4]$ — хвост списка L . На рис. 5.3 оператор $|$ строит список $L = [1, 2, [3, 4] | L'']$ из элементов 1, 2, $[3, 4]$ и списка L'' .

Оператор $|$ построения списка можно непосредственно использовать для формализации связки «следует за» для тел продукций, составленных из терминальных символов и завершаемых единственным нетерминальным. (В § 5.2.2 показывается, что такое правило характеризует «регулярные грамматики».)

В § 6.4.2 исследуется общий случай правил с произвольным порядком терминальных и нетерминальных символов в теле продукции.

Например, «следование за» формализуется глаголом группы существительного посредством списка из головы (глагол G) и хвоста (группа-сущ GS): $[G|GS]$.

Понятие списка подсказывает нам мысль связать с правилом грамматики

фраза \rightarrow *глагол*, *группа-сущ*

хорновский дизъюнкт из логики предикатов

фраза ($G | GS$) :— *глагол* ($[G]$), *группа-сущ* (GS)

Он состоит из предикатов и его аргументы учитывают следование группы существительного за глаголом. Словесное описание его таково: «список $[G|GS]$ является фразой, если G — глагол, а GS — группа существительного». Итак, переход от хорновских дизъюнктов логики высказываний к хорновским дизъюнктам логики предикатов завершает интерпретацию правил КС-грамматики в логических терминах.

Продукции (1) — (13) из § 5.1.2 переходят в хорновские дизъюнкты логики предикатов (1') — (13'). Каждый нетерминальный символ грамматики замещается предикатом, аргумент которого обозначается частью того выражения, которое он представляет. Всякий терминальный символ грамматики появляется в качестве аргумента предиката, отвечающего левой части правила. Правая часть соединяет аргументы левой части в порядке их появления в эквивалентной продукции:

- | | |
|--|--|
| (1') фраза ($[G GS]$) | :— глагол ($[G]$), группа-сущ (GS) |
| (2') группа-сущ ($[M, П, C]$) | :— местоимение ($[M]$),
прилагательное ($[П]$),
существительное ($[C]$) |
| (3') группа-сущ ($[M, П, C, Пред GS]$) | :— местоимение ($[M]$),
прилагательное ($[П]$),
существительное ($[C]$),
предлог ($[Пред]$),
группа-сущ (GS) |

- (4') глагол ([печатать])
- (5') глагол ([стереть])
- (6') местоимение ([этот])
- (7') местоимение ([эта])
- (8') прилагательное ([вторая])
- (9') прилагательное ([первый])
- (10') прилагательное ([последний])
- (11') существительное ([пароль])
- (12') существительное ([строка])
- (13') предлог ([в])

5.1.5. ОК-грамматики

В § 5.1.4 мы представляли КС-грамматику набором хорновских дизъюнктов логики предикатов (с одним аргументом). Была установлена следующая эквивалентность:

Продукции КС-грамматики



Хорновские дизъюнкты из одноместных предикатов

КС-грамматики имеют ряд ограничений. В частности, как подсказывает их название, они не позволяют «учитывать контекст». Поясим сказанное на примере из § 5.1.2. Правила переписывания из § 5.1.3, примененные к продукциям нашей грамматики, порождают фразы

- стереть эту последнюю строку,
- стереть эту вторую строку в этой второй строке.

Первая синтаксически некорректна: нарушено согласование родов. Вторая синтаксически корректна, но семантически абсурдна (бессмысленна). Есть средство избежать фраз-паразитов, оставаясь в рамках формализма КС-грамматик. Однако для оздоровления ситуации часто приходится вводить много дополнительных правил учета всех согласований, исключений и т. д. Разумнее модифицировать формализм КС-грамматик в более мощный, позволяющий учитывать

контекст и сохраняющий простоту исходного формализма.

Введем формализм *грамматик определенных клауз* (ОК-грамматик, ОКГ¹⁾, по-английски definite clause grammar, DCG). Покажем, как переход от КС-грамматик (КСГ) к ОКГ позволяет исключить из языка нежелательные фразы наподобие цитированных. Затем формализуем понятие ОКГ.

Напомним, что КС-правило

группа_сущ → *местоимение*, *прилагательное*,
существительное

представимо дизъюнктом

группа_сущ ($[M, P, C]$): — *местоимение* ($[M]$),
прилагательное ($[P]$),
существительное ($[C]$)

Введем согласование родов (женский или мужской) во фразах языка. Для этого введем контекстуальный аргумент (K) в элементах правила:

группа_сущ → *местоимение* (K), *прилагательное* (K),
существительное (K), *группа_сущ*

Этот аргумент является логической переменной, пробегающей некое множество значений. Для случая согласования родов это множество состоит из элементов «женский» и «мужской», рассматриваемых как логические константы. Иными словами, при переписывании переменная K заменяется на константу, что обеспечивает согласование родов в трех членах продукции. Эта замена соответствует правилу унификации логики предикатов (§ 1.2.13).

Таким образом, добавление аргументов в правило КС-грамматики приводит к правилу ОК-грамматики. Значит, правила переписывания ОКГ соответствуют таковым в КСГ (§ 5.1.3), но при этом добавляется логическая унификация. Подобное уже встречалось в гл. 1, когда резолюция исчисления предикатов (§ 1.2.14) получалась из резолюции исчисления высказываний (§ 1.1.12) посредством добавления унификации.

¹⁾ Или, иначе, *грамматик определенных дизъюнктов* (ГОД). — Прим. перев.

В свою очередь продукцию ОК-грамматики можно представить хорновским дизъюнктом

группа_сущ ($[M, P, C]$):— местоимение ($K, [M]$),
прилагательное ($K, [P]$),
существительное ($K, [C]$).

Отметим, что хорновские дизъюнкты, составленные из многоместных предикатов, интерпретируются как продукции ОКГ.

Перепишем пример из § 5.1.2, используя правил ОКГ, обеспечивающие согласование родов:

- | | |
|--------------------------------|---|
| (1'') фраза | → глагол,
группа_сущ |
| (2'') группа_сущ | → местоимение (K),
прилагательное (K),
существительное (K). |
| (3'') группа_сущ | → местоимение (K),
прилагательное (K),
существительное (K),
предлог,
группа_сущ |
| (4'') глагол | → [печатать] |
| (5'') глагол | → [стереть] |
| (6'') местоимение (мужск) | → [этот] |
| (7'') местоимение (женск) | → [эта] |
| (8'') прилагательное (мужск) | → [второй] |
| (9'') прилагательное (женск) | → [вторая] |
| (10'') прилагательное (мужск) | → [последний] |
| (11'') прилагательное (женск) | → [последняя] |
| (12'') существительное (мужск) | → [символ] |
| (13'') существительное (мужск) | → [пароль] |
| (14'') существительное (женск) | → [строка] |
| (15'') существительное (женск) | → [страница] |
| (16'') предлог | → [в] |

Терминальные символы, употреблявшиеся с кавычками в КС-грамматике из § 5.1.2, здесь помещены в квадратные скобки как списки (одноэлементные) формализма ОКГ. Правило местоимение (мужск) → [этот] соответствует дизъюнкту местоимение (мужск, [этот]). Синтаксическая категория группа_сущ должна удовлетворять согласованию родов. Поэтому

в самой полной общности в продукциях (2'') и (3'') следовало бы писать *группа_сущ*(K) вместо *группа_сущ*. Этого можно избежать, используя неопределенную форму глаголов (как в нашем примере).

Фраза «стереть эту последнюю строку» не может быть построена по правилу (2''), ибо невозможно осуществить унификацию аргументов, сопоставленных местоимению *эта*, прилагательному *последний* и существительному *строка*: одно слово мужского рода, два другие — женского.

Аргументами в правилах ОК-грамматик можно моделировать синтаксические контексты не только для согласований рода или числа. Подчас можно учесть и семантические контексты. Обогатим формализм ОКГ для исключения абсурдных фраз. С этой целью обобщим формализм КГС в ином направлении. Помимо списков переменных и терминальных символов, правые части продукций будут содержать «обращения к процедурам», записываемые в фигурных скобках. Эти обращения налагают на продукты условия применимости для введения контекстуальных сведений в грамматику.

Проиллюстрируем обращение к процедуре на примере из § 5.1.2, когда КС-грамматика порождала и распознавала такие фразы, как следующая: «стереть эту вторую строку в этой второй строке». Эта фраза грамматически допустимая, синтаксически корректная, но семантически абсурдная. Заменяем часть правил КС-грамматики на соответствующие правила ОК-грамматики:

- | | |
|--|---|
| (1''') фраза | → глагол,
<i>группа_сущ</i> (X) |
| (2''') <i>группа_сущ</i> (Y ₁) | → местоимение (K),
прилагательное
(K), существи-
тельное (K, Z ₁),
{Y ₁ < Z ₁ } |
| (3''') <i>группа_сущ</i> (Y ₂) | → местоимение (K),
прилагательное
(K), существи-
тельное (K, Z ₂), |

- $\{Y_2 < Z_2\}$,
 предлог,
 группа_сущ (Z_2)
- (12''') существительное (мужс, 1) \triangleright [символ]
 (13''') существительное (мужс, 2) \triangleright [пароль]
 (14''') существительное (жен, 3) \rightarrow [строка]
 (15''') существительное (жен, 4) \rightarrow [страница]

Первоначальная ОКГ приобрела две новые характеристики. Во-первых, в правиле (2''') появилось условие $\{Y_1 < Z_1\}$. Оно проверится для аргументов Y_1 и Z_1 до применения правила. Во-вторых, аргумент (со значениями 1, 2, 3, 4) формализует семантическую иерархию следующей последовательности элементов: *символ*, *пароль*, *строка* и *страница* — их вложенность друг в друга.

Покажем, что эти характеристики дают возможность предотвратить порождение и/или распознавание абсурдных фраз, таких, как «стереть эту последнюю строку в этой последней строке». Пусть в продукции группа_сущ (Y_2) переменные местоимение, прилагательное и предлог заменены константами:

группа_сущ (Y_2) \rightarrow [эта, последняя], существительное
 (жен, Z_2), $\{Y_2 < Z_2\}$, [в],
 группа_сущ (Z_2)

По правилу переписывания существительное (жен, 3) \rightarrow [строка] меняем выше существительное (жен, Z_2) на строку, поскольку Z_2 принимает значение 3. После переписывания имеем:

группа_сущ (Y_2) \rightarrow [эта, последняя, строка], $\{Y_2 < 3\}$,
 [в], группа_сущ (3)

По правилу переписывания группа_сущ (Y_1) \rightarrow [эта, последняя], существительное (жен, Z_1), $\{Y_1 < Z_1\}$ меняем группу_сущ (3) на [[эта, последняя], существительное (жен, Z_1), $\{3 < Z_1\}$], ибо Y_1 принимает значение 3. Имеем:

группа_сущ (Y_2) \rightarrow [эта, последняя, строка], $\{Y_2 < 3\}$,
 [в, этой, последней], существительное (жен, Z_1), $\{3 < Z_1\}$

Наконец, ищем замену в словаре для *существительного* (жен, Z_1) с учетом соотношения $Z_1 > 3$. Единственно подходящее правило переписывания *существительное* (жен, 4) \rightarrow [страница] дает:

группа_сущ (Y_2) \rightarrow [эта, последняя, строка], $\{Y_2 < 3\}$,
[в, этой, последней, странице] ¹⁾

Эта продукция порождает или распознает лишь фразы вида: *глагол* | [эта, последняя, строка, в, этой, последней, странице]. Вызов процедуры $\{3 < Z_1\}$ исключил абсурдные фразы.

Итак, формализм ОК-грамматик превосходит формализм КС-грамматик в двух направлениях:

- синтаксические категории могут быть многоместными предикатами,
- в телах продукций допустимы несколько обращений к процедурам (заключенные в фигурные скобки).

Расширение формализма ввело в грамматику кое-какую зависимость от контекста. Кроме того, ОК-грамматики представляют структуру фразы благодаря аргументам и унификации. Они не только воспринимают и порождают фразы, но и представляют синтаксические составляющие в форме дерева для «понимания» (помимо «распознавания») фраз. Пример построения синтаксического дерева см. в § 5.1.7.

5.1.6. ОК-грамматики и логика

Переход от КСГ к множеству дизъюнктов осуществляется непосредственно. В § 5.1.4 правило

группа_сущ \rightarrow местоимение, прилагательное, существительное

мы уподобили дизъюнкту

группа_сущ ($[M, P, C]$) :— местоимение ($[M]$),
прилагательное ($[P]$),
существительное ($[C]$)

¹⁾ Собственно по-русски следовало бы писать «на ... странице», но мы стеснены заданными формальными рамками. — Прим. перев.

Формализм ОКГ и логика тоже связаны непосредственным образом. Правило (3'') из § 5.1.5

группа_сущ (Y_2) \rightarrow местоимение (K), прилагательное (K), существительное (K, Z_2), $\{Y_2 < Z_2\}$, предлог, *группа_сущ* (Z_2)

подобно хорновскому дизъюнкту

группа_сущ ($Y_2, [M, П, С, Пред|ГС]$): — местоимение ($K, [M]$), прилагательное ($K, [П]$), существительное ($K, Z_2 [С]$), $Y_2 < Z_2$, предлог ($Пред$), *группа_сущ* ($Z_2, ГС$)

Синтаксические категории, вроде *существительное* (K, Z_2) и *группа_сущ* (Y_2) интерпретировались нами, как логические предикаты. Обращение к процедуре $Y_2 < Z_2$ тоже можно интерпретировать как логический предикат, принимающий значение И при $Y_2 < Z_2$, Л в противном случае.

Этот предикат неравенства не получает дополнительных аргументов, когда правило ОКГ берется в клаузуальной форме (т. е. в виде дизъюнкта). Предикаты в фигурных скобках не соответствуют никаким словам или частям фразы. Их единственное назначение — наложить ограничения на управляемые грамматикой параметры. Перевод формализма ОКГ в формализм хорновских дизъюнктов логики предикатов является непосредственным и его можно автоматизировать:

- каждая продукция соответствует хорновскому дизъюнкту;
- заголовок продукции становится позитивным предикатом хорновского дизъюнкта;
- каждый член тела продукции становится негативным предикатом; этот предикат получает дополнительный аргумент, если член продукции не заключен в фигурные скобки;
- позитивный предикат получает новый аргумент в виде списка дополнительных аргументов в порядке появления соответствующих членов в теле продукции.

Этот активный механизм грамматического формализма устроен по принципу переписывания. Именно этот механизм порождает или распознает фразы языка, описанного данной грамматикой.

Покажем, что переход от грамматики к логике преобразует это переписывание в дедукцию, точнее, в резолюцию (§§ 1.1.12, 1.2.14).

Каждое правило грамматики интерпретируется как хорновский дизъюнкт. Множество таких дизъюнктов представляет грамматику как множество гипотез, необходимых для доказательства некой логической теоремы, например: *данный список слов есть фраза языка*. Более строго: *данный список слов есть логическое следствие правил грамматики*.

Если G_1, \dots, G_n — правила грамматики, записанные в виде хорновских дизъюнктов, а L — логическое выражение списка слов, то доказательство теоремы сводится к логическому подтверждению общезначимости следующих (эквивалентных) формул (см. § 1.1.9):

$$(G_1 \wedge \dots \wedge G_n) \supset L,$$

$$(G_1 \wedge \dots \wedge G_n \wedge \neg L) \equiv \text{Л},$$

$$(\neg G_1 \vee \dots \vee \neg G_n \vee L) \equiv \text{И}.$$

Запишем КСГ из § 5.1.2 на языке дизъюнктов логики предикатов в соответствии с § 5.1.4. Затем преобразуем имплекативную форму хорновских дизъюнктов в эквивалентную дизъюнктивную. Получим следующие выражения:

$$(1) \text{ фраза } ([Г | ГС]) \vee \neg \text{ глагол } ([Г]) \vee \vee \neg \text{ группа_сущ } (ГС)$$

$$(2) \text{ группа_сущ } ([М, П, С]) \vee \neg \text{ местоимение } (К, [М]) \vee \vee \neg \text{ прилагательное } (К, [П]) \vee \neg \text{ существительное } (К, [С])$$

$$(3) \text{ глагол } ([\text{стереть}])$$

$$(4) \text{ местоимение } (\text{мужс}, [\text{этот}])$$

$$(5) \text{ прилагательное } (\text{мужс}, [\text{последний}])$$

$$(6) \text{ существительное } (\text{мужс}, [\text{пароль}])$$

и т. д.

Убедимся в том, что: список слов [стереть, этот, последний, пароль] является грамматически допустимой фразой. Это сводится к проверке того, что дизъюнкт

(7) фраза ([стереть, этот, последний, пароль])

является логическим следствием данной грамматики, т. е., в нашем случае, дизъюнктов (1) — (6). Таким образом, надо доказать, что конъюнкция (произведение) дизъюнктов (1) — (6) и отрицания дизъюнкта (7) ложна (равна нулю). Для каждого этапа доказательства выпишем последовательно

— результат резолюции,
— используемые в резолюции дизъюнкты,
— примененные унификации (в случае необходимости).

- (8) \neg глагол ([стереть]) \wedge \neg группа_сущ ([этот, последний, пароль]); резолюция: (1) и \neg (7), унификация: $\{(Г, \text{стереть}), (ГС, [\text{этот, последний, пароль}])\}$;
- (9) \neg группа_сущ ([этот, последний, пароль]); резолюция: (3) и (8);
- (10) \neg местоимение ($K, [\text{этот}]$) \vee \neg прилагательное ($K, [\text{последний}]$) \vee \neg существительное ($K, [\text{пароль}]$); резолюция: (2) и (9), унификация: $\{(М, \text{этот}), (П, \text{последний}), (С, \text{пароль})\}$;
- (11) \neg прилагательное (мужс, [последний]) \vee \neg существительное (мужс, [пароль]); резолюция: (4) и (10), унификация: ($K, \text{мужс}$);
- (11) Л, резолюция: (5), (6) и (11).

Эта процедура подтверждает равенство нулю произведения дизъюнктов (1) — (6), \neg (7). Следовательно, список

[стереть, этот, последний, пароль]

является допустимой фразой грамматики, представленной дизъюнктами (1) — (6).

5.1.7. Построение синтаксического дерева

В § 5.1.3 синтаксическое дерево на рис. 5.1 представлено списком со скобками:

фр (г (стереть), гс (м (этот), н (последний), с (пароль), гс (м (в этой), н (второй), с (строке))))).

Опишем процесс получения списка и (следовательно) дерева в ОК-грамматике из § 5.1.5, используя новые обозначения для аргументов. Синтаксическое дерево получится из последней унификации процедуры доказательства, приведенной в § 5.1.6. Начнем с переписывания нашей ОК-грамматики, добавляя к ней некоторые аргументы:

- (1) фраза (*фр* (*ГГ*, *Гс*)) → глагол (*ГГ*), группа_сущ (*0*, *Гс*)
- (2) группа_сущ (*Y₁*, *гс* (*Мс*, *Пл*, *СС*))) → местоимение (*К*, *Мс*),
прилагательное (*К*, *Пл*),
существительное (*К*, *Z₁СС*), {*Y₁* < *Z₁*}
- (3) группа_сущ (*Y₂*, *гс* (*Мс*, *Пл*, *СС*, *Предл*, *гс* (*Гс*))) → местоимение (*К*, *Мс*),
прилагательное (*К*, *Пл*),
существительное (*К*, *Z₂СС*), {*Y₂* < *Z₂*},
предлог (*Предл*),
группа_сущ (*Z₂*, *Гс*)
- (4) глагол (*г* (*стереть*))) → [*стереть*]
- (5) глагол (*г* (*печатать*))) → [*печатать*]
- (6) местоимение (*мужс*, *м* (*этот*))) → [*этот*]
- (7) местоимение (*жен*, *м* (*эта*))) → [*эта*]
- (8) прилагательное (*мужс*, *н* (*второй*))) → [*второй*]
- (9) прилагательное (*жен*, *н* (*вторая*))) → [*вторая*]
- (10) прилагательное (*мужс*, *н* (*последний*))) → [*последний*]

- (11) *прилагательное*
(жен, n (последняя)) \rightarrow [последняя]
(12) *существительное*
(мужс, 1, с (символ)) \rightarrow [символ]
(13) *существительное*
(мужс, 2, с (пароль)) \rightarrow [пароль]
(14) *существительное*
(жен, 3, с (строка)) \rightarrow [строка]
(15) *существительное*
(жен, 4, с (страница)) \rightarrow [страница]
(16) *предлог* (пред (v)) \rightarrow [v]

Если докажем, что дизъюнкт

- (17) *фраза* (X , [стереть, этот, последний, пароль])

является логическим следствием правил этой грамматики, то результатом последней унификации для переменной X будет скобочное представление синтаксического дерева списка [стереть, этот, последний, пароль]. Перепишем некоторые правила грамматики на логический лад:

- (1') *фраза*(фр ($ГГ$, $Гс$), [$Г$ | $ГС$]) $\vee \neg$ *глагол*($ГГ$, [$Г$]) \vee
 \neg *группа_сущ* (0 , $Гс$, $ГС$)
(2') *группа_сущ* (Y_1 , $гс$ ($Мс$, $Пл$, $СС$), [$М$, $П$, $С$]) \vee
 \neg *местоимение* ($К$, $Мс$, [$М$]) $\vee \neg$ *прилагательное* ($К$, $Пл$, [$П$]) $\vee \neg$ *существительное* ($К$, Z_1 , $СС$, [$С$]) $\vee Y_1 < Z_1$
(4') *глагол* ($г$ (стереть), [стереть])
(6') *местоимение* (мужс, $м$ (этот), [этот])
(10') *прилагательное* (мужс, n (последний), [последний])
(13') *существительное* (мужс, 2, с (пароль), [пароль])
(17') *фраза* (X , [стереть, этот, последний, пароль])

Этапы доказательства следующие:

- (18') *группа_сущ* (Y_1 , $гс$ ($Мс$, $Пл$, с (пароль)), [$М$, $П$, пароль]) $\vee \neg$ *местоимение* (мужс, $Мс$ [$М$]) $\vee \neg$ *прилагательное* (мужс, $Пл$, [$Пл$]) $\vee Y_1 < 1$,
резолюция: (2') и (13'),
унификация: $\{K, мужс\}, \{Z_1, 1\},$
 $\{СС, с (пароль)\}, \{[C], [пароль]\}$;

- (19') группа_сущ ($Y_1, гс (Мс, n (последний), с (пароль)), [М, последний, пароль] \vee \vee \neg местоимение (мужс, Мс, [М]) \vee Y_1 < 1$,
резолуция : (10') и (18'),
унификация : $\{(Пл, n (последний)), ([П], [последний])\}$;
- (20') группа_сущ ($Y_1, гс (м (этот), n (последний), с (пароль)), [этот, последний, пароль] \vee Y_1 < 1$,
резолуция : (6') и (19'),
унификация : $\{(Мс, м (этот)), ([М], (этот))\}$;
- (21') фраза (фр ($ГГ, гс (м (этот), n (последний), с (пароль))$), $[Г | [этот, последний, пароль]] \vee \vee \neg глагол (ГГ, [Г]) \vee 0 < 1$,
резолуция : (1') и (20'),
унификация : $\{(Гс, гс (м (этот), n (последний), с (пароль))), (Y_1, 0), (ГС, [этот, последний, пароль])\}$;
- (22') фраза (фр ($г (стереть), гс (м (этот), n (последний), с (пароль))$),
 $[стереть, этот, последний, пароль]$),
резолуция : (4') и (21'),
унификация : $\{(ГГ, г (стереть)), ([Г], [стереть])\}$;
- (23') Л,
резолуция : (17') и (22'),
унификация : $\{(X, фр (г (стереть), гс (м (этот), n (последний), с (пароль))))\}$

Этот пример иллюстрирует построение синтаксического дерева фразы с помощью процедуры доказательства теоремы.

Резюмируем наш подход. Логическая формула C является следствием набора логических формул F_1, \dots, F_n тогда и только тогда, когда логическая формула $((F_1 \wedge \dots \wedge F_n) \supset C)$ общезначима. Здесь F_i — гипотезы, C — заключение. В нашем примере гипотезами являются продукции грамматики G , а заключением теоремы «данный список слов есть допустимая фраза данной грамматики» — логическая формализация тестируемого списка слов. Теорема формально

выражается следующей тавтологией:

$$\models (\wedge (\text{продукции из } G) \supset ((\text{список слов}) \supset \supset (\text{фраза из } G))).$$

Для получения алгоритма автоматического доказательства надо к логической резолюции добавить некую стратегию. Стратегия — это алгоритм, указывающий порядок просмотра дизъюнктов при применении метода резолюций. Такая стратегия существует, например, для языка Пролог. В § 5.1.8 мы продемонстрируем, как ОКГ может быть реализована в виде программы на Прологе. В § 5.1.10 будут рассмотрены понятия стратегии вообще и Пролога в частности.

5.1.8. ОК-грамматики в Прологе

Язык Пролог — плод трудов Колмероз [20], применившего хорновские дизъюнкты и резолюцию в информатике для создания нового языка программирования задач исследования естественного языка. Основой Пролога (одного из самых привлекательных языков ИИ) является логика первого порядка, ограниченная хорновскими дизъюнктами и снабженная резолюцией как единственным правилом вывода. Важнейшие аспекты Пролога и введение в программирование на нем приводятся в гл. 6. Здесь же мы лишь установим связь между ним, ОК-грамматиками и формализмом хорновских дизъюнктов. Программа на Прологе представляет собой набор правил ОК-грамматики. Следовательно, продукции

$$\text{фраза} \rightarrow \text{глагол}, \text{группа_сущ} (0) \quad (5.1)$$

$$\text{фраза} (\text{фр} (Г, Гс)) \rightarrow \text{глагол} (Г), \text{группа_сущ} (0, Гс) \quad (5.2)$$

можно толковать как инструкции, допускающие реализацию на Прологе. Примеры реализаций инструкций и программ на Прологе даются в гл. 6. Пока же достаточно знать, что «реализация» или, иначе, «выполнение», подразумевает управление правилами переписывания, т. е. резолюцией, применяемой к продукциям (которые интерпретируются как хорновские дизъюнкты).

Внутренний механизм Пролога переводит продукции (5.1) и (5.2) в следующие инструкции:

фраза (X, Y) :— *глагол* (X, X_1), *группа_сущ* ($0, X_1, Y$)

(5.3)

фраза (*фр* ($\Gamma, \Gamma c$), X, Y) :— *глагол* (Γ, X, X_1),

группа_сущ ($0, \Gamma c, X_1, Y$)

(5.4)

Логическая интерпретация хорновских дизъюнктов требует аргумента для учета следования (см. § 5.1.4). Замена символа переписывания \rightarrow символом прологовской импликации «:—» требует двух аргументов-списков. Пусть (для определенности инструкций). (5.3) и (5.4) проверяют, что некий список слов есть фраза языка. Тогда аргумент X предиката *глагол* (X, X_1) является этим списком слов, а X_1 — тем же списком без заголовка, если заголовок является глаголом. Следовательно, X_1 — соответствующая группе существительного часть списка, ибо [*фраза*] = [*глагол* | *группа_сущ*]. Наконец, аргумент Y соответствует списку слов без *глагола* и слов, представляющих *группу_сущ*. Таким образом, следование слов (или вообще связка «следует за» для продукций) представляется разностью «—» двух списков (см. § 6.3.6). В рассматриваемом сейчас примере

$X_1 = [X - [\text{глагол}]]$.

Каждый тип инструкций прологовских программ интерпретируется двумя различными способами: *декларативно* и *процедурно*. Если инструкция (дизъюнкт) имеет непустые заголовок и тело, то она произносится так:

- Декларативное произнесение: список $X - Y$ является фразой, если список $X - X_1$ является глаголом, а список $X_1 - Y$ — группой существительного.
- Процедурное произнесение: чтобы найти список $X - Y$, являющийся фразой, надо найти список $X - X_1$, являющийся глаголом, и список $X_1 - Y$, являющийся группой существительного.

Прочтения дизъюнкта $P: - Q, R, S$ таковы:

- Декларативное: утверждение P истинно, если утверждения Q , R и S истинны.

Процедурное: чтобы достичь цель P , надо достичь цели Q , R и S .

Процедурный аспект Пролога состоит в интерпретации каждого дизъюнкта как некой процедуры:

- заголовок дизъюнкта представляет имя и формальные аргументы процедуры,
- тело дизъюнкта может содержать обращение к другим процедурам.

Дизъюнкты с терминальными символами в телах записываются на Прологе следующим образом:

глагол \rightarrow [*стереть*]. (5.5)

глагол (g (*стереть*)) \rightarrow [*стереть*]. (5.6)

Внутренний механизм Пролога преобразует эти инструкции в:

глагол ([*стереть* | S], S). (5.7)

глагол (g (*стереть*), [*стереть* | S], S). (5.8)

Обозначение [*стереть* | S] представляет список с заголовком из терминального символа *стереть* и телом¹⁾ из списка S . Дизъюнкт с пустым телом (например, (5.7)) интерпретируется как факт. Выражения (5.7) и (5.8) можно использовать вместо (5.5) и (5.6) и интерпретировать следующим образом:

- Декларативно: для всех S истинно, что заголовок списка [*стереть* | S] является глаголом.
- Процедурно: цель состоит в нахождении глагола, с помощью которого строился бы список с заголовком *стереть*.

Фактами являются дизъюнкты с телами I , например:

глагол ([*стереть* | S], S):— I .

Если заголовок дизъюнкта пустой, то дизъюнкт содержит лишь члены с отрицаниями. Тогда он тол-

¹⁾ В § 5.1.4. заголовок и тело списка были определены соответственно как *голова* и *хвост*. — Прим. перев.

куется как «вопрос» (или «цель»). Заголовок у вопроса — ложный. Например:

Л:— фраза ($\text{фр}(\Gamma\Gamma, \Gammaс), X, Y$).

Используя прологовский символ «?—», перепишем этот вопрос в следующем виде:

?— фраза ($\text{фр}(\Gamma\Gamma, \Gammaс), X, Y$).

Он интерпретируется двумя способами:

- Декларативно: Существует ли список $X—Y$, представляющий фразу с синтаксическим деревом $\text{фр}(\Gamma\Gamma, \Gammaс)$?
- Процедурно: Построить список $X—Y$, представляющий фразу с синтаксическим деревом $\text{фр}(\Gamma\Gamma, \Gammaс)$.

Итак, программа на Прологе состоит из дизъюнктов, являющихся фактами или правилами. Обращение к программе осуществляется посредством постановки вопроса, т. е. путем присоединения к ней «цели» (дизъюнкта с пустым заголовком).

5.1.9. Пример

Описываемая ниже программа на Прологе может распознавать допустимые фразы некой грамматики в множестве последовательностей ее слов и строить синтаксические деревья для исследуемых фраз.

- (1) фраза ($\text{фр}(\Gamma\Gamma, \Gammaс)$) \rightarrow глагол ($\Gamma\Gamma$),
группа_сущ ($0, \Gammaс$)
- (2) группа_сущ ($Y_1, \text{гс}(\text{Мс}, \text{Пл}, \text{СС})$) \rightarrow местоимение ($K, \text{Мс}$),
прилагательное ($K, \text{Пл}$),
существительное ($K, Z_1, \text{СС}$), $\{Y_1 < Z_1\}$.
- (3) группа_сущ ($Y_2, \text{гс}(\text{Мс}, \text{Пл}, \text{СС}, \text{Предл}, \Gammaс)$) \rightarrow местоимение ($K, \text{Мс}$),
прилагательное ($K, \text{Пл}$),
существительное ($K, Z_2, \text{СС}$), $\{Y_2 < Z_2\}$,
предлог (Предл),
группа_сущ ($Z_2, \Gammaс$).

- (4) глагол (*г* (стереть)) \rightarrow [стереть].
 (5) глагол (*г* (печатать)) \rightarrow [печатать].
 (6) местоимение (*мужс*, *м* (этот)) \rightarrow [этот].
 (7) местоимение (*жен*, *м* (эта)) \rightarrow [эта].
 (8) прилагательное (*мужс*, *п* (второй)) \rightarrow [второй].
 (9) прилагательное (*жен*, *п* (вторая)) \rightarrow [вторая].
 (10) прилагательное (*мужс*, *п* (последний)) \rightarrow [последний].
 (11) прилагательное (*жен*, *п* (последняя)) \rightarrow [последняя].
 (12) существительное (*мужс*, 1, *с* (символ)) \rightarrow [символ].
 (13) существительное (*мужс*, 2, *с* (пароль)) \rightarrow [пароль].
 (14) существительное (*жен*, 3, *с* (строка)) \rightarrow [строка].
 (15) существительное (*жен*, 4, *с* (страница)) \rightarrow [страница].
 (16) предлог (*пред* (*в*)) \rightarrow [*в*].

Внутренний механизм Пролога преобразует эти инструкции в следующие:

- (1') фраза (*фр* (*ГГ*, *Гс*), *X*, *Y*)
 :- глагол(*ГГ*, *X*, *Z*), группа_сущ(0, *Гс*, *Z*, *Y*).
 (2') группа_сущ (*Y*₁, *гс* (*Мс*, *Пл*, *СС*), *Z*, *Y*)
 :- местоимение(*K*, *Мс*, *Z*, *W*), прилагательное
 (*K*, *Пл*, *W*, *R*), существительное
 (*K*, *Z*₁, *СС*, *R*, *Y*), *Y*₁ < *Z*₁.
 (3') группа_сущ (*Y*₂, *гс* (*Мс*, *Пл*, *СС*, *Предл*, *Гс*), *Z*, *Y*)
 :- местоимение (*K*, *Мс*, *Z*, *W*),
 прилагательное (*K*, *Пл*, *W*, *R*),
 существительное (*K*, *Z*₂, *СС*, *R*, *T*), *Y*₂ < *Z*₂,
 предлог (*Предл*, *T*, *U*), группа_сущ (*Z*₂, *Гс*,
U, *Y*).
 (4') глагол (*г* (стереть), [стереть | *S*], *S*).
 (5') глагол (*г* (печатать), [печатать | *S*], *S*).
 (6') местоимение (*мужс*, *м* (этот), [этот | *S*], *S*).
 (7') местоимение (*жен*, *м* (эта), [эта | *S*], *S*).

- (8') *прилагательное* (*мужс, n* (*второй*),
[*второй* | *S*], *S*).
- (9') *прилагательное* (*жен, n* (*вторая*),
[*вторая* | *S*], *S*).
- (10') *прилагательное* (*мужс, n* (*последний*),
[*последний* | *S*], *S*).
- (11') *прилагательное* (*жен, n* (*последняя*),
[*последняя* | *S*], *S*).
- (12') *существительное* (*мужс, 1, с* (*символ*),
[*символ* | *S*], *S*).
- (13') *существительное* (*мужс, 2, с* (*пароль*),
[*пароль* | *S*], *S*).
- (14') *существительное* (*жен, 3, с* (*строка*),
[*строка* | *S*], *S*).
- (15') *существительное* (*жен, 4, с* (*страница*),
[*страница* | *S*], *S*).
- (16') *предлог* (*пред* (*в*), [*в* | *S*], *S*).

Рассмотрим пример. Для синтаксического анализа фразы «стереть этот последний пароль в этой второй строке» ставим этой программе, состоящей из 16 инструкций, следующий вопрос:

- (17') ? — фраза (Φ , [*стереть, этот, последний, пароль, в, этой, второй, строке*], []).

Смысл этого вопроса таков: «найти фразу с синтаксическим деревом Φ , исходным списком слов (до анализа) [*стереть, этот последний, пароль, в, этой, второй, строке*] и пустым заключительным (после анализа) списком []». Инструкция (17') задает обращение к программе. Выход программы — результат операций, последняя из которых дает нижеследующую конкретизацию переменной Φ :

$\Phi = \text{фр}(\text{г}(\text{стереть}), \text{гс}(\text{м}(\text{этот}), \text{n}(\text{последний}), \text{с}(\text{пароль}), \text{пред}(\text{в}), \text{гс}(\text{м}(\text{этой}), \text{n}(\text{второй}), \text{с}(\text{строке}))))).$

5.1.10. Графическое представление и стратегии

Программа на Прологе представляет собой множество инструкций, которые можно проинтерпретировать как правила некоторой ОК-грамматики. Пролог

использует резолюцию как элементарный механизм выполнения. Стратегия задает выбор последовательности выполнения инструкций.

Определим графический формализм И/ИЛИ-деревьев для представления грамматик и, следовательно, программ на Прологе, а также стратегий их выполнения. В И/ИЛИ-дереве узлы помечены именами предикатов, входящих в правила грамматики.

Каждое правило изображается в виде *двухуровневого дерева*. Корнем дерева является узел, помеченный заголовком продукции. Листья (узлы второго уровня) помечены предикатами из тела правила. Для удобства порядок расположения листьев согласован с порядком предикатов в теле правила. Если корень помечен символом P , а листья — символами X_1, X_2, \dots, X_n , то $P \rightarrow X_1, X_2, \dots, X_n$ есть продукция (правило) грамматики. Листья называются *И-узлами* относительно корня, ибо продукция требует выполнения *всех* предикатов, содержащихся в его теле (в логической терминологии — «резолютивности»).

На рис. 5.4 изображены двухуровневые деревья продукций (правил грамматики) из § 5.1.2. Цифры в круглых скобках указывают номера продукций. За корнями деревьев, соответствующих продукциям (1) — (3), следуют множества И-узлов, отмеченные дугами окружностей, пересекающих отрезки прямых, представляющих дуги графа. Множество двухуровневых деревьев определяет грамматику. Прежде чем определять понятие И/ИЛИ-дерева и описывать стратегии, уточним терминологию. *Пакет* представляет собой множество продукций, имеющих одно и то же имя предиката в заголовке. ИЛИ-узлы пакета соответствуют предикатам его заголовка. Например, на рис. 5.4 таковы два узла *группа_сущ* у пакета *группа_сущ*.

Чтобы построить И/ИЛИ-дерево грамматики (программы) из двухуровневых деревьев продукций (инструкций), надо соединить между собой листья и корни всех двухуровневых деревьев, имеющие одинаковые метки.

Перейдем к описанию стратегии выполнения (резолютивности) программ. Для выполнения инструкции

нужно выполнить все предикаты его тела (т. е. *все* И-узлы, следующие за корнем двухуровневого дерева этой инструкции). Каждый из них выполняется по одной из инструкций пакета с меткой данного предиката (т. е. выполняется *один* из ИЛИ-узлов пакета). Выполнение программы осуществляется путем простой итерации выполнения инструкций: сначала выполняется инструкция, соответствующая корню И/ИЛИ-дерева, затем — инструкции из И- и ИЛИ-узлов (как указывалось выше) и, наконец, инструкции, соответствующие листьям.

Стратегия выполнения должна осуществлять выбор порядка исполнения инструкций.

- Выполняются все И-узлы, являющиеся листьями одного двухуровневого дерева; на рис. 5.4 они соединены дугой окружности. Но надо выбрать порядок их выполнения. Стратегия Пролога выбирает узлы в текстуальном ¹⁾ порядке, начиная с самого левого узла и проходя до самого правого узла (стратегия *слева направо*).
- Пакетом продуктов выполняется только один ИЛИ-узел. Две простейшие стратегии — *вширь* и *вглубь*. Стратегия *вширь* пытается выполнить все ИЛИ-узлы разом. Это соответствует выдвижению разом всех гипотез в доказательстве. В конце выполнения (приход в некоторый лист И/ИЛИ-дерева) будут оставлены только гипотезы, приведшие к резолютивности. Стратегия *вглубь* отрабатывает одну гипотезу, спускаясь как можно ниже по дереву; затем переходит к рассмотрению другой гипотезы. Успехом в процессе выполнения называется такая ситуация, когда гипотеза довела до листа. Спуск по дереву осуществляют до получения резолютивности для всех И-узлов. *Неудача* — это такая ситуация, когда гипотеза не доводит до листа. Новую гипотезу проверяют, начиная спускаться по дереву из любого, еще не выполненного ИЛИ-узла. Если использование всех ИЛИ-уз-

¹⁾ По тексту прологовской программы. — Прим. перев.

лов успеха не дает, то это означает неудачу программы. *Возврат*¹⁾ (по-английски *backtracking*) — это подъем по дереву после успеха или неудачи. Пролог использует стратегию *вглубь с возвратом*.

Стратегию Пролога *слева направо и вглубь с возвратом при неудаче* легко понять, рассматривая ее как обход И/ИЛИ-дерева. На рис. 5.4 стрелками указан обход дерева, соответствующий распознаванию фразы «печатать этот последний пароль». Вопрос «? — фраза (Ф, [печатать, этот, последний, пароль], [])» инициализирует программу. Он унифицируется с инструкцией «фраза → глагол, группа_сущ», определяющей И/ИЛИ-дерево программы. Это *дерево резолюции* для данного вопроса. На выходе из программы — унификации, осуществленные при обходе. Приведшие к неудаче унификации отсутствуют (от начала возврата до прихода в новый ИЛИ-узел).

5.2. Иерархия Хомского

5.2.1. Введение

В § 5.1.2 было введено понятие КС-грамматики. Поместим такие грамматики в иерархию Хомского и опишем соответствующие им автоматы. Подробности можно найти в [49]. В конце главы будет установлена эквивалентность ОК-грамматик (см. § 5.1.5) самым общим грамматикам Хомского.

Граматики часто рассматривают как формальные *порождающие* системы. «Правильно построенные предложения» (фразы языка) получаются («порождаются») из исходного символа применением продукций (правил построения). Об этом говорилось в § 5.1.3 (для КСГ).

Автомат — это формальная *воспринимающая* система (акцептор). Она начинает с некой фразы. Правила автомата позволяют проверять, принадлежит или не принадлежит эта фраза данному языку. Автомат

¹⁾ Иные названия: процесс возврата, обратный шаг, бэктрекинг, поиск с возвратом, возврат при поиске и др. — Прим. перев.

эквивалентен данной грамматике, если он воспринимает весь порожденный ею язык и только этот язык.

Граматики Хомского (называемые также *грамматиками структуры фразы*) делятся на четыре типа с номерами от 0 до 3 в порядке убывания их общности. Наименее общими являются *регулярные грамматики*. Эквивалентные им акцепторы — *конечные автоматы*. На втором уровне общности находятся КС-грамматики и *стековые автоматы*. Этажом выше — грамматики типа 1 (*чувствительные к контексту*¹⁾) и *линейно ограниченные автоматы*. На вершине иерархии Хомского расположились грамматики типа 0 и *машины Тьюринга*.

Как и в частном случае КС-грамматик (см. § 5.1.3), грамматики Хомского — это четверки (V, T, P, S) . Продукции (элементы из P) имеют вид $\varphi \rightarrow \psi$, где $\varphi \in (V \cup T)^+$ и $\psi \in (V \cup T)^*$. Тип грамматики определяется ограничениями, наложенными на φ и ψ . Например, для КС-грамматик φ состоит из единственного символа-переменной: $\varphi \in V$. Порожденный грамматикой Хомского язык L представляет собой множество последовательностей слов, которые можно получить, исходя из исходного символа S . Формально:

$$L = \{w \in T^* \mid S \overset{\cdot}{\Rightarrow} w\}$$

где отношение $\overset{\cdot}{\Rightarrow}$ такое же, как определенное в § 5.1.3.

5.2.2. Регулярные грамматики

Грамматика Хомского называется *регулярной* (или типа 3), если ее продукции имеют вид $A \rightarrow a$ или $A \rightarrow bB$, где A и B — переменные, a и b — терминальные символы.

Из определения следует, что речь идет о регулярной *справа* грамматике. Существуют эквивалентные²⁾ понятия грамматик: регулярных слева, линейных сле-

¹⁾ Другие названия: *грамматики непосредственных составляющих (НС-грамматики)*, или *контекстно-зависимые грамматики*. — *Прим. перев.*

²⁾ Напомним, что формальные системы эквивалентны, если они определяют одно и то же множество фраз.

ва или линейных справа. *Линейная справа* грамматика имеет продукции видов $A \rightarrow x$ или $A \rightarrow yB$, где A и B — переменные, x и y — непустые последовательности слов ($x, y \in T^+$).

Язык называется *регулярным*, если он порождается регулярной грамматикой. Чтобы указать место нахождения этих языков в иерархии, отметим два следующих факта:

- любой конечный язык регулярен,
- существуют нерегулярные КС-языки.

Последний факт говорит о существенном различии между типами 2 и 3.

5.2.3. Конечные автоматы

Как и грамматики, автоматы определяются конечными алфавитами (словарями, глоссариями) и правилами переписывания. Конечный автомат — это пятерка $(Q, \Sigma, \delta, q_0, F)$, где

Q — конечное множество состояний,

Σ — конечное множество, называемое *входным словарем*,

δ — отношение¹⁾ между $Q \times \Sigma$ и Q , элементы которого называются правилами (переписывания) и обозначаются $q_i a_k \rightarrow q_j$, где q_i и q_j — состояния, a_k — входной символ,

q_0 — элемент из Q , называемый начальным состоянием,

F — подмножество из Q , элементы которого называются заключительными состояниями.

Конечный автомат можно рассматривать как машину, которая в каждый момент времени находится в некотором состоянии $q \in Q$ и читает поэлементно последовательность w символов из Σ , записанную на конечной слева ленте. При этом либо читающая головка машины движется в одном направлении (слева направо), либо лента перемещается (справа налево)

¹⁾ В литературе широко распространено название *функция переходов*. — *Прим. перев.*

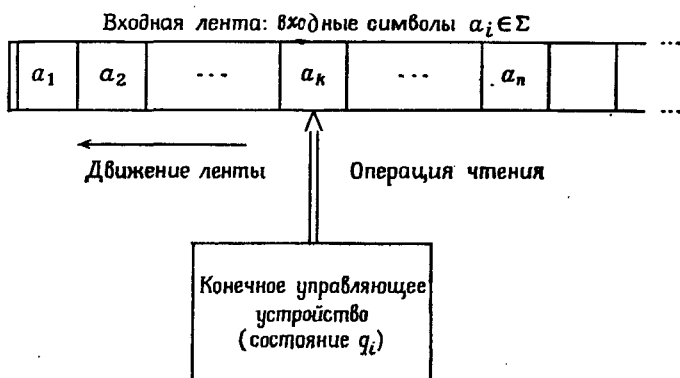


Рис. 5.5. Конечный автомат.

(см. рис. 5.5). Если автомат в состоянии q_i читает символ a_k и правило $q_i a_k \rightarrow q_j$ принадлежит δ , то автомат воспринимает этот символ и переходит в состояние q_j для обработки следующего символа.

Конфигурация конечного автомата — это элемент множества $Q \times \Sigma^*$, т. е. последовательность вида $q\omega$, где q — состояние из Q и ω — элемент из Σ^* . Отношения \Rightarrow и $\stackrel{*}{\Rightarrow}$ между конфигурациями определяются с использованием отношения δ , так, как это делалось для грамматик (см. § 5.1.3):

- $q_i a_k \omega \Rightarrow q_j \omega$, если $q_i a_k \rightarrow q_j$ — правило из δ (оно применяется к $q_i a_k \omega$ для получения $q_j \omega$),
- $C_1 \stackrel{*}{\Rightarrow} C_n$, если существует последовательность C_1, C_2, \dots, C_n , где $C_i \Rightarrow C_{i+1}$ для $i = 1, 2, \dots, n-1$.

Следовательно, отношение $\stackrel{*}{\Rightarrow}$ есть рефлексивное и транзитивное замыкание отношения \Rightarrow . Пусть ε — пустой элемент из Σ^* . Воспринимаемый конечным автоматом $(Q, \Sigma, \delta, q_0, F)$ язык L определяется как множество

$$L = \{\omega \in \Sigma^* \mid q_0 \omega \stackrel{*}{\Rightarrow} q_f \varepsilon, \text{ где } q_f \in F\}.$$

Значит, фраза ω воспринимается автоматом, если, начиная функционирование из начального состояния,

этот автомат читает все слова фразы и в некоторый момент времени приходит в заключительное состояние.

В частности, если все «заголовки» $q_i a_k$ правил из δ различны, то отношение δ является частичной функцией (вообще говоря, определенной на $Q \times \Sigma$ и принимающей значения из Q). Ее можно продолжить до функции, определенной на множестве конфигураций $Q \times \Sigma^*$. При этом получится частичная функция со значениями из множества Q (множества состояний). Соответствующий автомат является *детерминированным*: к любой конфигурации $q_i w$ применимо не более чем одно правило. Оказывается, что недетерминированные автоматы не более общи, чем детерминированные, ибо справедливо следующее утверждение:

- Для любого конечного недетерминированного автомата можно построить эквивалентный ему конечный детерминированный автомат.

Мы определили регулярные языки, порожденные регулярными грамматиками, и, кроме того, языки, воспринимаемые конечными автоматами. Следующее утверждение говорит о том, что эти два определения эквивалентны:

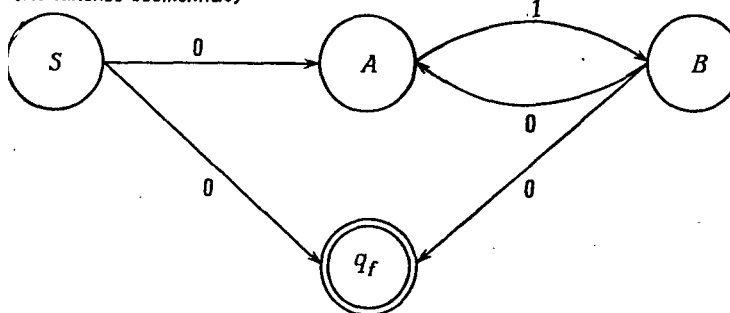
- Язык воспринимается конечным автоматом тогда и только тогда, когда он порождается регулярной грамматикой.

Это свойство допускает также «конструктивную» формулировку:

- По любой регулярной грамматике строится эквивалентный ей конечный автомат, и наоборот: по всякому конечному автомату можно построить эквивалентную ему регулярную грамматику.

Бывает удобно представлять конечный автомат ориентированным графом (*диаграммой состояний* или, иначе, *диаграммой переходов*). Узлы орграфа соответствуют состояниям автомата. Если входной символ a_k вызывает переход из q_i в q_j (т. е. δ содержит

(Начальное состояние)



(Заключительное состояние)

Рис. 5.6. Диаграмма состояний конечного автомата.

правило $q_i a_k \rightarrow q_j$), то в диаграмме дуга с пометкой a_k направлена из q_i в q_j . Пример будет рассмотрен в § 5.2.4 (см. рис. 5.6).

5.2.4. Пример

Пример, который будет сейчас рассмотрен, подсказывает идею доказательства эквивалентности грамматики и соответствующего ей автомата. Язык $L = \{0(10)^n \mid n \geq 0\}$ состоит из фраз 0, 010, 01010 и т. д. Он порожден регулярной грамматикой (V, T, P, S) , где:

$$V = \{S, A, B\},$$

$$T = \{0, 1\},$$

$$P = \{S \rightarrow 0, \\ S \rightarrow 0A, \\ A \rightarrow 1B, \\ B \rightarrow 0, \\ B \rightarrow 0A\}.$$

Он же воспринимается следующим (недетерминированным) конечным автоматом:

$$\Sigma = \{0, 1\},$$

$$Q = \{S, A, B, q_f\},$$

$$q_0 = S,$$

$$F = \{q_f\},$$

$$\delta = \{S0 \rightarrow q_f, \\ S0 \rightarrow A, \\ A1 \rightarrow B, \\ B0 \rightarrow q_f, \\ B0 \rightarrow A\}.$$

Проследим, каким образом строится этот автомат из приведенной выше грамматики. Рассматриваем порождение некоторой фразы:

$$S \Rightarrow 0A \Rightarrow 01B \Rightarrow 010A \Rightarrow 0101B \Rightarrow 01010.$$

Затем выясняем, как она воспринимается автоматом:

$$S01010 \Rightarrow A1010 \Rightarrow B010 \Rightarrow A10 \Rightarrow B0 \Rightarrow q_f.$$

После этого можно построить фрагмент диаграммы соответствующего конечного автомата (от S к A , затем к B , потом к A , снова к B и после этого к q_f). Полная диаграмма состояний изображена на рис. 5.6.

5.2.5. КС-грамматики и стековые автоматы

Существуют КС-языки, не порождаемые никакой регулярной грамматикой. Классический пример — язык, порожденный следующими КС-продукциями:

$$S \rightarrow 0S1$$

$$S \rightarrow 01$$

Очевидно, что $L = \{0^n 1^n \mid n \geq 1\}$. Покажем, что L не порождается никакой регулярной грамматикой.

Интуитивно ясно, что никакой конечный автомат не воспринимает L . Для восприятия фразы из L надо проверять равенство длин последовательностей нулей и единиц, что требует неограниченной памяти. Конечный автомат, память которого ограничена числом состояний в Q , не может быть акцептором языка L .

Как мы установим, память автоматов, эквивалентных КСГ, не ограничена. *Стековые автоматы* соотносятся с КС-грамматиками (т. е. грамматиками типа 2) так же, как *конечные автоматы* с *регулярными*

грамматиками. Стековый автомат имеет неограниченную память, называемую *стеком*¹⁾. В стеке можно записать произвольное число символов. Функционирование стека мы сейчас опишем. Формально стековый автомат представляет собой семерку $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, где

Q — конечное множество состояний (управляющее устройство),

Σ — конечный входной словарь,

Γ — конечный стековый словарь,

q_0 — элемент из Q (начальное состояние),

Z_0 — элемент из Γ (начальный символ стека),

F — подмножество из Q (множество заключительных состояний),

δ — множество правил переписывания, имеющих следующий вид:

$$q_i a_j Z_k \rightarrow q_l \gamma_m,$$

- где q_i и q_l — состояния, a_j — входной символ или пустая входная последовательность ϵ , Z_k — символ из стека и γ_m — последовательность символов из стека, возможно, пустая (в этом случае она обозначается через ϵ').

Интерпретация применения правила (см. рис. 5.7): автомат в состоянии q_i , читая входной символ a_j , обозревая в верхней ячейке стека символ Z_k , воспринимает a_j и заменяет Z_k на последовательность γ_m в верхней ячейке стека. Если a_j — пустая последовательность ϵ , то чтение входного символа не осуществляется.

Конфигурация для стекового автомата имеет вид $q\omega\gamma$, где q — состояние, ω — выражение из Σ^* (последовательность входных слов) и γ — выражение из Γ^* (последовательность символов из стека). Для описания процесса переписывания определим отношения

\Rightarrow и $\overset{*}{\Rightarrow}$ между конфигурациями:

$q_i a_j \omega Z_k \gamma \Rightarrow q_l \omega \gamma_m \gamma$, если правило $q_i a_j Z_k \rightarrow q_l \gamma_m$ принадлежит δ .

¹⁾ Иное название магазин. — *Прим. перев.*

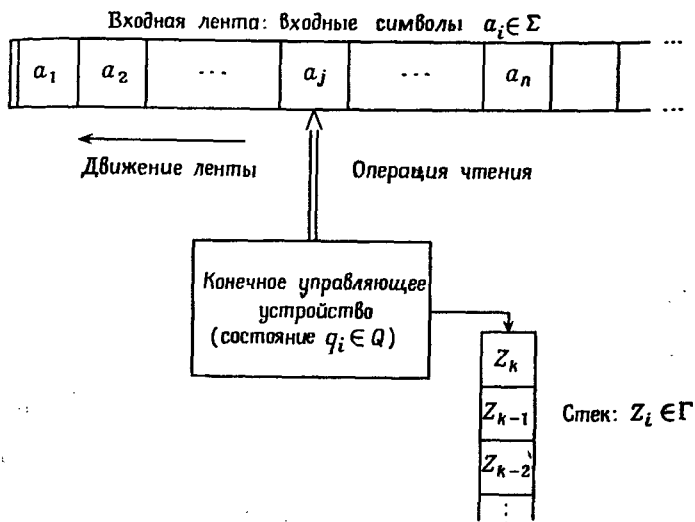


Рис. 5.7. Стекковый автомат.

(Заметим, что в случае $\gamma_m = \epsilon'$ содержимое стека уменьшается.)

- $C_1 \xrightarrow{*} C_n$, если существует последовательность конфигураций C_1, C_2, \dots, C_n с $C_i \Rightarrow C_{i+1}$ для $i = 1, 2, \dots, n-1$.

Есть два способа определить язык, воспринимаемый стековым автоматом. Во-первых, множество входных цепочек, для которых существует опустошающая стек последовательность операций (такой язык называют языком, *воспринимаемым пустым стеком*). Формально:

$$L = \{w \in \Sigma^* \mid q_0 w Z_0 \xrightarrow{*} q \epsilon \epsilon', \text{ где } q \in Q\}.$$

Во-вторых, множество входных цепочек, для которых существует последовательность операций, приводящая автомат в заключительное состояние (это язык, *воспринимаемый заключительным состоянием*):

$$L = \{w \in \Sigma^* \mid q_0 w Z_0 \xrightarrow{*} q_f \epsilon \gamma, \text{ где } q_f \in F \text{ и } \gamma \in \Gamma^*\}.$$

Определения эквивалентны: из одного автомата строится другой, воспринимающий тот же язык.

Сформулируем основную теорему данного параграфа:

- Язык воспринимается стековым автоматом тогда и только тогда, когда он порождается КС-грамматикой.

Эта теорема допускает также «конструктивную» формулировку.

Стековый автомат называется детерминированным, если все заголовки правил (т. е. $q_i a_j Z_k$) различны. Тогда к любой конфигурации применимо не более чем одно правило. Упомянутая выше эквивалентность детерминированных и недетерминированных конечных автоматов не обобщается на случай стековых автоматов: существуют порожденные КС-грамматиками языки, не воспринимаемые никакими детерминированными стековыми автоматами.

5.2.6. Пример

Рассмотрим язык $L = \{xax^R \mid x \in \{0, 1\}^*\}$, где x^R получена из последовательности x в результате применения операции инвертирования. Очевидно, что L порожден КС-грамматикой с продукциями

$$S \rightarrow 0S0$$

$$S \rightarrow 1S1$$

$$S \rightarrow a$$

Построим стековый автомат, воспринимающий (пустым стеком) язык L . Управляющим устройством (УУ) автомата будет множество состояний $\{q_0, q_1\}$, а стек может получать символы A , B и C . Функционирование автомата:

1. Автомат начинает работу в состоянии q_0 с символа A в верхней ячейке стека.
2. Если в состоянии q_0 на вход поступает 0, то в стек помещается B . Если же поступит 1, то в стек поместим C . В обоих случаях УУ останется в состоянии q_0 .

3. Если в состоянии q_0 на вход поступит символ a , то УУ переходит в состояние q_1 , а содержимое стека не меняется.
4. Если в состоянии q_1 с B или C в верхней ячейке стека на вход поступит 0 или 1 (соответственно), то символ из верхней ячейки убирается. В обоих случаях УУ остается в состоянии q_1 .
5. Если в состоянии q_1 в верхней ячейке стека находится символ A , то он из этой ячейки убирается без ожидания поступления входного слова.
6. В остальных случаях состояние и содержимое стека не меняются.

Формальное описание автомата $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ таково:

$$\begin{aligned}
 Q &= \{q_0, q_1\}, \\
 \Sigma &= \{0, 1, a\}, \\
 \Gamma &= \{A, B, C\}, \\
 Z_0 &= A, \\
 \delta &= \{q_0 0 A \rightarrow q_0 B A, \\
 &\quad q_0 1 A \rightarrow q_0 C A, \\
 &\quad q_0 0 B \rightarrow q_0 B B, \\
 &\quad q_0 1 B \rightarrow q_0 C B, \\
 &\quad q_0 0 C \rightarrow q_0 B C, \\
 &\quad q_0 1 C \rightarrow q_0 C C, \\
 &\quad q_0 a A \rightarrow q_1 A, \\
 &\quad q_0 a B \rightarrow q_1 B, \\
 &\quad q_0 a C \rightarrow q_1 C, \\
 &\quad q_1 0 B \rightarrow q_1 \epsilon', \\
 &\quad q_1 1 C \rightarrow q_1 \epsilon', \\
 &\quad q_1 \epsilon A \rightarrow q_1 \epsilon'\}.
 \end{aligned}$$

(Символ ϵ' , употребляемый в продукциях справа как стековое выражение, означает устранение символа из верхней ячейки стека, а символ ϵ , записываемый в продукциях слева, как входной символ, указывает на то, что на входе ничего не читается.) Заметим, что речь здесь идет о детерминированном стековом автомате.

В качестве примера работы этого автомата укажем последовательность $C_1 \Rightarrow C_2 \dots \Rightarrow C_n$,

описывающую восприятие фразы $w = 011a110$:

$$\begin{aligned} C_1 &= q_0 011a110A, \\ C_2 &= q_0 11a110BA, \\ C_3 &= q_0 1a110CBA, \\ C_4 &= q_0 a110CCBA, \\ C_5 &= q_1 110CCBA, \\ C_6 &= q_1 10CBA, \\ C_7 &= q_1 0BA, \\ C_8 &= q_1 \varepsilon A, \\ C_9 &= q_1 \varepsilon \varepsilon'. \end{aligned}$$

5.2.7. Грамматики и языки Хомского типа 1

Существуют языки, не порождаемые никакой КС-грамматикой. Простой пример — язык $L = \{0^n 1^n 0^n \mid n \geq 1\}$, который является языком структуры фразы типа 1 (в строгом смысле). Продукции $\varphi \rightarrow \psi$ грамматики Хомского типа 1 таковы, что в ψ не меньше символов, чем в φ . Значит, последовательные конфигурации φ_1, φ_2 и т. д., полученные в цепочке $S \Rightarrow \Rightarrow \varphi_1 \Rightarrow \dots \Rightarrow \varphi_n$, не уменьшаются по длине. Поэтому языки типа 1 (а значит, и типов 2 и 3) являются рекурсивными множествами (см. § 2.2.10). Можно показать, что язык $L = \{0^n 1^n 0^n \mid n \geq 1\}$ порожден грамматикой типа 1 с тремя переменными и нижеследующими продукциями:

$$\begin{aligned} S &\rightarrow 0B0 \\ B &\rightarrow 1 \\ B &\rightarrow 0BC \\ C0 &\rightarrow 100 \\ C1 &\rightarrow 1C \end{aligned}$$

Заметим, что, в силу данного выше определения¹⁾, язык типа 1 не содержит пустой фразы ε . Следовательно, не все КС-языки являются языками типа 1. Однако справедливо следующее свойство:

● Если L — КС-язык, то $L \setminus \{\varepsilon\}$ — язык типа 1.

Грамматикам типа 1 эквивалентны *линейно ограниченные автоматы* (машины Тьюринга с входной лентой ограниченной длины).

¹⁾ Существуют другие определения, эквивалентные приведенному.

5.2.8. Машины Тьюринга и грамматики типа 0

Граматики типа 0 (или неограниченные грамматики) — самые общие грамматики структуры фразы. В таких грамматиках на продукции накладывается только одно ограничение: $\varphi \neq \varepsilon$ в $\varphi \rightarrow \psi$. Языки типа 0, вообще говоря, не являются рекурсивными множествами. Другими словами, проблема выявления принадлежности выражения w языку, порожденному грамматикой типа 0, где $w \in T^*$, в общем случае неразрешима (см. § 2.2.6). Автоматы, воспринимающие языки типа 0, помещаются на самом верхнем уровне иерархии Хомского. Они и являются машинами Тьюринга. Эти «машины» были первой формальной моделью для понятия вычислимости (§ 2.2.7). Подробности см. в [49].

Машина Тьюринга — это семерка $(Q, \Sigma, D, G, B, \delta, q_0)$, где:

Q — конечное множество состояний,

Σ — входной словарь,

D и G — «правый» и «левый» символы (не входящие в $Q \cup \Sigma$),

B — «пробельный» символ из Σ .

δ — множество правил следующих видов:

$$q_i a_j \rightarrow q_k a_l,$$

$$q_i a_j \rightarrow q_k D,$$

$$q_i a_j \rightarrow q_k G,$$

где q_i и q_k — состояния, a_j и a_l — входные символы. Заголовки правил попарно различны. Значит, δ есть отображение некоторого подмножества из $Q \times \Sigma$ в $Q \times (\Sigma \cup \{D, G\})$. Следовательно, машины Тьюринга являются детерминированными преобразователями.

q_0 — начальное состояние, элемент из Q .

Конфигурации машины Тьюринга представляют собой элементы множества $\Sigma^* \times Q \times \Sigma^+$, т. е. последовательности вида $w_1 q_i a_j w_2$, где w_1 и $w_2 \in \Sigma^*$, $a_j \in \Sigma$ и $q_i \in Q$ (см. рис. 5.8). Правила из δ задают последовательные переходы от одной конфигурации

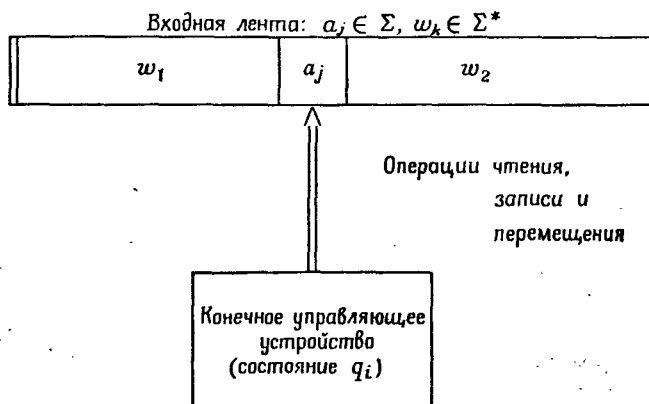


Рис. 5.8. Машина Тьюринга.

(до тех пор, пока не будет достигнута «заключительная» конфигурация). Конфигурация $w_1 q_i a_j w_2$ называется *заключительной*, если к ней не применимо ни одно из правил δ (т. е. правил с заголовком $q_i a_j$). Существуют и другие, эквивалентные данному, определения понятия машины Тьюринга.

На рис. 5.8 машина Тьюринга представлена в виде конечного автомата (управляющего устройства; коротче, УУ) и ленты, разбитой на ячейки и выступающей в качестве внешней памяти машины. УУ связано с лентой головкой, обзорающей в каждый дискретный момент времени одну ячейку. В процессе работы машины головка может:

- *читать* содержимое ячейки, обозреваемой головкой в текущий момент времени,
- *печатать* в обозреваемой ячейке соответствующий символ из входного словаря,
- *перемещаться* в соседнюю ячейку, находящуюся слева или справа от обозреваемой ячейки.

Итак, на каждом шаге работы машина действует следующим образом: находясь в состоянии q_i и прочитав символ a_j в обозреваемой ячейке, она переходит в состояние q_k и либо печатает в этой ячейке символ a_i , либо перемещает головку в соседнюю

ячейку (влево или вправо). Подробности см. в [73].

Отношение \Rightarrow на множестве конфигураций выражает простое переписывание и определяется следующим образом:

- для правила $q_i a_j \rightarrow q_k a_l$ имеем

$$w_1 q_i a_j w_2 \Rightarrow w_1 q_k a_l w_2$$

(смена ¹⁾ состояния и входного символа),

- для правила $q_i a_j \rightarrow q_k D$ возможны два случая:

$$w_1 q_i a_j a_l w_2 \Rightarrow w_1 a_l q_k a_l w_2$$

(смена ²⁾ состояния с перемещением вправо)
или

$$w_1 q_i a_j \Rightarrow w_1 a_j q_k B$$

(то же с удлинением цепочки символов присоединением B справа),

- для правила $q_i a_j \rightarrow q_k G$ имеем аналогичные возможности:

$$w_1 a_l q_i a_j w_2 \Rightarrow w_1 q_k a_l a_j w_2,$$

$$q_i a_j w_2 \Rightarrow q_k B a_j w_2.$$

Пусть $\overset{*}{\Rightarrow}$ — рефлексивное и транзитивное замыкание отношения \Rightarrow . Могут существовать конфигурации C , для которых никакая заключительная конфигурация C_f не удовлетворяет соотношению $C \overset{*}{\Rightarrow} C_f$. В таком случае говорят, что машина «не останавливается».

Воспринимаемый машиной Тьюринга язык L определяется следующим образом:

$$L = \{w \in \Sigma^* \mid q_0 w \overset{*}{\Rightarrow} C_f, \text{ где } C_f \text{ — заключительная конфигурация.}\}$$

Если w — фраза языка, то машина «находит» за конечное время (единственную) заключительную конфигурацию C_f . Если же w не принадлежит языку, то

¹⁾ Случай, когда состояние, или входной символ, или и то и другое не меняются, тоже может иметь место. — Прим. ред.

²⁾ Состояние может остаться прежним. — Прим. ред.

машина не останавливается. Это обстоятельство приводит к неразрешимости проблемы распознавания принадлежности данного выражения заданному языку. В действительности приведенные сейчас рассуждения служат неким обоснованием неразрешимости так называемой «проблемы остановки»: не всегда можно предусмотреть, остановится данная машина или нет.

В таблице на рис. 5.9 дано компактное описание грамматик Хомского и эквивалентных им автоматов.

	Грамматика	Автомат	Пример
Тип 0	Неограниченная	Машина Тьюринга	
Тип 1	Чувствительная к контексту $\varphi \rightarrow \psi$ $l(\varphi) \leq l(\psi)$	Машина Тьюринга с конечной лентой	$\{0^n 1^n 0^n \mid n \geq 1\}$
Тип 2	Контекстно-свободная $A \rightarrow \alpha$	Стековый	$\{0^n 1^n \mid n \geq 1\}$
Тип 3	Регулярная $A \rightarrow bB$ $A \rightarrow a$	Конечный	$\{0(10)^n \mid n \geq 0\}$

Рис. 5.9. Иерархия Хомского.

5.3. Формализм усиленных сетей переходов

5.3.1. Введение

Кратко изложим формализм *усиленных*¹⁾ *сетей переходов* или, короче, УП-сетей, УСП (по-английски Augmented Transition Network, ATN), предложенный

¹⁾ Или, иначе, *расширенных*. — Прим. перев.

в работе [114]. Будем строить его постепенно из формализма конечных автоматов (§ 5.2.3). Формализм УСП включает несколько классов *сетей* — соответственно типам грамматик иерархии Хомского (разд. 5.2). Элементарнейшие — *базовые сети переходов*, или, короче, БП-сети, БСП (английский термин — *Basic Transition Network*, BTN) соответствуют конечным автоматам. (Иногда «сеть» и «автомат» означают «класс сетей» и «класс автоматов».) С помощью БП-сетей регулярные языки определяются более сжато, чем с использованием конечных автоматов. Это достигается посредством интерпретаций синтаксических категорий грамматик с привлечением входных и терминальных символов сети. У конечного автомата входными символами являются только слова, построенные из символов входного словаря.

Второй тип сетей в иерархии образуют *рекурсивные сети переходов*, или, короче, РП-сети РСП (в англоязычной литературе — *Recursive Transition Network*, RTN). Такая сеть получается из БСП добавлением рекурсии. РП-сети соответствуют стековому автомату и воспринимают КС-языки (§ 5.2.5). На верхнем уровне иерархии находятся УСП. Такая сеть получается из РСП добавлением регистров. Любая УП-сеть эквивалентна подходящей машине Тьюринга, и наоборот — любая машина Тьюринга эквивалентна подходящей УСП (другими словами, множество языков, воспринимаемых УП-сетью, совпадает с множеством языков, воспринимаемых машинами Тьюринга). УП-сети соответствуют ОК-грамматикам (§ 5.1.5). Они являются основой языков программирования Лисп и Пролог соответственно. Ниже мы опишем некоторый метод перевода УП-сетей в ОК-грамматики.

5.3.2. Конечные автоматы и диаграммы переходов

Мы уже видели, как можно описать язык грамматикой. Другой подход состоит в перечислении всех фраз языка (если множество фраз конечное). Грамматику, представленную в §§ 5.1.2 и 5.1.4, можно «заменить» множеством фраз порождаемого языка. Приведем

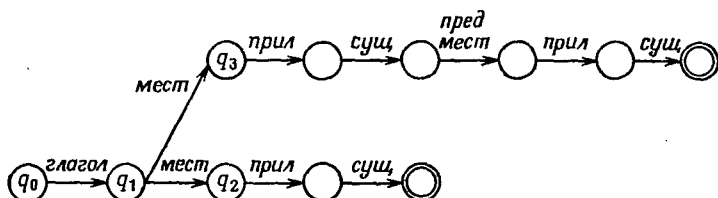


Рис. 5.11. Диаграмма переходов с проверками на принадлежность синтаксическим категориям.

категории, соответствующей текущему переходу. Мы получаем, таким образом, базовую сеть переходов (БСП).

Для определения формализма ОК-грамматик (см. § 5.1.5) использовалась запись, близкая Прологу. Сейчас мы поясним формализм БП-сетей, применяя близкие Лиспу обозначения. Команды БСП запишем следующим образом:

- WRD: сравнивать полученное слово со словом-меткой,
- CAT: проверить принадлежность полученного слова категории-метке,
- JUMP: перейти в следующий узел без учета полученного слова.

Команда CAT специфична для формализма БП-сетей. Команда WRD — проверка соответствия конечных автоматов. Лисп используется для выполнения команд из формализма БСП. Он содержит набор функций для обработки списков команд и создания новых процедур [113].

Регулярный язык $L = \{0(10)^n | n \geq 0\}$, воспринимаемый автоматом из § 5.2.4, можно описать с помощью БСП. Представим каждый узел сети списком, голова (первый элемент) в котором является меткой, а хвост состоит из команд, соответствующих переходам. Правила из множества δ , приведенного в § 5.2.4, перейдут в списки команд:

```
(M (S (WRD 0 TO A)
      (WRD 0 TO qf))
  (A (WRD 1 TO B)))
```

$(B \text{ (WRD } 0 \text{ TO } A)$
 $\text{ (WRD } 0 \text{ TO } q_i))$
 $(q_i \text{ ()))}$

Например, в узле S правила $S0 \rightarrow q_i$ и $S0 \rightarrow A$ заменяются на команды $(\text{WRD } 0 \text{ TO } q_i)$ и $(\text{WRD } 0 \text{ TO } A)$. Тем самым мы отразим тот факт, что автомат может перейти в узлы q_i или A при получении слова 0. Пустой список $(\)$ соответствует заключительным узлам (здесь — q_i). Результирующая БС-сеть кратко описывает язык в терминах грамматики, ибо диаграмма переходов исчерпывающе перечисляет фразы языка (§ 5.3.2). Например, сеть на рис. 5.11 представляет две продукции регулярной грамматики:

фраза \rightarrow *глагол, мест, прил, суц*

фраза \rightarrow *глагол, мест, прил, суц, предл, мест,*
прил, суц

Ниже приводится пример посложнее — чтобы лучше отразить связь формализма БСП с Лиспом.

5.3.4. Пример

Модифицируем грамматику, описанную в §§ 5.1.2 и 5.1.4, так, чтобы можно было осуществлять согласование слов по родам. Например, «мужской» и «женский» род синтаксической категории *группа_суц* представим выражениями через «гсм» и «гсж» соответственно. Обогащенная грамматика выглядит так:

- (1) *фраза* \rightarrow *глагол, группа_суц*
- (2) *группа_суц* \rightarrow "этот", *гсм*
- (3) *группа_суц* \rightarrow "эта", *гсж*
- (4) *гсм* \rightarrow ПМ, СМ,
- (5) *гсм* \rightarrow ПМ, СМ "в этом", *гсм*
- (6) *гсм* \rightarrow ПМ, СМ "в этой", *гсж*
- (7) *гсж* \rightarrow ПЖ, СЖ,
- (8) *гсж* \rightarrow ПЖ, СЖ, "в этом", *гсм*
- (9) *гсж* \rightarrow ПЖ, СЖ, "в этой", *гсж*
- (10) *глагол* \rightarrow "печатать" | "стереть"
- (11) ПМ \rightarrow "первый" | "второй" | "послед-

ний"

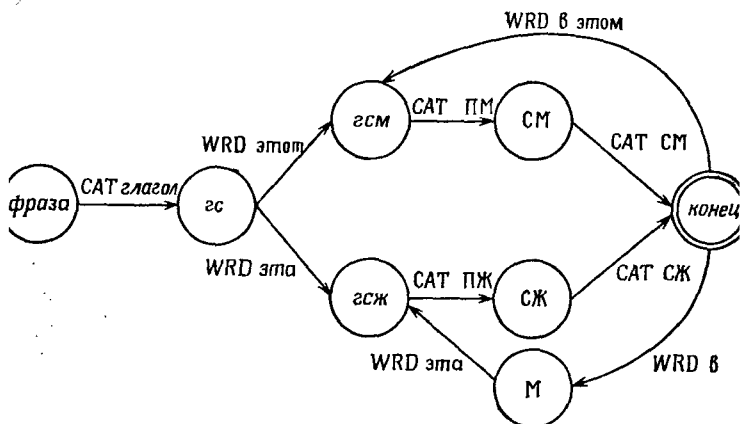


Рис. 5.12. Базовая сеть переходов.

(12) ПЖ → "первая" | "вторая" | "последняя"

(13) СМ → "символ" | "пароль"

(14) СЖ → "строка" | "страница"

Чтобы, например, достичь цель продукции (2), надо использовать одну из продукций пакета *гсм*: (4), (5) или (6). В формализме БСП продукции (1)–(9) выражены следующим списком:

```

((фраза      (CAT глагол ТО группа_сущ))
(группа_сущ (WRD этот ТО гсм)
              (WRD эта ТО гсж))
(гсм        (CAT ПМ ТО см))
(гсж        (CAT ПЖ ТО сж))
(см         (CAT см ТО конец))
(сж         (CAT сж ТО конец))
(конец      (WRD в этом ТО гсм)
              (WRD в ТО мест)
              ( ))
(мест       (WRD эта ТО гсж)))
  
```

Соответствующая БП-сеть изображена на рис. 5.12. Узлы сети соответствуют элементам основного списка, каждый из которых является списком с головой-

меткой узла и с хвостом из меток переходов, разрешенных данным узлом. Например, узел *группа_сущ* допускает два перехода: (WRD этот ТО *гсм*) и (WRD эта ТО *гсж*). Если извлечено местоимение мужского (соответственно женского) рода, то переход осуществляется в узел группы существительного мужского (соответственно женского) рода.

Переход в БСП описывается списком из четырех элементов:

- 1) команда WRD, CAT или JUMP,
- 2) параметр команды (для команд WRD или CAT) или пустая последовательность ϵ (для команды JUMP),
- 3) символ ТО,
- 4) узел назначения.

Имеется заключительный узел (в рассмотренном примере — *конец*) со специальным переходом «выход из сети», представленным пустым списком ().

5.3.5. Рекурсивные сети переходов

Будучи эквивалентной конечному автомату, БСП не воспринимает КС-язык $L = \{0^n 1^n | n \geq 1\}$, упоминавшийся в § 5.2.5. Напомним, что этот язык L порождается продуктами

$$S \rightarrow 01$$

$$S \rightarrow 0S1$$

и воспринимается стековым (но не конечным) автоматом.

Построим сети-акцепторы для КС-языков. Эти сети обобщают БП-сети, аналогично тому как стековые автоматы обобщают конечные. КС-язык лучше всего характеризует рекурсия, отраженная в правилах задающей его грамматики наличием одного и того же синтаксического символа в заголовке и теле некоторых продуктов (таков символ S в приведенном выше примере). Рекурсию обрабатывают РП-сети.

Главное в формализме БСП-проверка принадлежности к синтаксической категории (§ 5.3.3). Подлежащие выполнению проверки указываются метками

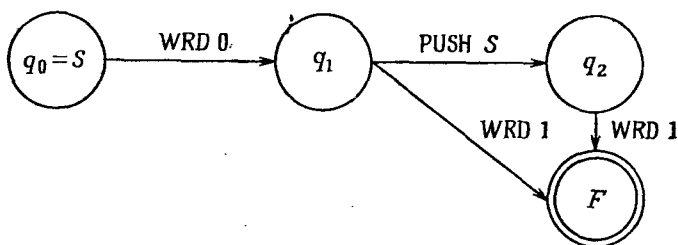
на дугах БП-сетей. Проверяемые синтаксические категории соответствуют таковым в *телах* продукций. Каждому *заголовку* продукции отвечает узел БП-сети. В продукциях КСГ синтаксические категории в заголовках и телах одинаковые. Поэтому при моделировании рекурсии продукций в соответствующих сетях метки дуг и узлов должны совпадать. Зададим поведение РП-сети следующим образом.

Когда на дуге встречается имя узла, то последовательное выполнение операций прерывается и осуществляется переход к этому узлу. Он становится начальным узлом новой сети. Если некая последовательность операций приведет новую сеть в заключительное состояние, то мы вернемся к выполнению прерванной последовательности операций в исходной сети. Ясно, что эта процедура может стать рекурсивной. Новая сеть может потребовать перехода к другой новой сети, и так далее.

По сути это — обращение (возможно, рекурсивное) к подпрограммам. Встреча на дуге с именем узла приостанавливает выполнение программы обращением к подпрограмме, для которой этот узел представляет начальную инструкцию. Введем новую команду, связанную с дугами сети.

- Команда PUSH, примененная к синтаксической категории *S*, прерывает текущие операции, запоминает местонахождение (состояние) системы и переходит в узел с меткой *S*. По достижении заключительного узла, операции возобновляются с того состояния, где были прерваны этой командой.
- Итак, РП-сеть получается из БП-сети добавлением команды PUSH.

КС-грамматики эквивалентны стековым автоматам (см. § 5.2.5), а РП-сети — КС-грамматикам и, следовательно, тем же автоматам. Команда PUSH соответствует добавлению элементов в верхнюю ячейку стека. Множество этих элементов представляет состояние сети при встрече с PUSH. Заход в заключительный узел и возобновление прерванных

Рис. 5.13. РПП для языка $\{0^n 1^n \mid n \geq 1\}$.

операций соответствуют удалению всех элементов из верхней ячейки стека.

На рис. 5.13 изображена РП-сеть для языка $L = \{0^n 1^n \mid n \geq 1\}$, построенная с использованием двух продукций грамматики, указанных в начале параграфа. Предположим, что поступила фраза 0011. Первое слово 0 переводит сеть из начального состояния $q_0 = S$ в q_1 , а команда $PUSH S$ переводит ее в q_2 и создает новую подсеть, идентичную исходной сети. Второе слово 0 переводит подсеть из $q_0 = S$ в q_1 , первое слово 1 переводит подсеть из q_1 в заключительное состояние F . Исходная сеть продолжает работу (реактивируется) с узла q_2 , где она была покинута обращением к команде $PUSH S$. Последнее слово 1 влечет переход из состояния q_2 в заключительное и, в свою очередь, восприятие фразы.

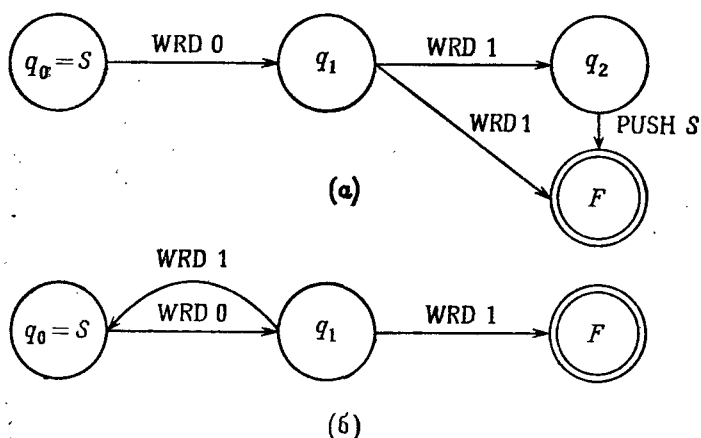
Рассмотрим кратко частный случай — регулярные, или линейные справа, грамматики. Здесь синтаксическая категория в теле правила — крайняя справа (см. § 5.2.2). Команда $PUSH$ заменяется другой, идущей от конца тела одного правила к заголовку другого. Фактически команда $PUSH$ нужна нерегулярным КС-грамматикам. Например, порожденный продукциями

$$S \rightarrow 01$$

$$S \rightarrow 01S$$

простейший регулярный язык $L = \{01\}^+$ воспринимается РП-сетью (рис. 5.14(а)) и эквивалентной БП-сетью (рис. 5.14(б)).

РП-сети и их подсети переводятся в процедуры Лиспа одной функцией и двумя командами:

Рис. 5.14. РСП и БСП для языка $\{01\}$.

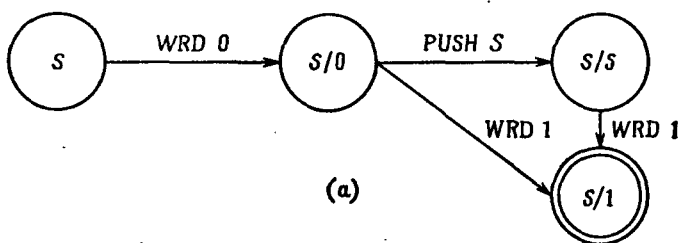
- функция DEFATN (от английского define ATN [113]) создает РП-подсеть компиляцией РП-сети в Лисп. Так, продукции ОК-грамматики путем добавления аргументов переводились в хорновские дизъюнкт-инструкции, выполнимые Прологом (см. § 5.1.8);
- команда PUSH с аргументом-именем РП-подсети инициализирует эту подсеть, запомнив текущее состояние;
- команда POP (без аргументов) находит конец подсети и реактивирует основную сеть с места прерывания.

На рис. 5.15(а), лисповской версии рис. 5.13, приведена РП-сеть восприятия фраз $0^n 1^n$. На рис. 5.15(б) дан ее перевод в список команд.

5.3.6. Пример

На рис. 5.16 БСП с рис. 5.12 описана четыремя РСП. Группы существительных мужского и женского рода описываются подсетями *гсм* и *гсж*, обращающимися к встроенной сети *гсв*. Списки команд:

(DEFATN *фраза*
(*фраза* (CAT *глагол* TO *гс*))



```

( DEFATN S
  (S (WRD 0 TO S/0))
  (S/0 (PUSH S TO S/S)
       (WRD 1 TO S/1))
  (S/S (WRD 1 TO S/1))
  (S/1 (POP )))
  
```

(б)

Рис. 5.15. РСП и программа на Лиспе для $L = \{0^n 1^n \mid n \geq 1\}$.

```

(гс (WRD этот ТО гсмж))
  (WRD эта ТО гсжн))
(гсмж (PUSH гсм ТО конец))
(гсжн (PUSH гсж ТО конец))
(конец (POP)))
(DEFATN гсм
  (гсм (CAT ПМ ТО гс/п))
  (гс/п (CAT СМ ТО гс/с))
  (гс/с (PUSH гсв ТО конец))
  (конец (POP)))
(DEFATN гсж
  (гсж (CAT ПЖ ТО гс/п))
  (гс/п (CAT СЖ ТО гс/с))
  (гс/с (PUSH гсв ТО конец))
  (конец (POP)))
(DEFATN гсв
  (гсв (WRD (в этом) ТО СМ)
        (WRD (в этой) ТО СЖ))
  (СМ (PUSH гсм ТО конец))
  (СЖ (PUSH гсж ТО конец))
  (конец (POP)))
  
```

Модульность способствует наглядности большой сети с повторяющимися компонентами.

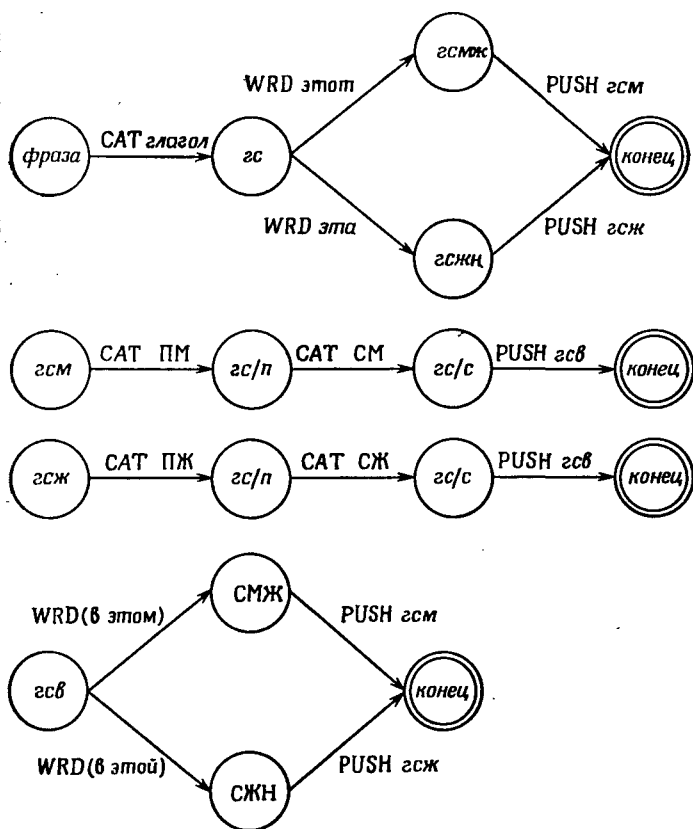


Рис. 5.16. Рекурсивные сети переходов.

5.3.7. Усиленные сети переходов

Язык $L = \{0^n 1^n 0^n | n \geq 1\}$ не порождается КС-грамматикой (§ 5.2.7). Поэтому язык L не воспринимается РП-сетью. Он порождается грамматикой типа I и воспринимается машиной Тьюринга с конечной лентой.

Ограниченность РСП хорошо видна на примере языка, порожденного грамматикой из § 5.1.4 и (затем) очищенного от семантически некорректных фраз. Он конечен, поэтому регулярен, а следовательно,

является КС-языком. Но КСГ эффективно не отсекает абсурдные фразы вроде: «стереть эту вторую строку в этом первом пароле». Придется перейти к более мощным ОК-грамматикам (§ 5.1.5). Последние эквивалентны грамматикам типа 0 и соответствуют машинам Тьюринга.

Порожденные ОК-грамматиками языки типа 0 воспринимаются УП-сетями, получаемыми из РП-сетей (§ 5.3.5) присоединением регистров и действий над ними. Так ОК-грамматики получаются из КС-грамматики присоединением аргументов и связанных с ними проверок. Например, продукция ОК-грамматики (§ 5.1.5)

$$\text{группа_сущ}(Y) \rightarrow M, P, C(Z), \{Y < Z\}, \\ \text{группа_сущ}(Z)$$

позволяет исключить возможность порождения и/или распознавания некоторых абсурдных фраз путем введения проверки условия $\{Y < Z\}$.

Каждую команду перехода в РП-сети снабдим проверкой и последовательностью действий над регистрами. Например, WRD будет иметь вид

$$(\text{WRD} \langle \text{слово} \rangle \langle \text{проверка} \rangle \langle \text{действие} \rangle^* \text{ТО} \langle \text{узел} \rangle),$$

где символ $\langle \text{действие} \rangle^*$ означает последовательность элементарных действий.

Переход осуществится, если $\langle \text{проверка} \rangle$ даст «истинно» и если $\langle \text{слово} \rangle$ совпадает с рассматриваемым. Регистры выполняют действия $\langle \text{действие} \rangle^*$ до перехода в состояние, указанное ТО $\langle \text{узел} \rangle$.

Основные элементарные действия над регистрами следующие:

- (GETR $\langle \text{регистр} \rangle$) : получить значение из регистра с именем $\langle \text{регистр} \rangle$,
 (SETQ $\langle \text{регистр} \rangle \langle \text{знач} \rangle$) : загрузить регистр с именем $\langle \text{регистр} \rangle$ (локальной сети) значением $\langle \text{знач} \rangle$,
 (SENDER $\langle \text{регистр} \rangle \langle \text{знач} \rangle$) : загрузить $\langle \text{регистр} \rangle$ подсети значением $\langle \text{знач} \rangle$ (используется только с командой PUSH).

Проверки — лисповские, например:

(GETR *reg1*) = *b* : значение регистра *reg1* будет *b*,
: булево значение истинно.

Для иллюстрации рассмотрим несколько переходов (функции, приписываемые дугам) УП-сети [4]:

(WRD <слово> <проверка> <действие>* TO <узел>)
(CAT <категория> <проверка> <действие>* TO <узел>)
(PUSH <подсеть> <проверка> <действие>* TO <узел>)
(JUMP <проверка> <действие>*)
(POP <знач> <проверка> <действие>*)

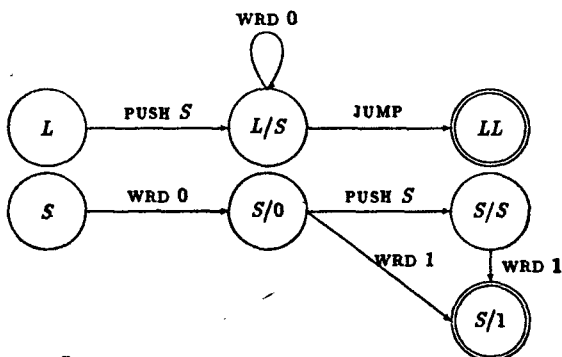
Если <проверка> дает И, то выполняются переход, команда и действия <действие>*. Для POP вычисляется значение <знач> и пересылается в сеть, указанную содержимым специального регистра *.

Регистры можно использовать для согласований в языке между местоимением, прилагательным и существительным, местоимением и глаголом. Местоимение записывает свой род в регистр. Этот регистр проверяется для выбора дуги перехода сообразно группе существительного мужского или женского рода.

5.3.8. Пример

Покажем, что язык $\{0^n 1^n 0^n | n \geq 1\}$ распознается УП-сетью, изображенной на рис. 5.17 и состоящей из основной сети *L* и подсети *S*, где *S* есть РП-сеть для языка $\{0^n 1^n | n \geq 1\}$ с добавленным регистром *N*, служащим для запоминания числа *n*. Пусть сеть *L* получила фразу 001100. Два встроенных обращения к *S* дают восприятие четырех первых слов. Число обращений (здесь 2) к *S* запоминается в *N*. При каждом выходе из *S* команда POP узла *S/1* загружает регистр * вызываемой сети значением регистра *N*, увеличенным на 1, как команда Лиспа (+*N1*). Регистр *N* инициализирован нулем с помощью действия, соответствующего дуге (WRD1 *t* (SETQ *N* 0) TO *S/1*). После каждого перехода (PUSH *S t* (SETQ *N* *) TO *S/S*) регистр *N* как при команде Лиспа (SETQ *N* *) получит через *S* значение из регистра *.

Сеть перешла в узел *L/S*. Регистр *N* содержит число *n* обращений к *S* (указанное выше в оп еделе-



DEFATN S

```

(S (WRD 0 t ( ) TO S/0))
(S/0 (PUSH S t (SETQ N *) TO S/S)
      (WRD 1 t (SETQ N 0) TO S/1))
(S/S (WRD 1 t ( ) TO S/1))
(S/1 (POP (+ N 1) t ( ) )))

```

DEFATN L

```

(L (PUSH S t (SETQ N *) TO L/S))
(L/S (WRD 0 (> N 0) (SETQ N (- N 1))) TO L/S)
      (JUMP (= N 0) ( ) ))
(LL (POP ( ) t ( ) )))

```

Рис. 5.17. УСП для языка $\{0^n 1^n \mid n \geq 1\}$.

нии языка). Пока значение регистра N положительно, выбирается дуга (WRD 0 ($> N 0$) (SETQ $N (- N 1)$) TO L/S), а в узле L/S выбирается петля. При каждой петле из входной фразы извлекается слово 0 и значение регистра N уменьшается на 1 командой Лиспа (SETQ $N (- N 1)$). Когда регистр N обнулится, то подтвердится лисповская проверка ($= N 0$). Тогда будет выбрана соответствующая команде JUMP дуга и система перейдет в заключительный узел LL .

5.3.9. Построение синтаксического дерева

Добавим регистры к РП-сети из примера § 5.3.6 для построения синтаксических деревьев анализируемых фраз. В § 5.1.7 для этого добавлялись аргументы к КС-грамматике из § 5.1.4. Действие BUILDQ производит тот же эффект в УП-сети:

(BUILDQ <структура> <элементы>)

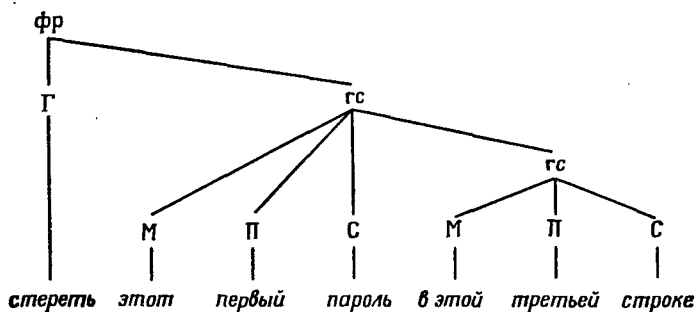


Рис. 5.18. Синтаксическое дерево.

Синтаксическое дерево строится в виде списка посредством собирания *⟨элементов⟩* сообразно *⟨структуре⟩*.

Грамматика, соответствующая сетям, изображенным на рис. 5.16, анализирует фразу «стереть этот первый пароль в этой третьей строке» и дает дерево, представленное на рис. 5.18. Это дерево задается следующим списком:

(фр (Г (стереть) (гс (М этот) (П первый) (С пароль)
(гс (М в этой)
(П третьей) (С строке))))

Благодаря регистрам и команде BUILDQ дерево строится автоматически в ходе синтаксического анализа. Команда

(BUILDQ (гс (М +) (П +) (С +)) РМ РП *)

заменяет три последовательных знака + значениями регистров РМ, РП и *. Спецрегрстр * содержит очередное слово фразы. Если в РМ, РП и * соответственно этот, первый и пароль, то вышеуказанное BUILDQ строит список: (гс (М этот) (П первый) (С пароль)).

Вот представление в формализме УП-сетей грамматики, соответствующей рис. 5.16:

(DEFATN фр 3
(фр (CAT Г t (SETQ РГ *) TO гс))
(гс (WRD этот t (SETQ РМ *) TO м)
(WRD эта t (SETQ РМ *) TO ж))


```

(м      (PUSH гсмже  $t$  ((SENDR  $PM$  (GETR  $MP$ ))
                                (SETQ  $P_{гс}$  *)) TO конец))
(ж      (PUSH гсжн  $t$  ((SENDR  $PM$  (GETR  $MP$ ))
                                (SETQ  $P_{гс}$  *)) TO конец))
(конец (POP (BUILDQ (фр (Г +) +)  $PG$   $P_{гс}$ )  $t$  ( )))
(DEFATN гсмже
(гсм    (CAT ПМ  $t$  (SETQ  $PP$  *) TO гс/п))
(гс/п   (CAT СМ  $t$  (SETQ  $PC$  *) TO гс/с))
(гс/с   (PUSH гсвм  $t$  (SETQ  $P_{гс}$  *) TO конец))
(конец (POP (BUILDQ (гс (М +) (П +) (С +) +)
                                 $PM$   $PP$   $PC$   $P_{гс}$ )  $t$  ( )))
(DEFATN гсжн
(гсж    (CAT ПЖ  $t$  (SETQ  $PP$  *) TO гс/п))
(гс/п   (CAT СЖ  $t$  (SETQ  $PC$  *) TO гс/с))
(гс/с   (PUSH гсвм  $t$  (SETQ  $P_{гс}$  *) TO конец))
(конец (POP (BUILDQ (гс (М +) (П +) (С +) +)
                                 $PM$   $PP$   $PC$   $P_{гс}$ )  $t$  ( )))
(DEFATN гсвм
(гсв    (WRD (в этом)  $t$  (SENDR  $PM$  *) TO СМ)
        (WRD (в этой)  $t$  (SETQ  $MP$  *) TO СЖ))
(СМ     (PUSH гсмже  $t$  ((SENDR  $PM$  (GETR  $MP$ ))
                                (SETQ  $P_{гс}$  *) TO конец))
(СЖ     (PUSH гсжн  $t$  (SETQ  $P_{гс}$  * ((TO конец))
(конец (POP  $P_{гс}$   $t$  ( )))

```

Текущее слово запоминается в некоем регистре посредством SETQ *<регистр>*. На выходе из сети (команда POP) действие BUILDQ строит список, комбинируя содержимое этих регистров. Команда SENDR загружает регистр нижнего уровня PM . На выходе из подсетей $гсмж$ и $гсжн$ содержимое этого регистра можно поместить в список, построенный действием BUILDQ, хотя «местоимение» уже обработано вне подсети.

5.3.10. УП-сети и ОК-грамматики

Как мы видели, формализм УП-сетей тесно связан с Лиспом и переводится в формализм ОК-грамматик (основу Пролога). Формализм УП-сетей базируется на сетевых моделях. Формализм ОК-грамматик использует продукции (правила переписывания). Для перевода УП-сетей в ОК-грамматки достаточно про-

интерпретировать сети и подсети как нетерминальные символы продукций, а дуги извлечения слов — как терминальные символы. Продукции соответствуют различным путям между начальным и заключительным узлами. Петли указывают на рекурсивные обращения к продукциям. Используя такую терминологию, можно сказать, что РП-сети переводятся в КС-грамматики, а регистры — (из УП-сетей) представляются аргументами в продукциях ОК-грамматик.

Связанные с дугами проверки и действия выражимы Прологом, каждый вид дуги в УП-сети представляется последовательностью прологовских целей [84]:

УСП**ОКГ**

(CAT <кат> <проверка> <действие>*...)	[M], {проверка, кат (M), действие}
(WRD <слово> <проверка> <действие>*...)	[слово], {проверка, действие}
(JUMP <проверка> <действие>*)	{проверка, действие}
(PUSH <сеть> <проверка> <действие>*...)	{проверка}, сеть, {действие}
(POP <проверка> <действие>*)	{проверка, действие}

Предикаты Пролога *проверка* и *действие* соответствуют меткам дуги <проверка> и <действие>*. Предикат *кат*(M) проверяет принадлежность слова M к синтаксической категории *кат*. Предикаты помещены в фигурные скобки формализма ОКГ, ибо выражают условия применимости продукций. Следовательно, при переводе продукций ОКГ на Пролог два дополнительных аргумента не понадобятся (см. § 5.1.10) УП-сетям, изображенным на рис. 5.16, соответствуют следующие продукции ОК-грамматики:

$фр(фр(g(G), GC)) \rightarrow глагол(G), гс(GC)$
 $гс(GC) \rightarrow [этот], гс.м(этот, GC)$

<i>гс</i> (<i>ГС</i>)	→ [<i>эта</i>], <i>гсж</i> (<i>эта</i> , <i>ГС</i>)
<i>гсм</i> (<i>М</i> , <i>гс</i> (<i>м</i> (<i>М</i>), <i>п</i> (<i>П</i>),	
<i>с</i> (<i>С</i>), <i>ГС</i>))	→ <i>пм</i> (<i>П</i>), <i>см</i> (<i>С</i>), <i>гсв</i> (<i>ГС</i>)
<i>гсж</i> (<i>М</i> , <i>гс</i> (<i>м</i> (<i>М</i>),	
<i>п</i> (<i>П</i>), <i>с</i> (<i>С</i>), <i>ГС</i>))	→ <i>пж</i> (<i>П</i>), <i>сж</i> (<i>С</i>), <i>гсв</i> (<i>ГС</i>)
<i>гсв</i> (<i>ГС</i>)	→ [<i>в, этом</i>], <i>гсм</i> ([<i>в, этом</i>], <i>ГС</i>)
<i>гсв</i> (<i>ГС</i>)	→ [<i>в, этой</i>], <i>гсж</i> ([<i>в, этой</i>], <i>ГС</i>)
<i>мм</i> (<i>этот</i>)	→ [<i>этот</i>]
<i>мм</i> ([<i>в, этого</i>])	→ [<i>в, этом</i>]
<i>мж</i> (<i>эта</i>)	→ [<i>эта</i>]
<i>мж</i> ([<i>в, этой</i>])	→ [<i>в, этой</i>]
<i>пм</i> (<i>первый</i>)	→ [<i>первый</i>]
<i>пм</i> (<i>второй</i>)	→ [<i>второй</i>]
<i>пм</i> (<i>третий</i>)	→ [<i>третий</i>]
<i>пж</i> (<i>первая</i>)	→ [<i>первая</i>]
<i>пж</i> (<i>вторая</i>)	→ [<i>вторая</i>]
.	.
.	.
.	.

Регистры заменены здесь аргументами предикатов, действие BUILDQ — построением списка в заголовке продукции. Переменные сразу появляются в нужном месте списка (§ 5.1.7). Для предикатов *фр* значение переменной *ГС* вставляется в прологовский список дерева.

Действием SENDR УП-сеть пересылает очередное слово из узла *гс* в подсети *гсм* и *гсж*. Формализм Пролога посылает текущее слово в нетерминальные символы *гсм* и *гсж* посредством дополнительного аргумента.

Приведенный выше словарь представлен последними продукциями ОК-грамматики с телами из одного терминального символа, соответствующего слову словаря. Заголовок продукции отвечает предикату *кат*(*М*), указывающему синтаксической категории слова, а аргумент *М* — регистру * УП-сети с текущим словом для синтаксического дерева.

Дополнительные аргументы могут, если надо, учесть *особенное*, *личное*, *переходное*, *одушевленное*, *нарицательное* и т. д. для слов словаря.

6. Пролог и логическое программирование

6.1. Основы языка

6.1.1. Введение

Эта глава посвящена Прологу — *средству* написания выполнимых на ЭВМ программ. Само название Пролог говорит о том, что это язык *логического программирования*, и оно навеяно формальной логикой (см. гл. 1). На Прологе написано громадное число программ, связанных с искусственным интеллектом (ИИ). Поэтому здесь вполне уместно хотя бы кратко описать этот язык. Приводимое нами описание познакомит читателя с основами Пролога, и, используя руководство программиста, он сможет написать свои первые программы.

У языка логического программирования два главных отличия от классических алгоритмических языков, предназначенных для численных расчетов.

- Это *символьное* программирование: данные в нем суть *символы*, представляющие только самих себя и не подлежащие интерпретации при выполнении программ.
- Алгоритмы получения определенных результатов, непосредственно не задаются, но описываются, объекты, их свойства и отношения между объектами, говорится о *фактах*, *правилах* получения новых фактов, *вопросах*, выясняющих установленность или возможность установления фактов.

Три вида *выражений* (факты, правила и вопросы) интерпретируются как логические предикаты, и тогда этот язык является подмножеством формальной логики (§ 1.1.17, 3.1.21). В то же время Пролог является языком программирования: для получения ответа на

вопрос выполняется некоторый алгоритм. В дальнейшем проявятся (иногда противоречиво) два аспекта: *декларативный* аспект формальной логики и *процедурный* аспект языка программирования.

Очарование Пролога в том, что он определенным образом олицетворяет логику в действии, а именно «выполнение» логических формул.

Например, рассмотрим аксиомы:

0 — натуральное¹⁾ число,
если N — натуральное число, то $s(N)$ — натуральное число.

Программа на Прологе²⁾, описывающая эти аксиомы, такова:

натуральное (0).

натуральное ($s(N)$): — натуральное (N).

Она не только порождает, но и распознает натуральные числа соответственно поставленному вопросу. Например:

? — натуральное ($s(s(s(0)))$).

— — > да

? — натуральное (мед).

— — > нет

? — натуральное (X).

— — > $X = 0$;

— — > $X = s(0)$;

— — > $X = s(s(0))$;

— — > $X = s(s(s(0)))$;

Первый вопрос таков: «Является ли три натуральным числом?» Последний вопрос формулируется следующим образом: «Что такое натуральные числа?» Ответы ясны³⁾. Но чудеса здесь исключены: если хотим, чтобы $s(s(s(0)))$ записывалось числом 3, то это надо также запрограммировать.

¹⁾ Таким образом, здесь и ниже ноль причислен к натуральным числам. — *Прим. перев.*

²⁾ Допускаем строчные и прописные русские буквы. — *Прим. перев.*

³⁾ Си-Пролог отвечает по-английски yes и no.

Программа символьного дифференцирования полиномов должна содержать факт

дифф ($X, X, 1$).

(производная от X по X есть 1). И содержать еще правила

дифф ($U + V, X, A + B$): — **дифф** (U, X, A),
дифф (V, X, B).

дифф ($U - V, X, A - B$): — **дифф** (U, X, A),
дифф (V, X, B).

(производная от $U \pm V$ по X есть $A \pm B$, если A — производная от U и B — производная от V). В таких выражениях Пролога проглядывает образ тернарного отношения «производная ... по ... есть ...». Это *логический* аспект, или декларативное прочтение, Пролога (§ 5.1.8). Аспект *языка программирования*, или процедурное прочтение Пролога, соответствует описанию вычислений, которые должны быть выполнены: «для получения производной от $U \pm V$ вычислить производные от U и от V (A и B соответственно), а затем построить выражение $A \pm B$ ».

Причины двойного прочтения далеко искать не нужно: система ¹⁾ Пролога есть средство доказательства теорем, использующее логический аспект языка. Она пытается обосновать противоречие между отрицанием поставленного вопроса и множеством фактов и правил. Удавшееся опровержение дает контрпримеры, представляющие собой ответ на поставленный вопрос (§§ 1.1.17, 1.2.14, 3.1.19, 5.1.9). Эта стратегия доказательства проще общего метода поиска доказательства и вполне осуществима. Поэтому пользователи принимают ее, руководствуются ею и *программируют* с ее помощью (см., например, [11], [105]).

6.1.2. Термы и объекты

В Прологе объекты (т. е. элементы) *универсума рассуждения* представляются с помощью *термов* так же,

¹⁾ Здесь понятие система означает автомат, реализующий Пролог. На практике это ЭВМ, снабженная компилятором или интерпретатором.

как в логике (§§ 1.2.2—1.2.3). В силу того что язык эти термы не интерпретирует, объекты сами *являются* термами — синтаксическими объектами одной из следующих категорий:

- *индивидуальные константы (атомы),*
- *переменные,*
- *функции (функциональные термы),* состоящие из имени функции и списка аргументов-термов.

Синтаксис различает атомы, переменные и функции. Используемые соглашения сообразны конкретным применениям. Здесь мы используем синтаксис Си-Пролога. Однако выбор конкретной версии языка Пролог не влияет на различие между переменной (именем нарицательным) и константой (именем собственным).

- Атом записывается тремя способами:
 - как идентификатор, начинающийся со строчной буквы

жаклин генрих_4

(допускается использование подчеркивания)

- как число

123 1.23

- как произвольную последовательность символов, расположенную между апострофами:

'Что Вы говорите?' 'Больше ничего не знаю.'

- Переменная — это идентификатор, начинающийся с прописной буквы

X Имя Король_Франции

- Функция — это имя функции, за которым следует список термов, помещенных в скобки и разделенных запятыми. Имя функции — это нечисловой атом.

автор(книга,1987) f(X) 'Что ты говоришь?' (штука)

На практике прологовская программа представляет собой некую действительность, а термы — реальные объекты (разд. 3.1). Например, при описании

библиотеки атомами будут имена авторов и/или издателей, годы издания, названия и т. д., а функция — издательства и/или книги:

издат (дюно, 1987)

**книга (жюль_верн, мишель_строгофф,
издат (этцель, 1876))**

Каждый атом — это отдельный объект, считающийся элементарным. Предпочтительны мнемонические идентификаторы, а не анонимные (вроде x_2 и bb). То же относится и к функциональным термам: лучше взять **издат** (____, ____), а не **и** (____, ____). Отметим, что в программе на Прологе функциональный терм является структурой данных, сложным объектом, который можно эффективно построить или анализировать (§ 6.3.9). Это не функция, сопоставляющая результат набору аргументов.

Данные (константы) Пролога — это термы, не имеющие переменных. В логике они называются *индивидуальными термами* (а также *константными термами*). Числовые атомы — это константы для программирования численных расчетов (§ 6.3.2).

6.1.3. Факты и элементарные вопросы

Простые предикаты (атомарные формулы и/или предикатные формы, см. § 1.2.1) формальной логики, такие, как

Автор (Эрнани, Гюго)

принимают значения истинности **И** или **Л**. Почти то же самое имеет место для фактов и вопросов Пролога, не содержащих переменных. Простой предикат Пролога записывается в виде функционального терма, например:

автор (эрнани, гюго)

дифф (X, X, 1)

Предикат от функции отличают по контексту. Простые предикаты являются составляющими базы фактов и вопросов. Приведем пример небольшой программы, представляющей собой множество фактов:

/* библио */

книга (грэм, 'рассуждать, чтобы программировать',

издат (дюно, 1986)).

книга (кондиляк, 'пролог', издат (дюно, 1986)).

книга (дъедонне, 'математика', издат (эрман, 1986)).

книга (гюго, 'отверженные', издат (пош, 1984)).

книга (гюго, 'эрнани', издат (галлимар, 1974)).

книга (хартман, 'параллельный паскаль',
издат (шпрингер, 1977)).

библиотекарь (эмиль).

начальник (эмиль, анри).

начальник (жозеф, анри).

идет_дождь.

Точка, стоящая после предиката, указывает на то, что рассматриваемое выражение является фактом. Первая строка — это *комментарий*: любая последовательность символов, записанная между парой ограничителей /* и */, при выполнении игнорируется. Каждый факт, содержащийся в программе, имеет соответствующее значение истинности и порождает (определяет) отношение между термами. Например, двухместное отношение (предикат) *начальник* установлено между термами *эмиль* и *анри*, а трехместное отношение *КНИГА* — для 6 троек термов и т. д. Последний факт в приведенном примере — *идет дождь*. — является нульместным отношением (не имеющим аргументов).

Множество фактов можно рассматривать как реляционную БД. Впрочем, мы употребляем понятие *базы данных* для обозначения множества фактов и правил некой программы.

Из простых предикатов строят также вопросы, например:

?—начальник(эмиль, анри).

Это выражение нового факта не устанавливает, но «система запрашивается» о том, установлен или нет данный факт. Значение вопроса (т. е. ответ) зависит от БД. В нашем примере вопрос простой, переменные и правила отсутствуют. Значение есть *И*, если в БД содержится факт с предикатом вопроса. В противном случае значение есть *Л*. Все это интуитивно ясно. На

практике задается последовательность «вопрос-ответ»:

?— начальник (эмиль, анри).

— — > да

Таким образом, простейшие программы на Прологе регистрируют элементарные факты в БД и отвечают на вопросы, связанные с этими фактами (§ 5.1.8). Сама же БД определяет некие расширения содержащихся в ней отношений. Например: идет_дождь,

начальник (____, ____), библиотекарь (____) и книга (____, ____, ____). Простые вопросы, не содержащие никаких переменных, называются *да-нет-вопросами*. Они допускают лишь два возможных ответа в соответствии с тем, имеется или отсутствует подходящий факт в БД. Семантика вопроса определяется состоянием БД при поиске ответа. Для БД библиотеки имеем:

?— книга (грэм, 'рассуждать, чтобы программировать', издат (дюно, 1986)).

— — > да

?— книга (рассуждать, чтобы программировать, издат (дюно, 1986)).

— — > нет

?— начальник (эмиль, генри).

— — > нет

?— идет_дождь(сейчас).

— — > нет

Так как в Прологе¹⁾ нет объявления и контроля типов, то нет и ошибок программирования. Неверно написанное пользователем имя для системы является новым атомом: *анри* и *генри* — два атома, которые не имеют ничего общего. Бинарное отношение не годится в качестве тернарного, зато тернарное отношение задает три бинарных отношения с одинаковыми именами — для трех пар из трех аргументов: отношения *книга* (____, ____), *начальник* (____, ____), *идет_дождь* (____, ____), для системы различны, чем и обусловлен ответ на второй приведенный выше вопрос.

¹⁾ Но не в Турбо-Прологе. — Прим. перев.

6.1.4. Конъюнкция

В логических формулах используются связки и простые предикаты (§ 1.1.2). То же самое имеет место в Прологе. Самую ходовую связку — конъюнкцию — обозначают запятой:

? — начальник (эмиль, анри),
начальник (жозеф, анри).

Это выражение соответствует такой логической формуле:

Начальник (Эмиль, Анри) \wedge
 \wedge Начальник (Жозеф, Анри).

Конъюнкция в Прологе, как и в логике, истинна только при истинности всех компонент. Однако имеется важное отличие: в Прологе семантика учитывает порядок оценки компонент (слева направо).

6.1.5. Переменные

Использование переменных в Прологе аналогично, но не идентично использованию их в логике. Вопросы, включающие переменные, носят *перечислительный характер* (в отличие от да-нет-вопросов): ответы в этом случае представляют собой *списки термов*. Например, вопрос «Кто в подчинении у Анри?» является вопросом «перечислительного» («списочного») типа. Ответ на него должен быть списком имен. Ответом БД библио мог бы быть [эмиль, жозеф].

Прологовским эквивалентом местоимения «кто» является переменная:

? — начальник (X, анри).

Ответ здесь хотелось бы получить посредством замены X на такую константу, для которой в БД найдется подходящий факт. Взяв константу эмиль, получаем:

? — начальник (X, анри).
— — > X = эмиль

Система ответила одним значением переменной, преобразовав вопрос в истинный предикат. Но почему не

жозеф и не [**эмиль, жозеф**]? Для объяснения произвольности выбора внимем в алгоритм получения ответа. Факты и правила БД — это не множество, а список. Обычно они текстуально *упорядочены*. Для получения ответа система просматривает БД в соответствующем порядке и выбирает первое удовлетворяющее предикату вопроса выражение.

Предикат вопроса представляет собой *цель*. Здесь это

начальник(Х,анри).

Цель *достигнута*, если в БД удалось найти факт или правило, который (которое) удовлетворяет этому предикату¹⁾ (т. е. превращает его в истинное высказывание). Чтобы удовлетворить приведенному только что простому предикату, нужно иметь факт вида **начальник (_____ ,анри)**, т. е. факт, содержащий:

- такое же имя предиката (**начальник**),
- столько же аргументов (два),
- те же константы на тех же местах (**анри** на втором месте).

При выполнении этих условий факт *соответствует* (следовательно, удовлетворяет) предикату. В нашем примере годится факт

начальник(эмиль, анри).

Переменная **Х** примет значение той константы, которая стоит на соответствующем месте в найденном факте. Хотя объяснять несколько долго, но это именно то, что подсказывает интуиция.

Еще один способ выразить семантику рассмотренного выше вопроса состоит в формулировании следующего запроса: «Можно ли сопоставить (присвоить, назначить) переменной **Х** такой терм, чтобы результатом была формула из БД?» Утвердительный ответ обеспечивает назначение **Х = эмиль**. Это лишь одно из возможных назначений-ответов. Си-Пролог в интерактивном режиме даст все ответы, если после

¹⁾ Будем говорить также: предикат удовлетворяется (удовлетворен) фактом (или правилом). — *Прим ред.*

каждого ставить точку с запятой:

?— начальник (X, анри).

— — > X = эмиль;

— — > X = жозеф;

— — > нет

Нет означает исчерпание возможностей.

Если вопрос

?— начальник (X, анри).

считать обращением к процедуре, то константа **анри** будет входом, а переменная **X** — результатом. Это взгляд пользователя, но не Пролога. В выражениях, содержащихся в БД, различие между аргументами не проводится. Лишь в вопросах переменные «заказывают» значения для ответов, например:

?— начальник (жозеф, Y).

— — > Y = анри;

— — > нет

?— начальник (X, Y).

— — > X = эмиль, Y = анри;

— — > X = жозеф, Y = анри;

— — > нет

Переменные можно также использовать в фактах:

начальник (X, адемар).

Это выражение является записью на Прологе фразы

«Адемар — большой начальник»

и логической формулы

$\forall x$ Начальник (x, Адемар).

Цель вида **начальник** (____, адемар) достигается в том случае, если в БД содержится факт **начальник** (X, адемар).

При использовании переменной в конъюнкции переменная устанавливает связь между сомножителями конъюнкции. Разумеется, все вхождения переменной принимают одно и то же значение. Ответ на вопрос

?— начальник (X, анри), библиотекарь (X).

(«Над какими библиотекарями Анри начальник?») получается путем последовательного удовлетворения двух частей (достижения двух *частичных целей, подцелей*) конъюнкции. Первая часть

начальник (X, анри)

удовлетворяется фактом

начальник (эмиль, анри).

реализующим присваивание (назначение)

X = эмиль.

Если бы первая подцель не достигалась, то ответ на весь вопрос был бы — **нет**. Присваивание значения переменной X преобразует вторую подцель

библиотекарь (X)

в

библиотекарь (эмиль)

которая достигается. Полный ответ:

— — > **X = эмиль.**

Для получения второго ответа система сперва ищет второе достижение второй подцели. Это всегда

библиотекарь (эмиль).

при дальнейшем пробегании по БД в предписанном порядке. (Дублирование выражений запрещено.) Подцель теперь не достигается вновь. Система возвращается к предыдущей подцели

начальник (X, анри).

т. е. осуществляется *возврат* (см. § 6.2.3). Подцель вновь достигается при использовании факта

начальник (жозеф, анри).

Отсюда получаем присваивание

X = жозеф.

и т. д. В рассмотренном нами примере только один ответ, ибо недостижима цель

библиотекарь (жозеф).

средствами (фактами) БД **библио.**

6.1.6. Анонимные переменные

Переменные, которые рассматривались выше, имели имена. Существуют и анонимные переменные. Допустим, что мы интересуемся наличием в БД **библио** хотя бы одной книги Виктора Гюго. Вопрос

?— книга (гюго, X , Y).

вызовет последовательность ответов

- — > X = отверженные, Y = издат (пош, 1984);
- — > X = эрнани, Y = издат (галлимар, 1974);
- — > нет

Это не совсем то, что надо: — предполагалось получить «да» или «нет». Однако, заменяя X и Y анонимными переменными (знак подчеркивания), получаем то, что хотели: на вопрос

?— книга (гюго, ,).

ответ будет

- — > да

Для получения списка авторов, книги которых имеются в наличии, ставим следующий вопрос:

?— книга (A , ,)

Это дает нам такой список

- — > A = грэм;
- — > A = кондияк;
- — > A = дьедоние;
- — > A = гюго;
- — > A = гюго;
- — > A = хартман;
- — > нет

Анонимные переменные не отличаются от обычных при поиске соответствий, но не принимают значений и не появляются в ответах. Отметим, что различные вхождения знака подчеркивания означают различные анонимные переменные.

6.1.7. Правила

Последним и самым мощным видом выражений в Прологе являются *правила*¹⁾ (обобщающие понятие факта). Например, для построения семейной БД можно перечислить родительские отношения при помощи списка фактов:

мать (каролина, юлия).

мать (каролина, альбертина).

Выражать фактами отношения внуков с дедами необходимости нет, ибо они могут быть выведены из отношений предшествования « X является дедом Y , если существует субъект Z , отец которого X и который сам отец или мать Y ». На Прологе это высказывание записывается в виде двух правил:

дед (X, Y) : — отец (X, Z), мать (Z, Y).

дед (X, Y) : — отец (X, Z), отец (Z, Y).

где символ «:—» читается «если» («при условии, что»). Этот символ уже использовался и имел похожий смысл (§§ 1.1.17 и 5.1.4).

Вообще правило выглядит так:

$H :— P_1, P_2, \dots, P_n$.

где H, P_1, P_2, \dots, P_n — простые предикаты (§ 6.1.9). Это выражение читается следующим образом: « H , если (P_1 и $P_2 \dots$ и P_n)». Предикат H называется *заголовком правила*, а последовательность предикатов P_1, P_2, \dots, P_n — *телом правила* (см. § 5.1.3). Такие правила похожи на хорновские дизъюнкты, которые рассматривались в логике (§§ 1.1.17, 5.1.4).

Приведенное только что правило является аналогом²⁾ предложения

H , если $(P_1 \wedge P_2 \dots \wedge P_n)$,

¹⁾ В литературе правила называют также *импликациями*. — Прим. перев.

²⁾ Именно «аналогом», а не «эквивалентом», так как возможны различия из-за порядка присваивания значений и проявления «краевых» эффектов, связанных с оператором отсечения (см. § 6.2.5).

или выражения

$$(P_1 \wedge P_2 \dots \wedge P_n) \supset H,$$

или формулы

$$\neg P_1 \vee \neg P_2 \dots \vee \neg P_n \vee H,$$

которые суть различные записи хорновских дизъюнктов. Приведенное правило похоже, кроме того, на объявление процедуры в языке программирования.

Заголовок правила

дед (X, Y)

объявлен как и тело правила

отец (X, Z), мать (Z, Y).

Это правило указывает на то, что вопрос, имеющий форму

?— дед (андре, каролина).

преобразуется в вопрос

?— отец (андре, Z), мать (Z, каролина).

Наличие правила (как и факта) в БД означает общезначимость соответствующей формулы. Считается, что все переменные в правиле связаны кванторами общности. Например, правило

дед (X, Y) :— отец (X, Y), мать (Z, Y).

толкуется как

$$\forall x \forall y \forall z [\text{Отец}(x, z) \wedge \text{Мать}(z, y) \supset \text{Дед}(x, y)].$$

Правила — самые общие выражения Пролога, факт представляет собой правило без правой части, а вопрос — без левой (§ 5.1.8):

Н истинно, если P1 и P2 и ... и Pn истинны (правило),

Н истинно (факт или правило без тела),

P1 и P2 и ... и Pn истинны? (вопрос или правило без заголовка).

Рассмотрим следующую БД:

/* семья*/

мать (каролина, юлия).

мать (каролина, альбертина).

мать (мари, проспер).

мать (анна, каролина).

отец (проспер, юлия).

отец (проспер, альбертина).

отец (альфонс, проспер).

отец (андре, каролина).

дед (X, Y):— отец (X, Z), мать (Z, Y).

дед (X, Y):— отец (X, Z), отец (Z, Y).

бабка (X, Y):— мать (X, Z), мать (Z, Y).

бабка (X, Y):— мать (X, Z), отец (Z, Y).

Отношения **мать** и **отец** заданы фактами, а отношения **дед** и **бабка** определяются с помощью правил.

На вопрос

?— дед (альфонс, юлия).

ответа среди фактов нет (ни один факт не соответствует предикату вопроса). Но заголовки некоторых правил соответствуют этому предикату.

Цель заголовка какого-либо правила считается достигнутой (как и для факта) при совпадении имени предиката, числа аргументов в нем и констант, стоящих на соответствующих местах предиката. Так, например, факт

дед (альфонс, юлия)

обеспечивает реализуемость (удовлетворение, достижение) цели

дед (X, Y).

Переменным из заголовка правила присваиваются значения:

X = альфонс, Y = юлия.

Новой целью будет далее тело с новыми значениями переменных:

отец (альфонс, Z), отец (Z, юлия).

Эта конъюнкция предикатов обрабатывается как последовательность подцелей. Их достижение обеспечивается присваиванием

Z = проспер.

В итоге получаем последовательность вопрос-ответ
 ?— дед (альфонс, юлия).

— — > да

Значение внутренней (отсутствующей в вопросе) переменной не входит в ответ. Это вполне естественно, ибо вопрос «да-нет» и переменные X , Y , Z нужны только для вычисления (нахождения) ответа.

6.1.8. Рекурсивные правила

База данных семья должна отвечать на вопрос, является ли X предком Y . Определение предиката «быть предком» рекурсивно. Для предков по женской линии соответствующие правила на Прологе записываются так:

предок (X , Y):— мать (X , Y).

предок (X , Y):— мать (X , Z), предок (Z , Y).

Такие рекурсии разрешается использовать, иначе язык был бы слишком примитивным. Примеры еще будут приведены. Рекурсивное определение предиката обязательно должно содержать базис (§ 5.1.2), т. е. нерекурсивную часть, иначе оно будет логически некорректным и программа заикнется. Чтобы избежать заикливания, надлежит, кроме того, позаботиться о порядке выполнения, а следовательно, и порядке написания соответствующих выражений. Поэтому практически полезно, а порой и необходимо придерживаться принципа: «сначала нерекурсивные выражения».

6.1.9. Дизъюнкция

Чистый Пролог разрешает применять только конъюнкцию в вопросах и правилах (хорновских дизъюнктах). Язык, используемый на практике, богаче: в нем допускаются дизъюнкция и отрицание в телах правил и вопросах (т. е. целях, которые исследуются на достижимость). Рассмотрим снова БД библио и следующие выражения со связкой дизъюнкции, обо-

значаемой точкой с запятой:

начальник (X, анри); библиотекарь (X)

Оно является аналогом логической формулы

Начальник (x, Анри) \vee Библиотекарь (x).

Для достижения цели, содержащей дизъюнкцию, система сперва пытается удовлетворить левую часть рассматриваемой дизъюнкции, а если это не удастся, то переходит к поиску удовлетворения для правой части той же дизъюнкции. Для нашего примера имеем:

?— **начальник (X, анри); библиотекарь (X).**

— —> X = эмиль;
 — —> X = жозеф;
 — —> X = эмиль;
 — —> нет

В этом простом примере дизъюнкция не включена ни в какой более сложный предикат и поэтому все дело свелось к двум последовательным вопросам. Первый предикат удовлетворен дважды, второй — один раз (причем независимо от первого).

6.1.10. Отрицание

В Прологе отрицание имеет имя **not** и для представления отрицания какого-либо выражения **P** используется запись **not (P)**. Цель **not (P)** достижима тогда и только тогда, когда не удовлетворяется предикат (цель) **P**. При этом переменным значения не присваиваются. В самом деле, если достигается **P**, то не достигается **not (P)**; значит, надо стереть все присваивания, приводящие к этому результату. Наоборот, если **P** не достигается, то переменные не принимают никаких значений. Рассмотрим пример, связанный с БД библио:

?— **not (начальник (эмиль, арсен)).**

— —> да

?— **not (начальник (эмиль, X)).**

— —> нет

Чтобы обработать отрицание, Пролог рассматривает невозможность проведения доказательства как эквивалент значения **Л**. Следовательно, **not(P)** считается истинным тогда и только тогда, когда не удовлетворяется **P**. Подход этот в высшей степени прагматический, не имеющий эквивалента в логике. (А что можно предложить иное?)

6.1.11. Области действия имен

В Прологе программист свободен в выборе имен констант, переменных, функций и предикатов. Исключения составляют резервированные имена (§ 6.2.4) и числовые константы (§ 6.3.2). Переменные не объявляются, отличаясь от констант первой буквой ¹⁾. Разные обозначения представляют разные объекты. Есть довольно очевидные исключения. Например, 133.1 и 133.10 — одно и то же число.

Область действия имени представляет собой часть программы, где это имя имеет один и тот же смысл (как во всех языках программирования). В Прологе:

- для переменной областью действия является выражение (факт, правило или вопрос), ее содержащее,
- для остальных имен (констант, функций или предикатов) — вся программа.

Множественно можно использовать в одной программе лишь имена переменных. В приводимом ниже фрагменте из БД **семья** имеются: два вхождения имени **каролина** — это одна и та же константа, четыре вхождения имени (предикатного) **мать** — один и тот же предикат (всюду двухместный). Но переменные первого и второго правил независимы (это интуитивно ясно ²⁾).

¹⁾ Строчная или прописная. — *Прим. перев.*

²⁾ Действительно, считается, что каждая переменная находится под действием некоторого (своего) квантора, который можно поместить в начале рассматриваемого выражения. Таким образом, область действия имени переменной в Прологе аналогична области действия квантора (см. § 1.2.4).

мать (каролина, юлия).

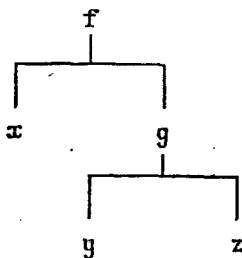
мать (каролина, альбертина).

дед (X, Y) :— отец (X, Z), отец (Z, Y).

бабка (X, Y) :— мать (X, Z), мать (Z, Y).

6.1.12. Операторы

Напомним, что функциональный терм — это имя функции с аргументами в скобках. Само имя функции представляет собой нечисловой атом. Вообще же нечисловой атом — это функциональный терм без аргументов. Любой терм можно представить в виде дерева; корню которого приписано имя внешней функции, а ветвям соответствующие наборы аргументов. Например, терм $f(x, g(y, z))$ задается следующим деревом:



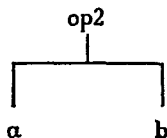
Такое представление является канонической формой: она не зависит от способа записи термов. Каноническую форму полезно бывает использовать в тех случаях, когда требуется уточнить представление термов (некоторые термы могут быть записаны многими способами). В частности, для термов с одним или двумя аргументами функциональное обозначение можно заменить именем операции, причем имя функции-операции записывается как унарный префиксный (или постфиксный) оператор, либо как бинарный инфиксный оператор:

op1 c a op2 b

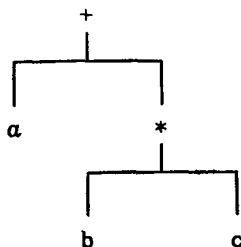
вместо

op1 (c) op2 (a, b).

При этом именам функций-операций (**op1** и **op2**) надо придать статус *операторов* с приоритетом (см. § 6.3.10). Очевидно, что выражения **a op2 b** и **op2(a, b)** задают один и тот же объект:



Некоторые имена функций и предикатов можно записывать в виде операторов. Например, **a + b * c** и **+(a, *(b, c))** — один и тот же терм; его дерево выглядит так:



Приоритет «+» ниже, чем у «*» (как обычно). Подчеркнем, что **a + b * c** — терм Пролога, а не числовой оператор, описывающий процедуру вычисления (см. § 6.3.2).

6.2. Алгоритмы Пролога

6.2.1. Введение

Общий принцип выполнения прологовых программ прост: ищут (вычисляют) ответы на вопросы, задаваемые базе данных (БД), состоящей из фактов и правил. Проверяется соответствие предикатов вопроса выражениям из БД. Это частный случай метода резолюций (§§ 1.1.12 и 1.2.14). Но детали не так просты. Результаты проводимого поиска нужно анализировать во вполне определенном порядке, применяя

для проверки соответствий спецфункции Пролога (делающие его языком программирования). Наиболее примечательной из этих функций (обладающей так называемым «краевым эффектом») является *отсечение* (§ 6.2.5).

Для получения *ответа* на заданный *вопрос* ищется подходящее *соответствие* в БД (§ 5.1.8). Например, рассмотрим БД *библио* и вопрос

? — *начальник (эмиль, Y)*.

Предикат, содержащийся в этом вопросе, становится *целью*. Соответствие ему факта

начальник (эмиль, анри)

влечет достижение цели и ответ

— — > Y = анри

Основные принципы и термины следующие:

- в задаваемом вопросе допускается только один предикат, он и является *целью*, *удовлетворяемой* выражением (фактом или правилом) из БД; в приведенном выше примере целью является выражение *начальник (эмиль, Y)*;
- если цель есть атомарный предикат, то она удовлетворяется
 - фактом — в том случае, когда он соответствует ей (см. § 6.2.2); в нашем примере цели соответствует факт *начальник (эмиль, анри)*;
 - правилом — в том случае, когда его заголовок соответствует цели, а его тело может быть удовлетворено посредством присваивания термов некоторым переменным;
- если цель представляет собой конъюнкцию (последовательность) предикатов, то она удовлетворяется в том случае, когда посредством присваиваний термов переменным будут последовательно (в текстуальном порядке) удовлетворены все входящие в нее предикаты.

Отметим, что в Прологе не проводится синтаксического различия между предикатом и функциональ-

ным термом, а также между нечисловой константой и 0-арной функцией. Следовательно, суть действия состоит в том, что ищется попарное соответствие между термами, один из которых является целью, а другой принадлежит БД.

6.2.2. Соответствие и унификация

Установление соответствия между термами является основной операцией при поиске (вычислении) ответа на вопрос. Она осуществляется следующим образом: на каждом шаге выбирается очередной терм и отыскивается соответствующее выражение в БД. При этом переменные получают или теряют значения. Этот процесс естественно описать в терминах текстуальных подстановок: выражение «присвоить переменной Y значение *анри*» преобразуется в выражение «подставить терм *анри* вместо переменной Y ». Для эквивалентности этих двух формулировок нужно использовать различные обозначения для разных переменных.

Заметим, что одни предикаты бывают шире других: они «покрывают» больше случаев. Например, предикат 'любит_читать' (жюль_вери, Все) шире, чем предикат 'любит_читать' (жюль_вери, 'Дети капитана Гранта'). Второй предикат — конкретизация первого (второй уже первого). Для термов такое понятие рассматривалось в § 1.2.6 и для его обозначения использовался символ \prec . Имеем:

Любит_читать (Жюль_Верн, 'Дети капитана Гранта')
 \prec *Любит_читать* (Жюль_Верн, x).

Некоторые из нижеследующих определений были сформулированы в § 1.2.6:

- Подстановка — это функция $\sigma: V \rightarrow T$, где V — множество переменных, а T — множество термов.
- Переменная $X \in V$ называется *активной* для подстановки σ , если $\sigma(X) \neq X$.
- Подстановка называется *элементарной*, если в ней ровно одна активная переменная.

- Подстановка называется *конечной*, если в ней конечное число активных переменных. (Далее мы рассматриваем только такие подстановки.)
- Множество составляющих подстановку пар обозначается ¹⁾ так:

$$\sigma = \{X_1 = t_1, X_2 = t_2, \dots, X_n = t_n\},$$

где X_i — различные активные в σ переменные, t_i — соответствующие термы (следовательно, $t_i = \sigma(X_i) \neq X_i$).

- Применение подстановки σ к выражению или множеству выражений E состоит в замене всех вхождений каждой переменной X термом $\sigma(X)$. Результат обозначают $\sigma[E]$.

- Выражение E_1 есть конкретизация выражения E , если найдется такая подстановка σ , что $E_1 = \sigma[E]$. Здесь E шире E_1 .

Важнейшим определением является определение *унификации* (или *сопоставимости*) двух термов.

- Термы t_1 и t_2 *унифицируемы* (или *сопоставимы*), если существует такая подстановка σ , что $\sigma[t_1] = \sigma[t_2]$. Эта подстановка σ называется *унификатором*, а результат — *общей унификацией* для термов t_1 и t_2 .

Например, подстановка

$$\sigma = \{X = \text{жонас}, Y = W, U = \text{жонас}, V = W, E = \text{жонас}\}$$

унифицирует термы

книга(X, Y , издат(X , 1985)),

книга(U, V , издат(E , 1985)).

Общая унификация такова:

книга($\text{жонас}, W$, издат(жонас , 1985)).

¹⁾ Эта запись напоминает присваивание значений переменным. В гл. 1 применялось другое обозначение:

$$\sigma = \{(X_1, t_1), (X_2, t_2), \dots, (X_n, t_n)\}.$$

Унифицировать можно по-разному. Вот еще два унификатора для тех же термов:

$$\sigma_1 = \{X = \text{жонас}, Y = \text{карл}, U = \text{жонас}, \\ V = \text{карл}, E = \text{жонас}\},$$

$$\sigma_2 = \{X = \text{жонас}, Y = Z, U = \text{жонас}, \\ V = Z, E = \text{жонас}\}.$$

Они дают следующие общие унификации

книга (жонас, карл, издат (жонас, 1985)),

книга (жонас, Z, издат, (жонас, 1985)).

Подстановки σ и σ_2 дают более общие унификации, чем подстановка σ_1 . На самом деле отличие σ и σ_2 — лишь в именах переменных. Мы видели (в § 1.2.13), что если существует унификация для любых двух термов, то существуют и максимально возможные унификации. Они отличаются между собой только именами переменных. Так как эти имена произвольны, то для наиболее общих унификаций мы можем задать каноническую форму, именуя переменные слева направо: **A1, A2, . . .**. Разумеется, что все вхождения одной и той же переменной получают одно и то же имя. Для нашего примера имеем:

книга (жонас, A1, издат (жонас, 1985)).

Следовательно, существует единственный с точностью до имен переменных наиболее общий унификатор. Его можно определить следующим образом:

- *Наиболее общим унификатором* (короче: **НОУ**) для двух унифицируемых термов t_1 и t_2 называется такой унификатор σ , что, каков бы ни был другой унификатор σ_1 для тех же термов, существует подстановка σ_2 , удовлетворяющая условию $\sigma_1 = \sigma_2 \circ \sigma$.

Любой унификатор получается из **НОУ** путем воздействия на него подходящей подстановкой. На рис. 6.1 приведено операционное описание унификации в виде рекурсивного алгоритма *унифицирование*. Входом являются два терма, которые надо унифицировать. Результат — пара (B, σ) , где B — *булево*

процедура унифицирование (t_1, t_2 : терм)

задано (B : булево, σ : последовательность подстановок)

$t_1 = t_2$

\Rightarrow присвоить ($B = \text{истинно}$, $\sigma = \{ \}$)

t_1 есть переменная X

\Rightarrow присвоить ($B = \text{истинно}$, $\sigma = \{X = t_2\}$)

t_2 есть переменная Y

\Rightarrow присвоить ($B = \text{истинно}$, $\sigma = \{Y = t_1\}$)

$[t_1$ есть функция $f(a_1, a_2, \dots, a_n)$

$\wedge t_2$ есть функция $g(b_1, b_2, \dots, b_m)$

$\wedge f = g$

$\wedge n = m]$

\Rightarrow [пусть i таково, что $a_1 = b_1, \dots, a_{i-1} = b_{i-1}$,
 $a_i \neq b_i$;

унифицировать (a_i, b_i) с результатом (B_1, σ_1);

$B_1 = \text{ложно} \Rightarrow$ присвоить ($B = \text{ложно}$, $\sigma = \{ \}$);

унифицировать ($\sigma_1[t_1], \sigma[t_2]$) с результатом
(B_2, σ_2);

$B_2 = \text{ложно} \Rightarrow$ присвоить ($B = \text{ложно}$, $\sigma = \{ \}$);

иначе \Rightarrow присвоить ($B = \text{истинно}$, $\sigma = \sigma_2 \circ \sigma_1$)

]

иначе

\Rightarrow присвоить ($B = \text{ложно}$, $\sigma = \{ \}$)

□

Рис. 6.1. Алгоритм унификации.

значение истинности соотносимо успеху или неудаче унификации, σ — НОУ для двух термов (точнее, σ есть последовательность элементарных подстановок). При процедурном взгляде на подстановки $\sigma_2 \circ \sigma_1$ — не только композиция, но и последовательное применение подстановок к термам t и $\sigma_1[t]$; в результате получается терм $(\sigma_2 \circ \sigma_1)[t] = \sigma_2[\sigma_1[t]]$.

Замечания к рис. 6.1:

- Данный алгоритм является инструкцией, выделяющей пять случаев вида $C_i \Rightarrow I_i$, где условия C_i проверяются в текстовальном порядке.
- Константы и индивидные¹⁾ термы унифицируемы, только если они равны. Первый случай учитывает все ситуации с равенством. Результат — тождественная подстановка $\sigma = \{ \}$.
- Во втором и третьем случаях возможно заикливание при подстановке вида $\{X = f(X)\}$.

¹⁾ Иное название — константные (см. § 6.1.2). — Прим. перев.

Будут порождаться термы $f(f(X))$, $f(f(f(X)))$, $f(f(f(f(X))))$ и т. д. Результат унификации можно рассматривать как бесконечный терм вида $f(f(f(...)))$, но такого объекта нет ни в классической логике, ни в классическом Прологе. Отсюда следует, что надо запретить замещать переменную термом, ее содержащим. Значит, в принципе надо проводить соответствующую проверку вхождения. Однако на практике такая проверка обычно не делается из-за ее трудоемкости.

6.2.3. Вычисление ответа

Каноническая форма вопроса является конъюнкцией атомарных предикатов, т. е. последовательностью подцелей, разделенных запятыми:

$$Q = Q_1, Q_2, \dots, Q_n.$$

На рис. 6.2 представлен рекурсивный алгоритм *вычислить_ответ*, использующий такую форму вопроса¹⁾. Напомним, что выражения, содержащиеся в БД, — это правила вида

$$t_0 :- t_1, t_2, \dots, t_k$$

и факты (правила с пустым телом). Пустую последовательность обозначим символом ϵ . Пробегая БД по

процедура *вычислить_ответ* (Q_1, Q_2, \dots, Q_n) =
для всех выражений $t_0 :- t_1, t_2, \dots, t_k$ из БД
делать

унифицирование (Q_1, t_0) с результатом (B, σ);
 $B = \text{истинно}$

$\Rightarrow [t_1, t_2, \dots, t_k, Q_2, \dots, Q_n = \epsilon$

$\Rightarrow \text{печатать } (\sigma)$

иначе

$\Rightarrow \text{вычислить_ответ } (\sigma [t_1, \dots, t_k, Q_2, \dots, Q_n])$

]

факт

Рис. 6.2. Вычисление ответа.

¹⁾ Разумеется, речь идет лишь о теоретически осуществимом алгоритме. Для эффективной реализации требуется гораздо более производительная процедура.

порядку и подряд, алгоритм выдает все возможные ответы (см. § 5.1.8). Ответ выдается инструкцией **печатать** (σ) при каждом получении пустой последовательности подцелей. Печатаются лишь значения переменных из вопроса или **да** (**нет**), если вопрос не содержит переменных. Важно заметить, что каждое рекурсивное обращение к *вычислить_ответ* просматривает БД каждый раз вновь с самого начала. После получения или неполучения подответа берется исходная последовательность для перехода к очередному выражению из БД. Именно это обстоятельство реализует *возврат*.

В этом алгоритме виден частный случай резолюции применительно к хорновским дизъюнктам (§§ 1.1.16, 1.2.14). Вопрос Q является правилом без заголовка, аналогом выражения $\neg Q$, т. е. $Q \supset \text{Л}$ ¹⁾. Пусть D — база данных (множество дизъюнктов). На вопрос Q ответим поиском такой подстановки σ , для которой множество $\sigma[D \cup (\neg Q)]$ невыполнимо. Стратегия выбора очередной пары дизъюнктов для резолюции здесь очень проста: подцели и выражения пробегаются в текстуальном порядке.

Вспомним БД семья (§ 6.1.7).

/ семья */*

мать (каролина, юлия).

мать (каролина, альбертина).

мать (мари, проспер).

мать (анна, каролина).

отец (проспер, юлия).

отец (проспер, альбертина).

отец (альфонс, проспер).

отец (андре, каролина).

дед (X, Y) :— отец (X, Z), мать (Z, Y).

дед (X, Y) :— отец (X, Z), отец (Z, Y).

бабка (X, Y) :— мать (X, Z), мать (Z, Y).

бабка (X, Y) :— мать (X, Z), отец (Z, Y).

¹⁾ Точнее, вопрос $? - Q$. является правилом: $- Q$, с телом Q и пустым заголовком (см. § 6.1.7). Это правило аналогично формуле $Q \supset \text{Л}$, т. е. $\neg Q$. — Прим. перев.

Зададим составной вопрос

$Q_1, Q_2 = \text{отец}(X, Y), \text{мать}(\text{каролина}, Y).$

Подцель

$Q_1 = \text{отец}(X, Y)$

соответствует пятому выражению (факту) БД

отец(проспер, юлия).

Это дает подстановку

$\sigma_1 = \{X = \text{проспер}, Y = \text{юлия}\}.$

Затем, используя рекурсивное обращение, алгоритм применяется к

$\sigma_1[Q_2] = \text{мать}(\text{каролина}, \text{юлия}).$

Этой подцели соответствует первое выражение БД, что дает пустую последовательность подцелей. Значит, ответ найден:

$X = \text{проспер}, Y = \text{юлия}.$

Для получения нового ответа в БД ищется новое соответствие для $\sigma_1[Q_2]$. Так как такового нет, то вычисления прекращаются на этом шаге рекурсии. Затем система вновь рассматривает последовательность Q_1, Q_2 и для Q_1 ищется новое соответствие в продолжении БД (начиная с шестого выражения). Это и есть «возврат». Шестое выражение сразу же дает желаемое соответствие — с подстановкой

$\sigma_2 = \{X = \text{проспер}, Y = \text{альбертина}\},$

и новое рекурсивное обращение, применяемое к

$\sigma_2[Q_2] = \text{мать}(\text{каролина}, \text{альбертина}),$

и т. д. В итоге имеем:

? — **отец(X, Y), мать($\text{каролина}, Y$).**

— — > **$X = \text{проспер}, Y = \text{юлия};$**

— — > **$X = \text{проспер}, Y = \text{альбертина};$**

— — > **нет**

Когда для заголовка правила соответствие установлено, то тело правила идет в заголовки неудовлетворенных подцелей. В вопросе

$Q_1, Q_2 = \text{дед}(U, V), \text{отец}(\text{проспер}, V)$

подцели Q_1 соответствует заголовок первого правила (девятого выражения) БД, что дает подстановку

$$\sigma_1 = \{X = U, Y = V\}$$

и новую последовательность подцелей

$Q' = \sigma_1 [\text{отец}(X, Z), \text{мать}(Z, Y), \text{отец}(\text{проспер}, V)],$

т. е.

$Q' = \text{отец}(U, Z), \text{мать}(Z, V), \text{отец}(\text{проспер}, V).$

Первому предикату из Q' соответствуют четыре факта, используемые один за другим. Так как первые три ничего не дают, то сразу переходим к четвертому

отец(андре, каролина).

дающему подстановку

$$\sigma_2 = \{U = \text{андре}, Z = \text{каролина}\}$$

и последовательность подцелей

$Q'' = \text{мать}(\text{каролина}, V), \text{отец}(\text{проспер}, V).$

Очередное соответствие с первым выражением дает подстановку

$$\sigma_3 = \{V = \text{юлия}\}$$

и последовательность

$Q''' = \text{отец}(\text{проспер}, \text{юлия}).$

которой соответствует один факт и тождественная подстановка $\sigma_4 = \{ \}$. Тем самым Q''' преобразуется в пустую последовательность. Значит, ответ таков:

U = андре, V = юлия.

В итоге имеем

?— дед(U, V), отец(проспер, V).

— — > U = андре, V = юлия;

— — > U = андре, V = альбертина;

— — > U = альфонс, V = юлия;

— — > U = альфонс, V = альбертина;

— — > нет

Интересно, что подцель последовательности $Q = Q_1, Q_2, \dots, Q_n$ может быть предикатом-значением вычисленной переменной. Например, правило

искать(P):— P .

использует переменную как предикат. Переменной надо присвоить значение-предикат при поиске соответствия в БД. Аналогично вопрос

?— **искать(отец(X, Y)).**

ставится в соответствие предикату **искать(отец(X, Y))**, проистекающему от вопроса с заголовком правила **искать(P)**, что дает подстановку

{ P = отец(X, Y)}.

Тело правила, т. е. значение предиката P , станет очередной подцелью, что дает ответ

X = проспер, Y = юлия.

Пример поинтереснее приведен в конце § 6.3.9.

6.2.4. Встроенные предикаты

У каждого языка программирования свой репертуар *встроенных* (готовых к применению) операций. Таковы, например, арифметические операции в классических языках. Пролог базируется на предикатах и поэтому встроенные операции в нем рассматриваются как предикаты, но это всего лишь номинально. Операция, называемая «предикатом», «функцией» или «инструкцией» (такая как ввод-вывод), имеющая краевой (остаточный) эффект (см. с. 368), не может

выполняться в точности так, как указано в процедуре *вычислить-ответ* (рис. 6.2). Такие операции называют «предикатами», потому что они рассматриваются как цели, которые надо достичь, что позволяет их считать элементами языка Пролог. Но ответы для них могут вычисляться без поиска соответствия в БД, т. е. их семантика в основном не зависит от БД.

Начнем со следующих встроенных предикатов:

```

true   fail
==  \==  =
atom  integer  number atomic var
repeat

```

Далее будут рассмотрены: *отсечение* (в § 6.2.5, важнейший из встроенных предикатов) и ввод-вывод (в § 6.3.7). Остальные будут вводиться по мере необходимости.

Предикаты **true** и **fail** (без аргументов) представляют значения истинности: первый достигается всегда, второй — никогда. Факт «**true**.» как бы присутствует в любой БД, «**fail**.» — ни в какой. Примеры покажут полезность **fail** при поиске всех соответствий для подцелей.

Три предиката со странными именами

```

==  \==  =

```

синтаксически — бинарные операторы (§§ 6.1.12, 6.3.10) сравнения аргументов-переменных **X** и **Y** без учета БД.

- **X == Y** достигается, если оба аргумента являются одним и тем же термом. Никакой подстановки не производится.

```

?- f(a) == f(a).

```

```

--> да

```

```

?- c == X.

```

```

--> нет

```

```

?- f(X) == f(X).

```

```

--> X = _0

```

Поясним последний ответ. Цель достигнута, ибо аргументы предиката равны. Система печатает значения переменных из вопроса и, если переменная не получила значения (не унифицирована), числовой код после подчеркивания.

● $X \mid== Y$ достигается, если не достигается
 $X == Y$

?— $f(a) \mid== f(a)$.

— — > нет

?— $c \mid== b$.

— — > да

?— $f(X) \mid== f(X)$.

— — > нет

● $X = Y$ играет более активную роль, пытаясь уравнивать свои аргументы унификацией. Он достигается вместе с унификацией и может повлечь подстановку.

?— $X = a$.

— — > $X = a$

?— $f(X, Y, Z) = f(X, U, a)$.

— — > $X = _0, Y = _1, Z = a, U = _1$

Следующие пять предикатов одноместные. Они анализируют свой аргумент, не запрашивая БД и не делая подстановок. Предназначены они для использования в телах правил, а не в вопросах.

● **atom**(X) достигается, если X — атомарная константа (нечисловой атом).

?— **atom** (полетта).

— — > да

?— **atom** (подруга (изабелла)).

— — > нет

?— **atom**(X),

— — > нет

atom(123).

— — > нет

● **integer**(X) достигается, если X — целый числовой атом.

?— **integer**(123).

— — > да

- ?— integer (1.23).
— — > нет
- number (X) достигается, если X — произвольный числовой атом.
- ?— number (123).
— — > да
- ?— number (1.23).
— — > да
- atomic (X) достигается, если X — произвольный атом.
- ?— atomic (полетта).
— — > да
- ?— atomic (подруга (изабелла)).
— — > нет
- ?— atomic (X).
— — > нет
- ?— atomic (123).
— — > да
- var (X) достигается, если X — не унифицируемая к терму переменная.
- ?— var (кадилл).
— — > нет
- ?— var (f (X)).
— — > нет
- ?— var (X).
— — > X = _0

Примеры использования предиката «= \Rightarrow » для унификаций:

- ?— X = a, var (X)
— — > нет
- ?— var (X), X = a, atom (X).
— — > X = a
- ?— X = Y, var (X), var (Y).
— — > X = _0, Y = _0
- ?— X = f (Y).
— — > X = f (_1), Y = _1
- ?— X = f (Y), var (X).
— — > нет

- Наконец **repeat** (без аргументов). Встроенность не препятствует определению его выражениями Пролога

repeat.

repeat:— repeat.

Достигается при каждом возврате к нему. Пусть имеется последовательность подцелей

$Q = \text{repeat}, Q_2, \dots, Q_n.$

Подцель **repeat** впервые достигается для факта «**repeat.**» из БД с подстановкой $\sigma = \{ \}$ и новой последовательностью $Q' = Q_2, \dots, Q_n$. Если Q' не дает ответа, то вновь пытаются (это возврат) удовлетворить Q другим выражением из БД, достигая подцель заголовка правила

repeat:— repeat.

Получается последовательность $Q'' = \text{repeat}, Q_2, \dots, Q_n$. Рекурсивный вызов *вычислить_ответ* (Q'') возобновит поиск с начала БД. Следовательно, подцель **repeat** вновь достигнута фактом «**repeat.**», и т. д. Возврат «отскакивает» от подцели **repeat**. Предикат возобновляет поиск соответствия для сопровождающих подцелей (см. пример в конце § 6.3.8).

6.2.5. Отсечение ¹⁾

Прологовский поиск (см. рис. 6.2) обычно выявляет все решения. Если БД велика, это обременительно. Подчас хватает одного решения. Встроенный предикат *отсечение* «срезает» поиск дальнейших соответствий. Его обозначают символом «!» (т. е. восклицательным знаком)²⁾. Предикат отсечение используется в тех случаях, когда:

- программист знает о единственности решения (прервав после его получения поиск, предикат «!» меняет эффективность, но не логику, программы);

¹⁾ Иное название — *усечение*. — Прим. перев.

²⁾ В некоторых версиях Пролога принято обозначение «/». — Прим. перев.

- пользователю хватает одного решения (предикат «!» исключит остальные влияя на логику программы),
- корректно лишь первое решение и нужно удалить другие решения (предикат «!» вовлечен в логику программы так, как это проиллюстрировано в приведенном ниже примере).

БД микро-дифф 1 представляет собой набросок программы¹⁾, осуществляющей символьное дифференцирование полиномов от одной переменной. В ней реализованы следующие правила дифференцирования:

$$dx/dx = 1, \quad dy/dx = 0, \quad d(u + v)/dx = du/dx + dv/dx.$$

/ микро_дифф1 */*

дифф(X, X, 1).

дифф(Y, X, 0):— atomic(Y).

*дифф(U + V, X, U1 + V1):— дифф(U, X, U1),
дифф(V, X, V1).*

Вспомним, что $U + V$ — прологовский терм, использующий синтаксис операторов (§ 6.1.12). Программе нужны аргументы для предиката **дифф**: первый должен представлять терм-полином, второй — атомарную константу (переменную дифференцирования), третий — прологовскую переменную, значением которой будет искомый результат. Если требуется получить все ответы, то диалог будет таким:

?— **дифф(y, x, R).**

— —> **R = 0;**

— —> **нет**

?— **дифф(x, x, R)**

— —> **R = 1;**

— —> **R = 0;**

— —> **нет**

Программа эта некорректна. Для ответа на второй вопрос найдено сперва соответствие с первым

¹⁾ В Прологе БД и программа часто являются синонимами. — *Прим. перев.*

выражением из БД и с подстановкой

$$\{X = x, R = 1\},$$

что приводит к подответу $R = 1$. Не остановившись, нашли еще одно соответствие с заголовком второго выражения из БД, с подстановкой

$$\{Y = x, X = x, R = 0\}$$

и предикатом $\text{atomic}(x)$, удовлетворение которого ищем. Осуществив достижение этого предиката, получаем очередной подответ $R = 0$. Несуразица устранится, если второе выражение из БД применять только лишь при различии его аргументов. Это условие реализовано в нижеследующей БД:

/ микро_дифф2 */*

$$\text{дифф}(X, X, 1).$$

$$\text{дифф}(Y, X, 0) :- Y \neq X, \text{atomic}(Y).$$

$$\text{дифф}(U + V, X, U1 + V1) :- \text{дифф}(U, X, U1),$$

$$\text{дифф}(V, X, V1).$$

Тот же эффект, т. е. те же самые вопросы-ответы, можно получить с помощью предиката «!». Это реализовано в следующей ниже БД:

/ микро_дифф3 */*

$$\text{дифф}(X, X, 1) :- !.$$

$$\text{дифф}(Y, X, 0) :- \text{atomic}(Y), !.$$

$$\text{дифф}(U + V, X, U1 + V1) :- \text{дифф}(U, X, U1),$$

$$\text{дифф}(V, X, V1).$$

Здесь первое отсечение «!» участвует в логике программы, а второе лишь сокращает поиск.

Предикат «!» как подцель всегда достигается (подобно `true`). Однако обладает он существенным краевым эффектом:

- Когда предикат «!» достигнут, то прекращается поиск соответствий для терма t , сопоставленного заголовку правила, которое содержит предикат «!» в своем теле.

Например, цель $t = \text{дифф}(x, x, R)$ соответствует заголовку первого выражения БД микро_дифф3

$$\text{дифф}(X, X, 1) :- !.$$

с подстановкой

$$\sigma = \{X = x, R = 1\}.$$

Целью становится тело правила, т. е. предикат «!», который достигается и дает пустую последовательность подцелей. Ответ (единственный, ибо прекращен поиск соответствий для t)

$$R = 1.$$

Важно заметить, что возможна унификация t с заголовком второго правила БД. Значит, важен порядок следования выражений. Переставив два первых выражения, получаем неверную программу. Используя «!» для избежания бесполезных поисков, на практике пишут сперва выражения для частных случаев, затем — для все более общих. Отсечение «!» ставят за предикатами, достижение которых выявляет решение (либо его отсутствие). Достижение цели **atomic(Y)** во втором выражении **микро_дифф3** означает, что Y — атом (в отличие от X , ибо **дифф(X, X, 1)** не достигается).

Пусть правило

$$t_0 :- t_1, t_2, \dots, t_{i-1}, !, t_{i+1}, \dots, t_k$$

(с предикатом «!») и последовательность подцелей

$$Q = Q_1, Q_2, \dots, Q_n$$

таковы, что t_0 и Q_1 соответствуют друг другу при некой подстановке σ . Получаем новую последовательность подцелей

$$t_1, t_2, \dots, t_{i-1}, !, t_{i+1}, \dots, t_k, Q_2, \dots, Q_n.$$

Подстановки подразумеваем, но не пишем. Если первые $i - 1$ подцелей достигнуты, то приходим к последовательности

$$!, t_{i+1}, \dots, t_k, Q_2, \dots, Q_n.$$

Одноразовая достижимость предиката «!» проявится при попытке повторить достижение этого же отсечения для получения новых решений. Поиск соответствий прекратится для Q_1 и членов последовательности

t_1, t_2, \dots, t_{i-1} , какие бы подстановки не реализовывались на этой стадии. Если Q_1 — подцель из последовательности

$$P_1, P_2, \dots, P_j, Q_1, P_{j+2}, \dots,$$

то продолжением операций будет поиск нового соответствия для P_j . Возможно, что придем к Q_1 с новыми значениями переменных. Предикат «!» препятствует осуществлению возврата в предшествующей этому отсечению части правила, включая заголовок. Это есть, таким образом, разновидность точки невозврата: полученный для Q_1 подответ по предшествующей «!» части правила войдет в итоговый ответ. Для рассматриваемой подцели обнаружится либо единственный ответ, либо его отсутствие. Если имеет место возврат, то пропускается предикат, который был достигнут для заголовка правила, содержащего «!», и вновь исследуется предыдущая подцель. Вот небольшой пример (БД и последовательность вопросов-ответов), который может помочь понять этот механизм:

```
/* ! */
p1 (X, Y) :— p2 (X), !, p3 (Y).
p1 (X, Y) :— p4 (Y).
p2 (a).
p2 (b).
p3 (b).
p4 (a).
p5 (a).
p5 (b).
?— p1 (a, a).
   — —> нет
?— p5 (X), p1 (X, X).
   — —> X = b;
   — —> нет
```

Предикат первого вопроса

$$Q_1 = p1(a, a)$$

соответствует заголовку первого выражения БД, давая последовательность подцелей

$$p2(a), !, p3(a).$$

Последовательно достижимы $p2(a)$ и «!», но не будет соответствия для $p3(a)$. Вернуться бы назад для поиска других соответствий, да мешает «!»: неудача цели $p1(a, a)$, полная и окончательная.

В поисках ответа на второй вопрос исходим из последовательности

$$Q_2 = p5(X), p1(X, X).$$

Поиск соответствия $p5(X)$ с $p5(a)$ преобразует Q_2 в $Q_3 = p1(a, a)$.

Цель Q_3 (ср. с Q_1 выше) недостижима, что обрекает на неудачу цель $p1(X, X)$ с $X = a$. Сделав возврат в последовательность Q_2 для поиска нового соответствия к $p5(X)$, найдем таковое — $p5(b)$. Это дает цель

$$Q4 = p1(b, b).$$

Она достижима, ибо $p2(b)$ и $p3(b)$ есть в БД. Ответ $X = b$.

6.3. Инструментарий и пример

6.3.1. Введение

Этот раздел посвящен полезным на практике средствам программирования: вычислительным процедурам, спискам, вводу-выводу и т. д. Напомним, что прологовские функции и операторы представляются как предикаты с операндами-термами. В § 6.3.11 приводится пример программы поиска в пространстве решений.

6.3.2. Вычислительные процедуры

В Прологе есть обычные вычислительные (или, иначе, числовые) операции. Например,

$$+ \quad - \quad * \quad / \quad \text{mod} \quad \text{log} \quad \text{sin}$$

У них имена функциональных термов, иногда являющихся операторами. Подчеркнем, что оператор в Прологе есть имя функции, которое можно помещать

между аргументами (если их два), либо до или после аргумента, если аргумент один (§ 6.3.10). Оператор обладает *приоритетом*. Например, термы Пролога

юлия * эмиль, — альфонс,
юлия * эмиль—эрнест

эквивалентны соответственно термам

* (юлия, эмиль), — (альфонс),
— (* (юлия, эмиль), эрнест).

Это не арифметика, как видно из следующего примера:

/* опера */
+ (юлия, эмиль).
жозеф * анриетта + эмиль.
12 + 6.
x + 2 * 21.
log (3).
sin (2).

Приведем последовательность вопросов-ответов, удовлетворяющую обычным правилам Пролога, но никакого отношения не имеющую к числовым выкладкам:

?— X + эмиль.
— —> X = юлия;
— —> X = жозеф * анриетта;
— —> нет
?— 12 + 6.
— —> да
?— 12 + 5.
— —> нет
?— X + Y.
— —> X = юлия, Y = эмиль;
— —> X = жозеф * анриетта, Y = эмиль;
— —> X = 12, Y = 6;
— —> X = x, Y = 2 * 21;
— —> нет
?— log (X).
— —> X = _ 3

Числовые выкладки здесь, очевидно, отсутствуют.

Так как вычислительные средства необходимы даже в символьном программировании, то мы также обзаведемся ими. Нетрудно видеть, что обойтись только унификацией практически нельзя — даже если это теоретически возможно. Нужны встроенные операторы. Вычисления в Прологе имеют следующие характерные черты:

- существуют числовые атомы, отличающиеся от других атомов своим описанием (§ 6.1.2),
- *исполнимы* в числовом смысле некоторые операторы и функции, такие, как
— встроенные:

$+ \quad - \quad * \quad / \quad \log \quad \sin \dots$

— термы, образованные только лишь из исполнимых операторов и функций, числовые атомы и переменные со значениями, являющимися исполнимыми термами¹⁾,

- существует встроенный бинарный оператор *is*, который преобразует исполнимый терм в числовой атом — результат вычисления (наподобие нахождения числового значения арифметического выражения) и унифицирует результат с другим термом (обычно с переменной).

Запись

X is T

где **X** — переменная и **T** — исполнимый терм, равносильна инструкции

X := T

из языка Паскаль.

Это подтверждается следующей последовательностью вопросов-ответов:

? — **X is 12 + 44.**

— — **> X = 56**

? — **X = 22, Y is 2 * X.**

— — **> X = 22, Y = 44**

¹⁾ Исполнимый терм — эквивалент арифметического выражения в языках наподобие Паскаля.

?— X is $2 + 3$, Y is $X + X$.
 — — $\rightarrow X = 5$, $Y = 10$
 ?— $X = 2 + 3$, Y is $X + X$.
 — — $\rightarrow X = 2 + 3$, $Y = 10$
 ?— X is $\log(2)$.
 — — $\rightarrow X = 0.693147$
 ?— X is $\log 10(2)$.
 — — $\rightarrow X = 0.30103$

Семантика этого встроенного оператора, рассматриваемого как бинарный оператор

$T1$ is $T2$

и как специфический предикат, определяется следующим образом:

- Правый операнд — исполнимый терм $T2$ — интерпретируется как арифметическое выражение. Выполняются определенные им числовые выкладки. Результат — числовой атом — унифицируется (если возможно) с левым операндом $T1$.
- Цель-предикат

$T1$ is $T2$

достигается только в тех случаях, когда $T1$
 — либо переменная, принимающая значение результата вычислений (обычный случай),
 — либо число, равное результату вычислений.

Например, диалог с использованием оператора is может быть таким:

?— 3 is $2 + 1$.
 — — \rightarrow да
 ?— X is $4 + 3$.
 — — $\rightarrow X = 7$
 ?— X is $4 + P$
 — — \rightarrow ! Ошибка в арифметическом выражении: не число
 ?— $f(X)$ is $3 + 8$.
 — — \rightarrow нет

?— $X = \log_{10} (2)$, Y is X .
 — — $\rightarrow X = \log_{10} (2)$, $Y = 0.30103$
 ?— $X = Y + \log_{10} (Z)$, $Y = 1$, $Z = 6 - 4$;
 R is X .
 — — $\rightarrow X = 1 + \log_{10} (6 - 4)$,
 $Y = 1$, $Z = 6 - 4$, $R = 1.30103$

Сообщение об ошибке здесь обусловлено либо неисполнимостью терма T_2 в данный момент, либо обработкой T_1 is T_2 как цели.

В вопросе после сообщения об ошибке T_1 нельзя унифицировать с числовым результатом. Предикат в таком случае считается не имеющим соответствия. Он не достигается в данном вопросе и, после возврата в текущей последовательности подцелей, возможно, будет отыскиваться другое соответствие.

Многие из обычных числовых функций и операторов исполнимы в определенном выше смысле. Детали надо уточнять в руководстве по Прологу конкретной системы.

Пролог может шокировать программистов, привыкших определять новые операции через объявления процедур: нет средств сделать исполнимыми новые функции (операторы). Пример ¹⁾

/ факториал */*
 $\text{fact}(0, 1) :- !.$

$\text{fact}(N, R) :- \text{integer}(N), N > 0, N1 \text{ is } N - 1,$
 $\text{fact}(N1, R1), R \text{ is } N * R1.$

Любая унификация терма $\text{fact}(X, Y)$ требует таковой для X — с целым неотрицательным числом и для Y — с переменной (факториалом от X). БД вычисляет факториал, но fact не «исполнимый». Это предикатное имя не может появляться в арифметических выражениях. В частности, для умножения $12!$ на 33 надо писать

$\text{fact}(12, A), R \text{ is } 33 * A.$

¹⁾ Разумеется, смысл символа $!$ ниже различен: в БД отсечение, а вне — факториал. — Прим. перев.

6.3.3. Списки

Списки — это последовательности символьных элементов, которые сами могут быть списками¹⁾. Эти структуры данных очень полезны и популярны в символьном программировании и в ИИ (обратите внимание на неизменный успех Лиспа). В Прологе, как и в Лиспе, для записи списков используют символьный атом `[]` пустого списка и *пары с точками* — бинарные функциональные термы (§ 6.1.2.). Точка — имя функции. Например, списками являются:

`[]` .(a,X) .(a,.(b,[]))

Рекурсивное определение:

- атом `[]` является списком и представляет пустой список (*нуль* в Лиспе),
- если `L` — список и `T` — произвольный терм, то `.(T,L)` — список с *головой* `T` и *хвостом* `L` (*car* и *cdr* в Лиспе)²⁾.

Например, список, состоящий из элементов `a`, `b` и `c`, есть терм вида

`.(a,.(b,.(c,[])))`

Однако для списков существует несколько лучший синтаксис:

- символический атом `[]` представляет пустой список,
- непустой список — это последовательность термов, разделенных запятыми и помещенных в квадратные скобки, причем термы любые, не обязательно атомы или списки.

Например,

`[a,b,c]` [*эта, маленькая (шапка), [для, кюре]*]

Непустой список допускает две эквивалентные записи — «с точками» и «в квадратных скобках». На-

¹⁾ См. также § 5.1.4. — *Прим. перев.*

²⁾ Можно напомнить синонимы: `T` — *заголовок*, `L` — *тело*. — *Прим. перев.*

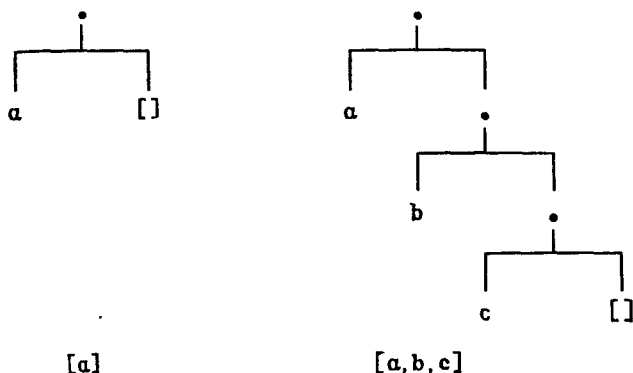


Рис. 6.3. Два списка.

пример,

$$[a] \equiv .(a, [])$$

$$[a,b,c] \equiv .(a, .(b, .(c, [])))$$

Представление деревьями — одно и то же для обоих обозначений (см. рис. 6.3). Знайки Лиспа здесь должны почувствовать себя в родной стихии. Теоретически это, быть может, «синтаксическое лакомство», но практически это весьма важно; таково следующее «лакомство»: обозначение

$$[T|Q]$$

представляет собой список с заголовком **T** и хвостом **Q**,

$$[A,B,C|Q]$$

представляет собой список с заголовком **A**, **B**, **C** и хвостом **Q** (§ 5.1.4). Эти обозначения, использованные для удовлетворения целей, годны для анализа списков:

$$?- [T|Q] = [a].$$

$$\text{---} > T = a, Q = []$$

$$?- [A1,A2, -, A4|Q] = [\text{прекрасная, маркиза, ваши, прекрасные, глаза}].$$

$$\text{---} > A1 = \text{прекрасная}, A2 = \text{маркиза},$$

$$A4 = \text{прекрасные}, Q = [\text{глаза}]^{1)}$$

¹⁾ Вспомним, что анонимные переменные (§ 6.1.6) не появляются в ответах. — *Прим. перев.*

6.3.4. Операции над списками

Ниже приводятся прологовские выражения (факты и правила) для операций над списками. В них аргументы для входных данных и результатов неразличимы. Такой теоретический подход иногда дает осечку на практике (см. вопросы-ответы).

Предикат `elem` достигается на парах (элемент, список):

`elem(X, [X|_]).`

`elem(X, [_|Y]) :- elem(X, Y).`

Использование этого предиката демонстрируется следующим диалогом:

?— `elem(X, [a, b, c]).`

— —> `X = a;`

— —> `X = b;`

— —> `X = c;`

— —> нет

При «нормальном» (т. е. отвечающем намерениям программиста) использовании предиката `elem` второй аргумент является списком или унифицируется с таковым. Вот пример «ненормального» использования этого предиката:

?— `elem(a, X).`

— —> `X = [a|_6];`

— —> `X = [_5, a|_10];`

— —> `X = [_5, _9, a|_14];`

Здесь предикат `elem(a, X)` интерпретируется следующим образом: `a` есть элемент списка `X`, если `a` является j -м элементом `X` при некотором j . Пролог строит последовательно списки, состоящие из первого элемента ($j = 1$), второго ($j = 2$) и т. д., а остальные элементы и хвост будут при этом неунифицированными переменными. Иногда это полезно, но чревато заикливанием: например, в последовательности

`elem(a, X), Q`

с недостижимой подцелью `Q` возврат бесконечно ищет новые решения для `elem(a, X)`.

Предикат **concat** достигается, если его третий аргумент есть конкатенация двух первых:

concat([], Z, Z).

concat([X|Y], Z, [X|R]) :— **concat**(Y, Z, R).

?— **concat**([a,b], [c,d], L).

— —> L = [a,b,c,d];

— —> нет

?— **concat**(X, Y, [a,b,c,d]).

— —> X = [], Y = [a,b,c,d];

— —> X = [a], Y = [b,c,d];

— —> X = [a,b], Y = [c,d];

— —> X = [a,b,c], Y = [d];

— —> X = [a,b,c,d], Y = [];

— —> нет

Функционирование **concat** «двусмысленно»: либо получаем единственный результат — конкатенацию двух списков, либо строится исчерпывающее перечисление таких пар списков, что конкатенация списков, входящих в пару, совпадает с третьим аргументом этого предиката. (Эта симметрия исчезнет, если применить отсечение.) Между тем не исключено заикливание, если первый и последний аргументы — переменные. Пример приводится далее.

Предикат **obr** достигается, если аргументы инверсны (взаимно перевернуты):

obr([], []).

obr([X|Y], R) :— **obr**(Y, Z), **concat**(Z, [X], R).

Это дает

?— **obr**([a,b,c,d], L).

— —> L = [d,c,b,a];

— —> нет

?— **obr**(L, [a,b,c,d]),

— —> L = [d,c,b,a];

[выполнение прервано]

Переворачивание идет в обоих направлениях, но симметрии нет — во втором случае происходит заикливание.

Предикат **mesto** достигается, если третий аргумент представляет собой список, содержащий все элементы

второго аргумента, расположенные в таком же порядке, в каком они располагаются во втором аргументе, а первый аргумент может находиться в произвольном месте:

$\text{mesto}(E, L, [E | L]).$

$\text{mesto}(E, [H | L], [H | Y]) :- \text{mesto}(E, L, Y).$

Это дает

?— $\text{mesto}(x, [a, b, c], L).$

— —> $L = [x, a, b, c];$

— —> $L = [a, x, b, c];$

— —> $L = [a, b, x, c];$

— —> $L = [a, b, c, x];$

— —> нет

?— $\text{mesto}(x, L, [a, x, b]).$

— —> $L = [a, b];$

— —> нет

?— $\text{mesto}(X, [a, b], [a, b, x]).$

— —> $X = x;$

— —> нет

Приведенные примеры проливают некоторый свет на кое-какие стороны программирования на Прологе. В этом языке все аргументы равноправны. В то же время для программиста обычно одни аргументы выступают в роли входных данных (причем эти аргументы унифицированы с термами-константами в момент поиска соответствия), в то время как другие аргументы представляют результаты и остаются переменными (не конкретизированными до нахождения соответствия). Пролог не обеспечивает автоматического контроля такого рода намерений программиста.

6.3.5. Перестановки и сортировки

Приведем примеры прологовских программ перестановок и сортировок списков. Это иллюстрация логического программирования на стандартных задачах без претензии на высокую эффективность. Намерения программиста предполагаются известными: первые аргументы — вход, последний — результат.

Предикат **perest** дает все перестановки первого аргумента:

perest([], []).

perest([H|L], Z):— **perest**(L, Y), **mesto**(H, Y, Z).

Индукцией по длине первого аргумента можно показать, что это дает нужный результат, если каждый поиск соответствия для предиката **mesto** помещает терм **H** на новое место в списке **Y** для получения **Z**. Последнее проверено выше.

?— **perest**([a, b, c], L).
 — — > L = [a, b, c];
 — — > L = [b, a, c];
 — — > L = [b, c, a];
 — — > L = [a, c, b];
 — — > L = [c, a, b];
 — — > L = [c, b, a];
 — — > нет

Для сортировки списка термов нужно задать отношение порядка. В Прологе их два. Первое — для термов-чисел. Оно дает 4 бинарных оператора с обычной числовой интерпретацией:

> < >= =<

Аргументы — исполнимые термы (§ 6.3.2).

Второе отношение — для символьных термов («следует за», «предшествует», «следует за или совпадает», «предшествует или совпадает»):

@> @< @>= @=<

со следующей интерпретацией этих бинарных операторов:

- переменные предшествуют числам и упорядочены системой,
- числа упорядочены естественным образом и предшествуют атомам,
- нечисловые атомы упорядочены по кодам **ASCII**¹⁾ (по крайней мере в Си-Прологе) и предшествуют функциям,

¹⁾ С добавлением кириллицы соответствует коду КОИ-7. — Прим. перев.

- функции упорядочены следующим образом:
 - сперва по числу аргументов: n -местная предшествует $(n + 1)$ -местной,
 - затем в алфавитном порядке имен функций,
 - наконец, в алфавитном порядке аргументов (текстуально упорядоченных).

Встроенный бинарный предикат `sort(X, Y)` унифицирует второй аргумент с сортировкой первого по `@`-возрастанию и убирает дубли:

```
?- sort([2,2,3,1], L).
   --> L = [1,2,3]
?- sort([ [a], [a,b], [a,b,c], f(x), f(a,b),
          f(a,X), X ], L).
   --> X = _31,
       L = [_31, f(x), [a], [a,b], [a,b,c],
            f(a, _31), f(a,b)].
```

Пролог обладает встроенной функцией сортировки. Примеры, приводимые ниже в учебных целях, реализуют сортировки вставкой, слиянием и быструю сортировку. Отношение порядка задается предикатом `pered`, например:

```
pered(A, B) :- A @ = < B.
```

(Ему дали имя, чтобы сортировки не зависели от выбора отношения порядка.)

Сортировка вставкой сортирует хвост списка и вставляет туда голову на подходящее место (пустой список считается отсортированным):

```
/* сортировка вставкой */
sor — vst([T|Q], L) :- sor_vst(Q, Z),
vstav(T, Z, L).
sor_vst([], []).
/* vstav(A, L, R) вставляет элемент A в упорядоченный
   список L, выдавая упорядоченный список R */
vstav(A, [T|Q], [A, T|Q]) :- pered(A, T), !.
vstav(A, [T|Q], [T|L]) :- vstav(A, Q, L).
vstav(A, [], [A]).
```

Сортировка слиянием делит список на два, сортирует оба полученных списка и осуществляет слияние

их в упорядоченный список (если список содержит менее чем два элемента, то он считается отсортированным):

```
/* сортировка слиянием */
sor_sli ([ ], [ ]).
sor_sli ([A], [A]).
sor_sli ([A, B | Q], R) :—
    delim(Q, Q1, Q2), sor_sli ([A | Q1], R1),
    sor_sli ([B | Q2], R2), sli(R1, R2, R).
delim ([ ], [ ], [ ]).
delim ([A], [A], [ ]).
delim ([A, B | Q], [A | Q1], [B | Q2]) :—
delim(Q, Q1, Q2).
/* sli(L1, L2, R) помещает в голову списка R
наибольшую из голов списков L1 и L2 */
sli ([ ], Y, Y).
sli(X, [ ], X).
sli([A | X], [B | Y], [A | Z]) :—
    pered(A, B), !, sli(X, [B | Y], Z).
sli([A | X], [B | Y], [B | Z]) :— sli([A | X], Y, Z).
```

Быстрая сортировка делит список на два так, что все элементы одного предшествуют всем элементам другого. Затем каждый подсписок сортируется и к полученным новым спискам применяется операция конкатенации:

```
/* быстрая сортировка */
/* sperva_men(E, L, L1, L2) помещает в L1 все
элементы из L, которые предшествуют E,
а остальные помещает в L2. Голова для L
выбирается как значение для E */
sperva_men(_, [ ], [ ], [ ]).
sperva_men(A, [B | X], [B | L1], L2) :—
    pered(B, A), !, sperva_men(A, X, L1, L2).
sperva_men(A, [B | X], L1, [B | L2]) :—
    sperva_men(A, X, L1, L2).
sor_bistr ([ ], [ ]).
sor_bistr ([A | X], R) :—
    sperva_men(A, X, L1, L2), sor_bistr(L1, M1),
    sor_bistr(L2, M2), concat(M1, [A | M2], R).
concat([ ], Z, Z).
concat([X | Y], Z, [X | R]) :— concat(Y, Z, R).
```

6.3.6. Представление списка в виде разности списков

Эффективные программы помогают получать следующий прием программирования представления списков (этот прием — сверх штатных средств Пролога):

- разность $L_1 - L_2$ списков L_1 и L_2 есть L_3 , если L_1 конкатенация L_3 с L_2 , иначе неопределенность.

Например,

$$[a, b, c, d] - [c, d] = [a, b].$$

Любой терм вида $L_1 - L_2$ интерпретируется как список. Всякий список представляется разностью: для списка $[a, b]$ возможны, в частности, представления $[a, b, c] - [c]$ и $[a, b, c, d] - [c, d]$. Существует наиболее общее представление, где вычитаемый список — неконкретизированная переменная: $[a, b] == [a, b | L] - L$. Такие термы можно записывать в выражениях Пролога, ибо минус есть оператор языка.

Назовем $[a, b | L] - L$ *d-списком*, представляющим $[a, b]$. Для пустого списка $[]$ *d-список* имеет вид $L - L$. Вообще, если $L - R$ представляет список l , то конкретизация R к $[]$ конкретизирует L к l . Конкатенация списков становится прямой и эффективной:

$$\text{dconcat}(L1 - R1, L2 - R2, L1 - R2) :- R1 = L2.$$

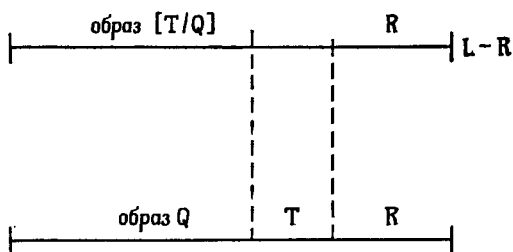
или короче:

$$\text{dconcat}(L1 - L2, L2 - R2, L1 - R2).$$

Ниже приведена эффективная версия предиката **obr** (§ 6.3.4), **obr 2**:

```
obr2(L, R) :- dobr(L, R - []).
dobr([], L - L).
dobr([T|Q], L - R) :- dobr(Q, L - [T|R]).
?- dobr([a, b], L - []).
--> L = [b, a]
?- obr2(L, [a, b]).
--> L = [b, a]
?- obr2(L, [b, a]).
--> L = [a, b]
```

Второй аргумент предиката **dobr** — d -список:



Эффективность возросла заметно. Для списков длины n сложность вычисления **obr** есть $O(n^2)$, а **dobr** — $O(n)$. Для первого вопроса, приведенного выше, вычисления таковы:

```
? —  dobr ([a, b], L — [ ]).
      dobr ([b], L — [a])
      dobr ([ ], L — [b, a])
      dobr ([ ], [b, a] — [b, a])
      — — > L = [b, a]
```

Использование неконкретизированных переменных в структурах данных — это общий метод Пролога. Его применяют не только для списков (соответствующий пример см. в § 6.4.4).

6.3.7. Ввод-вывод

В Прологе предикаты ввода-вывода немногочисленны и весьма рудиментарны. Есть пересылки терма за термом и символа за символом. Можно вставлять пробелы и переходы к новой строке. При посимвольных пересылках символ пересылается в виде целого числа, а именно кода **ASCII**.

Предусмотрен один стандартный канал ввода-вывода (терминал в интерактивном режиме). Можно явно указать другие каналы, обратившись к руководству пользователя для конкретной версии Пролога.

Особенность предикатов ввода-вывода как подцелей состоит в том, что они игнорируются при возвратах. Пусть Q , E , Q' — последовательность подцелей, где E — ввод (вывод). Достигнув E , перейдем к Q' .

При неудаче осуществляется возврат на Q с пропуском E . Это вполне естественно, так как бесполезно записывать одно и то же, а повторное чтение нецелесообразно.

Правильное использование этих предикатов перехода не всегда так просто, как может показаться, усвоить последовательность выполнения прологовской программы подчас нелегко. Возвраты могут привести к преждевременной печати ошибочных результатов.

Ниже приводятся некоторые самые ходовые предикаты ввода-вывода. Каналы не указаны: они предполагаются стандартными.

- **read(X)** читает терм, за которым стоит точка из входного канала и унифицирует его с X . Если X переменная, то ей присваивается значение согласно прочитанному терму. Цель **read(X)** достигается успехом унификации.

```
?— read(X).
   |: f(a).
   — —> X = f(a)
?— read(f(X)).
   |: f(p).
   — —> X = p
?— read(f(X)).
   |: [p].
   — —> нет
```

Система запросила ввод посредством символа $[:$. В последнем вопросе ввод не достигнут. (Напомним, что при достижении ввода как цели в ответе появляются значения переменных вопроса.)

- **get(X)** читает «печатаемый» символ и унифицирует его код **ASCII** (как целое число) с аргументом X . Достигается успехом унификации. В Си-Прологе символ является печатаемым, если его код **ASCII** больше 31. Непечатаемый символ игнорируется и берется следующий за ним.

```
?— get(X).
   |: a
   — —> X = 97
```

?— `get()`.

`[:]`

— —> `X = 93`

- `get 0 (X)` читает любой (печатаемый или нет) символ.

- `write(X)` печатает значение своего аргумента. Всегда достигим.

?— `X = 'Мадемуазель, хотите карамель?'`

`write(X)`.

Мадемуазель, хотите карамель?

— —> `X = Мадемуазель, хотите карамель?`

- `put(X)` печатает символ, код ASCII которого совпадает с `X`. Если значение `X` не число, то сообщается об ошибке. Если значение `X` число, но не код ASCII, то поведение Си-Пролога непредсказуемо.

? `put (97)`.

`a`

— —> да

- `nl` устанавливает новую строку. Без аргументов. Достигается всегда.
- `tab(N)` печатает `N` пробелов. Достигается, если значение `N` число, иначе появляется сообщение об ошибке.

г

6.3.8. Примеры ввода-вывода

Проследим эффекты возврата на правиле печати всех перестановок списка:

`vse_perestki(L) :- perestav(L, R), write(R),
nl, fail.`

Это правило напечатает все перестановки списка `L`. Предикат `perestav` (§ 6.3.5) порождает новую перестановку для каждого поиска соответствия. Ее печатает `write(R)`, `nl` осуществляет переход на новую строку, а `fail` никогда не достигается. В последовательности подцелей тела правила есть возврат. Он игнорирует вводы-выводы. Новое соответствие ищется

для `perestav(L, R)`, что создает новую перестановку. Для цели `vse_perestki(_)` имеет место) неудача, но все перестановки порождены и напечатаны.

?— `vse_perestki('прекрасная маркиза', 'ваши прекрасные глаза', 'сулят мне смерть от любви')`.

[прекрасная маркиза, ваши прекрасные глаза, сулят мне смерть от любви]

[ваши прекрасные глаза, прекрасная маркиза, сулят мне смерть от любви]

[ваши прекрасные глаза сулят мне смерть от любви, прекрасная маркиза]

[прекрасная маркиза, сулят мне смерть от любви ваши прекрасные глаза]

[сулят мне смерть от любви, прекрасная маркиза, ваши прекрасные глаза]

[сулят мне смерть от любви ваши прекрасные глаза, прекрасная маркиза]

— —> нет

Когда Пролог применяется на практике с текстовыми данными, фразы обычно представляются списками — как это продемонстрировано выше. Для каналов ввода-вывода предпочтителен обычный формат.

Пример вывода:

`печ_фразы ([] :— put(46). /* заключительная точка */`

`печ_фразы ([T| Q]) :— write(T); tab(1),`

`печ_фразы(Q).`

Зададим предикат считывания первого непробельного символа, игнорирующего пробелы между словами; 32 — это код пробела. Правило

`не_пробел (Car) :— get (Car), Car \= = 32.`

не годится, ибо, когда не достигается подцель

`Car \= = 32,`

то возврат пропустит `get(Car)`. Воспользуемся встроенным предикатом `repeat` (§ 6.2.4), достигающим всегда безотказно и неограниченное число

раз — как пружина:

не_пробел (Car) :— repeat, get (Car), Car \= = 32.

6.3.9. Анализ и построение термов

В Прологе функциональный терм или предикат можно рассматривать как структуру данных, подобную *записи* (по-английски *record*) в языке Паскаль. Ниже описываются встроенные предикаты Пролога, позволяющие осуществлять извлечение компонент из терма.

- Бинарный оператор $=..$ преобразует функциональный терм в список для доступа к имени функции и каждому аргументу:

$F = .. L$

достигается, если

— L есть список из $n + 1$ элементов

$L = [E_0, E_1, E_2, \dots, E_n],$

— F есть n -местный функциональный терм

$F = f(A_1, A_2, \dots, A_n),$

— f есть имя функции, соответствующее элементу E_0 ,

— каждый аргумент A_i соответствует элементу E_i .

Этот бинарный предикат обычно записывается в следующей форме:

$f(a, b, c) = .. L.$

т. е. слева расположен терм, справа — переменная. Имеем:

?— $f(a, b, c) = .. L.$

— — $\rightarrow L = [f, a, b, c]$

?— $f(a, b, c) = .. [F | Args].$

— — $\rightarrow F = f, Args = [a, b, c]$

?— $(a * b + c) = .. [Op | Args].$

— — $\rightarrow Op = +, Args = [a * b, c]$

Два следующих предиката анализируют термы.

- **functor** (F, T, N) достигается при соответствии T функциональному терму, F — имени этого термина и N — числу аргументов в терме.
 - **arg** (N, T, A) достигается при соответствии T функциональному терму, N — целому числу n и A — n -му аргументу в этом терме.
- ? — **functor** (имеет (жан, зонтик), F, N).
 — — > F = имеет, N = 2
- ? — **functor** (f (a, b), f, 2).
 — — > да
- ? — **arg** (2, имеет (жан, зонтик), A).
 — — > A = зонтик

Следующим предикатом расщепляется имя атома:

- **name** (N, L) достигается соответствием N атому, причем L является списком кодов ASCII символов имени.
- ? — **name** (bla bla, L).
 — — > L = [98, 108, 97, 98, 108, 97]
- ? — **name** (bac, [A | _]), name (X, [A]).
 — — > A = 98, X = b

Для иллюстрации присоединим к БД семья (§ 6.1.7) правило

все (A, F) :— **functor** (T, F, 2), T, **write** (A), **tab** (1),
write (F) **arg** (2, T, A2), **write** (', '),
write (A2), **arg** (1, T, A1) **write** (' — '),
write (A1), **nl**, **fail**.

Получим таким образом следующую последовательность вопросов-ответов.

- ? — **все** (твой, отец).
 твой отец, юлия — проспер
 твой отец, альбертина — проспер
 твой отец, проспер — альфонс
 твой отец, каролина — андре
 — — > нет
- ? — **все** (твоя, бабушка).
 твоя бабушка, юлия — анна
 твоя бабушка, альбертина — анна
 твоя бабушка, юлия — мари

твоя бабушка, альбертина — мари
 — — > нет

Ответы более удобочитаемы, чем термы Пролога. Полнота поиска обеспечивается предикатом **fail**. При нормальном использовании предиката **все (А, F)** аргумент **F** является именем предиката из БД, а аргумент **A** — притяжательным местоимением подходящего рода. Вникнем в детали. Цель —

все (твой, отец).

Тело правила выполняется с подстановкой

{A = твой, F = отец}.

Поиск соответствия для **functor(T, F, 2)** сопоставляет аргументу **T** значение, совпадающее с бинарным термом *отец*:

T = отец(—, —).

Это очередная подцель, соответствующая первому факту из БД:

отец (проспер, юлия).

Получаем

T = отец (проспер, юлия).

Последовательность подцелей тела правила анализирует терм и печатает его элементы в виде фразы. Последняя подцель **fail** обеспечивает исчерпывающий поиск соответствий:

6.3.10. Объявление оператора

Прологовские *операторы* (как мы видели в § 6.1.12) суть имена функций и/или предикатов (унарных и/или бинарных), записанные до, после или между аргументами. Им свойственны приоритет и ассоциативность:

? — **a + b * c = X + Y.**
 — — > **X = a, Y = b * c**
 ? — **a + b + c = X + Y.**
 — — > **X = a + b, Y = c**
 ? — **- a + b + c = X + Y.**
 — — > **X = - a + b, Y = c**

Атрибутика оператора:

- *предшествование*, выраженное натуральным числом: чем больше предшествование, тем меньше приоритет,
- *позиция* — инфиксная, префиксная или постфиксная,
- *ассоциативность*.

Предшествование выражают числом, а позицию и ассоциативность — одним из следующих способов:

- префиксность оператора (точнее, префиксная запись оператора): fx или fy ,
- постфиксность оператора: xf или yf ,
- инфиксность оператора: xfx , xfy , yfx или yfy .

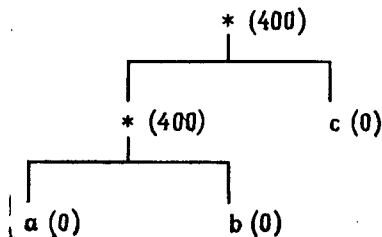
Здесь f — оператор, x и y — операнды:

- операнд y означает, что «предшествование этого операнда не больше, чем предшествование оператора»,
- операнд x означает, что «предшествование этого операнда строго меньше, чем предшествование оператора».

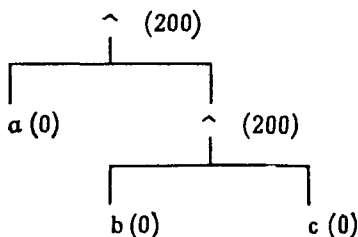
«Предшествование терма» полагается равным предшествованию его оператора, если таковой есть, в противном случае предшествование терма равно нулю. В частности, предшествование у термов $a + b * c$ и $a * b + c$ совпадает с предшествованием оператора $+$, т. е. равно 500.

Примеры типов встроенных операторов:

- оператор $*$ имеет тип yfx и предшествование 400, следовательно, $a * b * c$ имеет вид



- оператор $\hat{}$ имеет тип $\mathbf{x}\mathbf{f}\mathbf{y}$ и предшествование 200, следовательно, $\hat{a}\hat{b}\hat{c}$ имеет вид ¹⁾



- оператор $=$ имеет тип $\mathbf{x}\mathbf{f}\mathbf{x}$, следовательно, $a = b = c$ некорректно,
- оператор **not** имеет тип $\mathbf{f}\mathbf{y}$, следовательно, **not not t** корректно,
- оператор $-$ (унарный) имеет тип $\mathbf{f}\mathbf{x}$, следовательно $- - t$ некорректно.

Пролог может объявлять операторы префиксными, инфиксными или постфиксными с помощью встроенного предиката. Например, цель

op (500, yfx, союз)

всегда достижима с перманентным эффектом:

- союз становится бинарным оператором с предшествованием 500 и типом $\mathbf{y}\mathbf{f}\mathbf{x}$.

Рассмотрим пример.

- ? — $X = \text{союз}(a, b), \text{write}(X), \text{nl}, \text{op}(500, \text{yfx}, \text{союз}), \text{write}(X).$
 союз(a, b)
 a союз b
 — — > $X = a \text{ союз } b$
- ? — $Y = \text{fact}(a), \text{write}(Y), \text{nl}, \text{op}(700, \text{xf}, \text{fact}), \text{write}(Y).$
 fact(a)
 a fact
 — — > $Y = a \text{ fact}$

¹⁾ Здесь идет речь о возведении в степень. — Прим. перев.

Имена функций **союз** и **fact** стали операторами: первый из них — бинарным, второй — унарным постфиксным. Последний раз напоминаем, что речь идет лишь о синтаксисе.

6.3.11. Поиск в пространстве решений

Группа, в которой поровну людоедов и миссионеров, переезжает реку на пароме. Требуется в каждый момент времени избежать преобладания людоедов над миссионерами на берегах и на пароме.

Пусть $2N$ — численность группы (N миссионеров и N людоедов), P — число мест на пароме. При $P \geq 2N$ решение тривиально. При $P < 2N$ нужна подходящая стратегия. Пишем программу на Прологе.

Назовем *ситуацией* положение до или после каждой (очередной) переправы. Игра состоит в переходе от начальной ситуации (все на левом берегу) к заключительной (все на правом), осуществляемом последовательностью *рейсов*.

Некоторые ситуации и рейсы неприемлемы. Пусть в одном месте людоедов C и M миссионеров. Должны выполняться условия: на каждом берегу

$$(M \geq C) \text{ либо } (M = 0),$$

а на пароме

$$((M \geq C) \text{ или } (M = 0)) \text{ и } (M + C > 0).$$

Берег может быть пустым, но паромом кто-то должен управлять.

Каждую ситуацию будем описывать двумя тройками чисел: числом паромов, числом людоедов и числом миссионеров на левом и правом берегах соответственно:

$$[[\text{Пл}, \text{Лл}, \text{Мл}], [\text{Пп}, \text{Лп}, \text{Мп}]].$$

Налицо избыточность: одна тройка вычисляется по другой с использованием условий

$$\begin{aligned} \text{Пл} + \text{Пп} &= 1, \quad \text{Лл} + \text{Лп} = C, \quad \text{Мл} + \text{Мп} = M, \\ \text{Пл}, \dots, \text{Мп} &\geq 0. \end{aligned}$$

Предпочтем избыточность вычислениям. Ищем переход от ситуации

$[[1, N, N], [0, 0, 0]]$

к ситуации

$[[0, 0, 0], [1, N, N]]$.

Задача сводится к поиску пути, не имеющему циклов в ориентированном графе, узлами которого являются допустимые ситуации, а дугами — допустимые рейсы. Подходящий путь ищем следующим образом. Запоминаем встретившиеся ситуации, составляя список состояний-троек левого берега *Сл*. Список выполненных рейсов *Рс* послужит для печати результата.

Хотим получить диалог наподобие следующего:

?—ответ(3,2).

1 людоед и 1 миссионер переправляются направо.

1 миссионер переправляется налево.

2 людоеда переправляются направо.

1 людоед переправляется налево.

2 миссионера переправляются направо.

1 людоед и 1 миссионер переправляются налево.

2 миссионера переправляются направо.

1 людоед переправляется налево.

2 людоеда переправляются направо.

1 миссионер переправляется налево.

1 людоед и 1 миссионер переправляются направо.

— —> да

Предикат ответ(*N*, *P*) определяется правилом

ответ(*N*, *P*):— поиск(*P*, $[[1, N, N], [0, 0, 0]]$,

$[[0, 0, 0], [1, N, N]]$, $[[0, 0, 0]]$, *Рс*), показ(*Рс*).

Здесь предикат *поиск* ищет путь, унифицируя список выполненных рейсов с переменной *Рс* (пятый аргумент). Предикат *показ* печатает результат в удобочитаемом виде. Первые четыре аргумента *поиска* — вход: вместимость паромы, начальная и заключительная ситуации, список состояний левого берега.

Предикат **поиск** определен рекурсивно:

поиск (**Р**, **До**, **После**, **Сл_н**, [**Рейс**]):—

рейс (**Р**, **До**, **После**, **Сл_н**, **Сл_к**, **Рейс**).

поиск (**Р**, **До**, **После**, **Сл_н**, [**Рейс** | **Рс**]):— **рейс** (**Р**, **До**, **Сит**, **Сл_н**, **Сл**, **Рейс**), **поиск** (**Р**, **Сит**, **После**, **Сл**, **Рс**).

Предикат **рейс**¹⁾ определяется предикатами: **езжайте** (вычисляет возможные рейсы), **нов** и **ок** (проверяют — приводит ли вычисленный рейс в новую допустимую ситуацию). Ниже — **Л1** и **П1** — тройки берегов до рейса, а **Л2** и **П2** — после. Выше:

Р — вместимость парома,

До — ситуация до очередного рейса,

После — последующая ситуация,

Сл_н — список встречавшихся к данному моменту состояний,

Сл_к — то же с прибавленным новым состоянием (левого берега),

Рейс — выполненный рейс.

рейс (**Р**, [**Л1**, **П1**], [**Л2**, **П2**], **Сл**, [**П2** | **Сл**], **Рейс**):—

езжайте (**Р**, **С**, **М**, [**Л1**, **П1**], [**Л2**, **П2**], **Рейс**),

нов (**П2**, **Сл**), **ок** (**Л2**), **ок** (**П2**).

Предикат **езжайте** ищет (допустимые или нет) рейсы и запоминает стратегию. Правило перехода направо:

езжайте (**Р**, **С**, **М**, [[**1**, **С л1**, **М л1**], [**0**, **С п1**, **М п1**]],
[[**0**, **С л2**, **М л2**], [**1**, **С п2**, **М п2**]],
[**С**, **М**, 'направо.']):—

min (**М л1**, **Р**, **РМ**), **сокр** (**РМ**, **М**), **Q is** **Р ÷ М**,

min (**Q**, **С л1**, **РС**), **сокр** (**РС**, **С**), **ок_пар** (**С**, **М**).

С л2 is **С л1 — С**, **М л2 is** **М л1 — М**,

С п2 is **С п1 + С**, **М п2 is** **М п1 + М**.

Для достижения максимума перевозимых миссионеров **М** сперва вычисляем **РМ** (максимум транспортных миссионеров) как функцию от **Р** и от **М л1** (число миссионеров на левом берегу). Предикат **сокр** присваивает **М** целые значения, последовательно убывающие от **РМ** до 0. Другими словами, если нельзя

¹⁾ Не следует путать с переменной **Рейс**. — *Прим. перев.*

переправить **РМ** миссионеров, то выясняем, можно ли переправить **РМ** — 1 миссионеров, и т. д.

Значение **Q** — число мест для людоедов. Ищется максимум **C**. Предикат **ок_пар(C, M)** проверяет допустимость рейса. Приведенные выше две последние строки правила вычисляют новую ситуацию.

Правило перехода налево:

езжайте $(P, C, M, [[0, C_{л1}, M_{л1}], [1, C_{п1}, M_{п1}]],$
 $[[1, C_{л2}, M_{л2}], [0, C_{п2}, M_{п2}]],$
 $[C, M, 'налево.']):-$
 $\min(C_{п1}, P, PC), \text{прир}(PC, C), Q \text{ is } P - C,$
 $\min(Q, M_{п1}, PM), \text{прир}(PM, M),$
 $\text{ок_пар}(C, M),$
 $C_{п2} \text{ is } C_{п1} - C, M_{п2} \text{ is } M_{п1} - M,$
 $C_{л2} \text{ is } C_{л1} + C, M_{л2} \text{ is } M_{л1} + M.$

Налево (т. е. во встречном направлении, ведь переправа слева направо) переправляется минимальное число пассажиров. Продолжение БД:

$\text{ок}([_,_,0]):-!$
 $\text{ок}([_,C,M]):-C \leq M.$
 $\text{ок_пар}(C,0):-!, C > 0.$
 $\text{ок_пар}(C,M):-C \leq M.$
 $\text{нов}([_,_]):-!.$
 $\text{нов}(A,[A|L]):-!, \text{fail}$
 $\text{нов}(A,[_|L]):-\text{нов}(A,L).$
 $\text{сокр}(N,N).$
 $\text{сокр}(N,K):-M \text{ is } N-1, M \geq 0, \text{сокр}(M,K).$
 $\text{прир}(N,K):-\text{сокр}(N,R), K \text{ is } N-R.$
 $\min(A,B,A):-B \geq A, !.$
 $\min(A,B,B).$
 $\text{показ}([_]).$
 $\text{показ}([[C,M,L] | T]):-\text{печ}(C,M), \text{write}(L), \text{nl},$
 $\text{показ}(T).$
 $\text{печ}(0,1):-!, \text{write}('1 \text{ миссионер переправляется').}$
 $\text{печ}(1,0):-!, \text{write}('1 \text{ людоед переправляется').}$
 $\text{печ}(0,N):-!, \text{write}(N), \text{write}('миссионера$
 переправляются').
 $\text{печ}(N,0):-!, \text{write}(N), \text{write}('людоеда переправ-$
 ляются').
 $\text{печ}(1,1):-!, \text{write}('1 \text{ людоед и } 1 \text{ миссионер}$

переправляются').

печ(1, N):—!, write ('1 людоед и'), write (N),
write ('миссионера переправляются').

печ(N, 1):—!, write (N), write ('людоеда и 1 мис-
сионер переправляются').

печ(C, M):— write (C), write('людоеда и'), write (M),
write ('миссионера переправляются').

все_от(N, P):— ответ(N, P), nl,
write ('Другое решение:'), nl, fail.

Последнее правило вызовет все ответы:

? — все_от(2,2).

1 людоед и 1 миссионер переправляются на-
право.

1 миссионер переправляется налево.

2 миссионера переправляются направо.

1 миссионер переправляется налево.

1 людоед и 1 миссионер переправляются на-
право.

Другое решение:

1 людоед и 1 миссионер переправляются на-
право.

1 миссионер переправляется налево.

2 миссионера переправляются направо.

1 людоед переправляется налево.

2 людоеда переправляются направо.

Другое решение:

2 людоеда переправляются направо.

1 людоед переправляется налево.

2 миссионера переправляются направо.

1 миссионер переправляется налево.

1 людоед и 1 миссионер переправляются на-
право.

Другое решение:

2 людоеда переправляются направо.

1 людоед переправляется налево.

2 миссионера переправляются направо.

1 людоед переправляется налево

2 людоеда переправляются направо.

Другое решение:

— —> нет

6.4. Пролог и КС-грамматики

6.4.1. Введение

В гл. 5 КС-грамматики связывались с логикой, а продукции — с хорновскими дизъюнктами логики предикатов (§ 5.1.4). Сейчас мы вновь вернемся к этой тематике и рассмотрим связь Пролога с КС-грамматиками. Причем встанем на точку зрения пользователя, желающего программировать алгоритмы анализа КС-языков, а возможно даже — алгоритмы обращения с такими языками.

6.4.2. Распознавание КС-фраз

Из приводимых ниже продукций некоторой КС-грамматики

Фраза → *Подлежащее*, *Сказуемое*, *Определение*

Подлежащее → *Местоимение*, *Существительное*

Определение → *Прилагательное*

Местоимение → «эта»

Существительное → «дама» | «машина» | «зебра»

Прилагательное → «голубой» | «прекрасной» | «изящной»

Сказуемое → «становится» | «является»

первая гласит: «фраза есть конкатенация подлежащего, сказуемого и определения», а последняя — что сказуемые суть «становится» и «является»¹⁾. Вторая продукция представляет собой хорновский дизъюнкт (§ 5.1.4): $(\text{Местоимение}(\mathbf{M}) \wedge \text{Существительное}(\mathbf{C}) \wedge \text{По} = \mathbf{M} \parallel \mathbf{C}) \supset \text{Подлежащее}(\text{По})$. Формула $\text{По} = \mathbf{M} \parallel \mathbf{C}$, где символ \parallel означает конкатенацию, задает текстуальный порядок (§ 5.1.4). Используя хорновские дизъюнкты, легко перевести КС-продукции в правила Пролога:

/* КС */

фраза (Ф): — подл (По), сказ (Ск), опр (О),

кк (По, Ск, Z), кк (Z, О, Ф).

¹⁾ Обозначение *Сказуемое* → «становится» | «является» — классическое сокращение двух правил: *Сказуемое* → «становится» и *Сказуемое* → «является».

подл (По):— мест (М), суц (С), кк (М, С, По).

опр (О):— прил (П).

мест ([эта]).

суц ([дама]).

суц ([машина]).

суц ([зебра]).

прил ([голубой]).

прил ([прекрасной]).

прил ([изящной]).

сказ ([становится]).

сказ ([является]).

кк ([], Z, Z).

кк ([X|Y], Z, [X|R]):— кк (Y, Z, R).

Применяя эту БД, имеем:

?— фраза ([эта, дама, является, голубой]).

— —> да

?— фраза ([эта, дама, является, белой]).

— —> нет

?— фраза (L).

— —> L = [эта, дама, становится, голубой];

— —> L = [эта, дама, становится,
прекрасной];

— —> L = [эта, дама, становится, изящной];

— —> L = [эта, дама, является, голубой];

— —> L = [эта, дама, является, прекрасной];

— —> L = [эта, дама, является, изящной];

— —> L = [эта, машина, становится,
голубой];

— —> L = [эта, машина, становится,
прекрасной];

— —> L = [эта, машина, становится,
изящной];

— —> L = [эта, машина, является, голубой];

— —> L = [эта, машина, является,
прекрасной];

— —> L = [эта, машина, является, изящной];

— —> L = [эта, зебра, становится, голубой];

— —> L = [эта, зебра, становится,
прекрасной];

— —> L = [эта, зебра, становится, изящной];

- — > $L = [\text{эта, зебра, является, голубой}];$
- — > $L = [\text{эта, зебра, является, прекрасной}];$
- — > $L = [\text{эта, зебра, является, изящной}];$
- — > нет

Эта программа написана так, что ее можно использовать и как генератор, и как акцептор описанного нами языка. Но так бывает не всегда. Чаще программы можно эффективно использовать только в качестве акцепторов.

На практике особенно желательно уметь строить анализаторы ¹⁾, ²⁾. Таковых немало, различной общности и эффективности. Наш акцептор неэффективен до чрезвычайности, зато он быстро пишется и легко понимается. Сохраняя эти достоинства, улучшим его, вводя бинарные предикаты вида

- подл(L, X) достигается, если L — список анализируемых слов с началом — «подлежащим» и концом — списком X .

Например:

- ? — подл($[\text{эта, зебра, едет, на, поезде}], X$).
- — > $X = [\text{едет, на, поезде}]$

Соответствующее правило, записанное на Прологе, выглядит так:

подл(L, X):— мест($L, L1$), сущ($L1, X$).

Заметим, что мы обходимся без явных конкатенаций предикатов с переменными (нетерминальными символами КС-грамматики). Действительно:

- ? — $L = [\text{эта, зебра, едет, на, поезде}],$
 мест($L, L1$), сущ($L1, X$).
- — > $L = [\text{эта, зебра, едет, на, поезде}],$
 $L1 = [\text{зебра, едет, на, поезде}],$
 $X = [\text{едет, на, поезде}]$

Для предикатов с терминальными символами нужна явная конкатенация. Продукция *существи-*

¹⁾ Анализатор — это акцептор, строящий синтаксические деревья воспринятых фраз.

²⁾ Иные названия — распознаватели, парсеры. — *Прим. перев.*

тельное \rightarrow «зебра» записывается как следующее правило:

$\text{сущ}(\text{L}, \text{X}) : - \text{кк}([\text{зебра}], \text{X}, \text{L}).$

Ранее рассматривался частный случай конкатенаций, где первый аргумент являлся списком из одного элемента. В общем случае предикат, реализующий построение списка по заголовку и хвосту, логически определяется фактом:

$\text{с}([\text{X} | \text{Y}], \text{X}, \text{Y}).$

Это часто используемый встроенный предикат (аналог функции `cons` Лиспа). Пример:

? — $\text{с}(\text{R}, \text{заголовок}, [\text{a}, \text{b}, \text{c}]).$
 — — $\rightarrow \text{R} = [\text{заголовок}, \text{a}, \text{b}, \text{c}]$
 ? — $\text{с}([\text{a}, \text{b}, \text{c}], \text{X}, \text{Y}).$
 — — $\rightarrow \text{X} = \text{a}, \text{Y} = [\text{b}, \text{c}].$

Получаем следующую БД:

```
/* КС0 */
фраза(L, X) : - подл(L, L1), сказ(L1, L2) опр(L2, X).
подл(L, X) : - мест(L, L1), сущ(L1, X).
опр(L, X) : - прил(L, X).
мест(L, X) : - с(L, эта, X).
сущ(L, X) : - с(L, дама, X).
сущ(L, X) : - с(L, машина, X).
сущ(L, X) : - с(L, зебра, X).
прил(L, X) : - с(L, голубой, X).
прил(L, X) : - с(L, прекрасной, X).
прил(L, X) : - с(L, изящной, X).
сказ(L, X) : - с(L, становится, X).
сказ(L, X) : - с(L, является, X).
```

Этот вид записи БД менее удобочитаем, чем предыдущий. Зато он имеет одно очень важное свойство: существует средство автоматического преобразования продукций КС-грамматик в утверждения-вопросы Пролога. Система делает это сама.

Системное средство Пролога *ad hoc* дает следующую БД, вполне эквивалентную предыдущей (мы писали КС0 и автоматически получили КС1):

/* КС1 */

фраза — —> подл, сказ, опр.

подл — —> мест, сущ.

опр — —> прил.

мест — —> [эта].

сущ — —> [дама].

сущ — —> [машина].

сущ — —> [зебра].

прил — —> [голубой].

прил — —> [прекрасной].

прил — —> [изящной].

сказ — —> [становится].

сказ — —> [является].

Терминальные символы помещены в квадратные скобки как списки. Нетерминальные записаны без скобок. Выражения Пролога похожи на продукции КС-грамматик. Для их корректного использования вспомним о *бинарности* содержащихся в них предикатов. Имеем:

? — фраза ([эта, дама, является, голубой], []),

— —> да

? — фраза (L, []).

— —> L = [эта, дама, становится, голубой];

— —> L = [эта, дама, становится,
прекрасной];

и. т. д.

На практике программист добавляет интерфейс. Например:

акцепт (L) :— фраза (L, []).

6.4.3. Анализ КС-фраз

Такие базы данных, как КС1, не очень полезны: это акцептор, но не анализатор. Чтобы иметь возможность запечатлеть след анализа — синтаксическое дерево, и т. д. (фактически, чтобы иметь возможность получать ответы, отличные от «да» и «нет»), продолжим программу (§ 5.1.9). Снабдим дополнительными аргументами предикаты, которые представляют

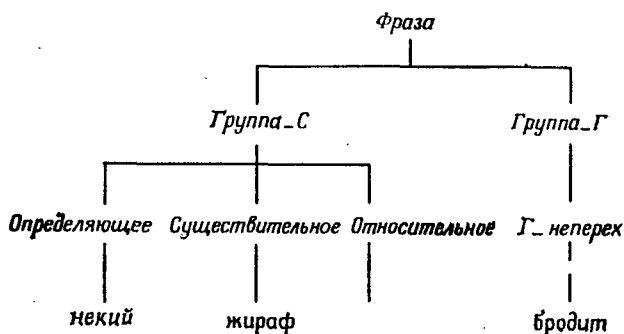


Рис. 6.4. Синтаксическое дерево.

нетерминальные символы данной грамматики. Рассмотрим следующую КС-грамматику:

<i>Фраза</i>	→ <i>Группа_С</i> , <i>Группа_Г</i>
<i>Группа_С</i>	→ <i>Определяющее</i> , <i>Существительное</i> , <i>Относительное</i>
<i>Группа_С</i>	→ <i>Имя_собственное</i>
<i>Группа_Г</i>	→ <i>Г_перех</i> , <i>Группа_С</i>
<i>Группа_Г</i>	→ <i>Г_неперех</i>
<i>Относительное</i>	→ "который", <i>Группа_Г</i> "которая", <i>Группа_Г</i>
<i>Относительное</i>	→ ε
<i>Определяющее</i>	→ "один" "некий" "каждая"
<i>Существительное</i>	→ "мужчина" "женщина" "жираф"
<i>Имя_собственное</i>	→ "жаклин" "эмиль" "адрианна"
<i>Г_перех</i>	→ "любит" "поддерживает"
<i>Г_неперех</i>	→ "бродит" "блаженствует"

Порожденный этой грамматикой КС-язык содержит фразы наподобие следующих: «некий жираф бродит», «адрианна поддерживает мужчину, который бродит», «некий жираф любит женщину» (впрочем, эту фразу следовало бы исключить). Займемся разработкой программы построения синтаксических деревьев (§ 5.1.9) для фраз этого КС-языка. На рис. 6.4. изображено синтаксическое дерево, представ-

ляющее следующий функциональный терм Пролога:

**фр (гс (опр (некий),
сущ (жираф),
отн ()),
гг (гн (бродит)))**).

Для построения дерева к каждому предикату, представляющему нетерминальный символ, добавляем аргумент. Например, первое правило рассматриваемой нами грамматики запишется так:

**фраза (фр (ГС, ГГ)) — — > группа_с (ГС),
группа_г (ГГ).**

Аргумент является термом. Если ГС и ГГ — термы-деревья группы существительного и группы глагола, то фр (ГС, ГГ) — дерево фразы.

Для записи предикатов с терминальными символами можно обойтись без переменных. Например, восьмое правило нашей грамматики можно записать так:

определяющее (опр (один)) — — > [один].

Предпочтем пользоваться переменными для лучшего учета синтаксических категорий (существительное, глагол, и т. д.):

**определяющее (опр (X)) — — > [X], явл_опр (X).
явл_опр (один).
явл_опр (некий).
явл_опр (каждая).**

Это не дает окончательного результата, так как написанные в таком виде правила Пролога содержат предикаты, имеющие два неявных аргумента. В действительности предикат **явл_опр (X)** имеет три аргумента, а надо бы только один. В качестве «лекарства» достаточно написать:

определяющее (опр (X)) — — > [X], {явл_опр (X)}.

Фигурные скобки указывают на отсутствие неявных аргументов. БД в примере на с. 406 представляет ОК-грамматику (§ 5.1.6). В эту БД включены правило перехода на новую строку и правило интерфейса. Предикат **вход (Ф)** в интерфейсе позволяет

использовать эту БД, не вводя явно трех аргументов предиката фразы.

/*АНАЛИЗ*/

фраза (фр (ГС, ГГ)) — — > группа_с (ГС),
группа_г (ГГ).

группа_с (гс (Оп, С, От)) — — > опр (Оп),
сущ (С), отн (От).

группа_с (гс (ИС)) — — > имя_собс (ИС).

группа_г (гг (Гп, Гс)) — — > г_перех (Гп),
группа_с (ГС).

группа_г (гг (Гн)) — — > г_неперех (Гн).

отн (от ('которая', ГГ)) — — > 'которая',

группа_г (ГГ).

отн (от (' ')) — — > [].

имя_собс (имс (X)) — — > [X], {явл_имс (X)}.

опр (оп (X)) — — > [X], {явл_опр (X)}.

сущ (с (X)) — — > [X], {явл_с (X)}.

г_перех (гп (X)) — — > [X], {явл_гп (X)}.

г_неперех (гн (X)) — — > [X], {явл_гн (X)}.

явл_имс (эмиль).

явл_имс (жаклин).

явл_имс (адрианна).

явл_опр (один).

явл_опр (некий).

явл_опр (каждая).

явл_с {мужчина}.

явл_с (женщина).

явл_с (жираф).

явл_гп (любит).

явл_гп (поддерживает).

явл_гн (бродит).

явл_гн (блаженствует).

/* интерфейс: правило с предикатом вход */

вход (Ф) :— фраза (Д, Ф, []), печ_дерево (Д).

печ_дерево (Д) :— печ_функ (Д, 0).

печ_функ (Ф, X) :— Ф = .. [E], write (E).

печ_функ (Ф, X) :— Ф = .. [L], печ_список (L, X).

неч_список (Ф | Args, N) :— write (Ф),

write (' '), K is N + 3, печ_хвост (Args, K).

печ_хвост ([Д], K) :— печ_функ (Д, K), write (' ').

печ_хвост ([Т | Q], K) :— печ_функ (Т, K),

write (' '), nl, tab (K), печ_хвост (Q, K).

Пример использования этой программы:

? — вход ([некий, жираф, бродит]).
 фр (гс (оп (некий),
 с (жираф),
 от ()),
 гг (гн (бродит)))
 — — > да

? — вход ([каждая, женщина, которая, любит,
 некий, жираф, который, бродит,
 блаженствует]). ¹⁾
 фр (гс (оп (каждая),
 с (женщина),
 от (которая,
 гг (гп (любит),
 гс (оп (некий),
 с (жираф),
 от (который,
 гг (гн (бродит))))))),
 гг (гн (блаженствует)))
 — — > да

6.4.4. ОК-грамматики и атрибутные грамматики

Грамматика базы данных анализ является атрибутной в смысле Кнута. Другими словами, аргумент в предикатах представляет собой синтезированный атрибут синтаксической структуры. Мы не будем заниматься условиями эквивалентности ОК-грамматик и атрибутных грамматик, а рассмотрим один классический пример из статьи [55], в которой впервые были представлены атрибуты. Этот пример проиллюстрирует *синтезированные* и *унаследованные атрибуты* в Прологе и послужит введением в *атрибутные системы*.

Речь пойдет о преобразовании двоичного числа (цепочки битов) в десятичное. КС-грамматика двоич-

¹⁾ По-русски эта фраза означает: «Каждая женщина, которая любит жирафа (который бродит), блаженствует». — *Прим. перев.*

ных чисел задается следующими правилами:

$$\begin{aligned} N &\rightarrow L \\ N &\rightarrow L_1 \text{ '}', L_2 \\ L &\rightarrow B \\ L &\rightarrow B, L \\ B &\rightarrow '0' \\ B &\rightarrow '1' \end{aligned}$$

Атрибутная система, вычисляющая десятичное значение цепочки битов по этой грамматике, имеет вид

$$\begin{aligned} N &\rightarrow L \\ &\quad val(N) := val(L) \\ &\quad pos(L) := 0 \\ N &\rightarrow L_1 \text{ '}', L_2 \\ &\quad val(N) := val(L_1) + val(L_2) \\ &\quad pos(L_1) := 0 \\ &\quad pos(L_2) := -long(L_2) \\ L &\rightarrow B \\ &\quad val(L) := val(B) \\ &\quad long(L) := 1 \\ &\quad ves(B) := pos(L) \\ L &\rightarrow B, L_1 \\ &\quad val(L) := val(B) + val(L_1) \\ &\quad long(L) := 1 + long(L_1) \\ &\quad ves(B) := pos(L) + long(L_1) \\ &\quad pos(L_1) := pos(L) \\ B &\rightarrow '0' \\ &\quad val(B) := 0 \\ B &\rightarrow '1' \\ &\quad val(B) := 2^P, \text{ где } P = ves(B) \end{aligned}$$

В этой системе атрибуты *val* (искомое десятичное значение) и *long* (длина цепочки битов) синтезированные, а *pos* (позиция данного бита относительно точки) и *ves* («вес» данного бита в составе результата) — унаследованные. Кроме того, *pos* зависит от *long*, *ves* — от *pos*, а *val* — от *pos* и *ves*. Следовательно, вычисление атрибутов потребует, вообще говоря, многих проходов по синтаксическому дереву.

В Прологе эти атрибуты станут аргументами (предикатов) **Val**, **Ves** и **Long**, а предикат **Pos** рассматри-

вать здесь ни к чему. Правило

$$L \rightarrow B$$

превратится в правило

$$\text{list}(\text{Val}, \text{Ves}, \text{Long}) \longrightarrow \text{bit}(\text{Val}, \text{Ves}), \{\text{Long}=1\}.$$

Синтезированные атрибуты **Val** и **Long** будут унифицированы с телом правила и «переданы» в переменные заголовка. Унаследованный атрибут **Ves** уже обладает неким значением после нахождения соответствия тела правила с БД.

Все это допустимо, ибо в переменную можно подставлять не только термы-константы. Терм при этом может содержать переменные. Например,

$$\text{Val} = \text{V1} + \text{V2}$$

унифицирует **Val**, если даже **V1** или **V2** — термы с неконкретизированными переменными. Это неверно для терма

$$\text{Val is V1} + \text{V2}$$

Действительно, его унификация требует, чтобы термы **V1** и **V2** были исполнимы (§ 6.3.2). Следовательно, они должны быть константами. В атрибутивной системе то же самое имеет место для классического присваивания

$$\text{val}(N) := \text{val}(L_1) + \text{val}(L_2).$$

Требуется предварительное присваивание значений всем переменным в правой части. (Иначе присваивание сработает «позже» — при одном из новых проходов по синтаксическому дереву.)

В прологовской БД КНУТО (которая приводится ниже) для вышеописанной атрибутивной системы унификации явные; но это, вообще говоря, является необязательным условием (см. БД КНУТ далее — в самом конце параграфа).

/* КНУТО */

$$\text{number}(\text{Val}) \longrightarrow \text{list}(\text{Val}, 0, _).$$


```

number(Val) — —> list(V1,0,_),['.'],
list(V2,Ves,Long),{Ves = - Long, Val = V1 + V2}.
list(Val,Ves,Long) — —> bit(Val,Ves),
                                     {Long = 1}.
list(Val,Ves,Long) — —> bit(V1,P1),
                        list(V2,Ves,L1), {P1 = Ves + L1,
                        Val = V1 + V2, Long = L1 + 1}.
bit(0,_) — —> [0],{!}.
bit(Val,Ves) — —> [1], {!,Val = 2 ^ Ves}.
s(N):— number(Val,N,[ ]),write(Val),nl,X is Val,
                                     write(X).

```

Последнее выражение печатает результат сперва в форме исполнимого терма, а затем — в чисто числовом виде:

```

?- s([1,0 1]).
2^(0 + (1 + 1)) + (0 + 2^0)
5
— —> да
?- s([1,0,.',0,1,0]).
2^(0 + 1) + 0 + (0 + (2^(-(1 + 1 + 1) + 1) +
+ 0))
2.25
— —> да

```

У первоначальной цели $s([1, 0, 1])$ первая подцель $\text{number(Val, [1, 0, 1], [])$.

Она требует синтаксического анализа списка битов при унификации переменной **Val** с термом

$$2^{(0 + (1 + 1)) + (0 + 2^0)},$$

после чего терм печатается. Затем он выполняется для присваивания **X** значения 5.

Если в БД заменить $=$ на is , то получим сообщения об ошибках. Действительно, исполнимый терм будет получен лишь по завершении анализа входного списка. Многократные проходы по синтаксическому дереву Пролог заменяет последовательностью под-унификаций. Усовершенствование (обещанное) БД

выглядит следующим образом:

```

/* КНУТ */
number (Val) — — > list (Val, 0, _).
number (V1 + V2) — — > list (V1, 0, _), ['.'],
                                list (V2, — Long, Long).
list (Val, Ves, 1) — — > bit (Val, Ves).
list (V1 + V2, Ves, Long + 1) — — > bit (V1,
                                Ves + Long), list (V2, Ves, Long).
bit (0, _) — — > [0], {!}.
bit (2^Ves, Ves) — — > [1], {!}.
s (N) :— number (Val, N, [ ]), write (Val),
                                nl, X is Val, write (X).

```

Литература

1. Abadi M. and Manna Z. Modal theorem proving. In: Proc. 8th Int. Conf. on Automated Deduction (J. H. Siekmann, ed.), LNCS 230, Springer Verlag, Berlin, pp. 172—189, juillet 1986.
2. Ait-Kaci H. and Nasr R. LOGIN: A logic programming with built-in inheritance, J. Logic Programming, vol. 3, no. 3, pp. 185—215, octobre 1986.
3. Barwise J. Handbook of Mathematical Logic, North-Holland, Amsterdam, 1974. Имеется перевод: Барвайс Дж. (ред.) Справочная книга по математической логике. — М.: Наука, 1982 (1—3), 1983 (4).
4. Bates M. The theory and practice of augmented transition network grammars. In: Natural Language Communication With Computers (L. Bolc, ed.), pp. 191—259, Lecture Notes in Computer Science, Springer-Verlag, Berlin, vol. 63, 1978.
5. Bell J. L. and Machover M. A Course in Mathematical Logic, North Holland, Amsterdam, 1977.
6. Besnard P., Quiniou R. and Quinton P. A theorem prover for a decidable subset of default logic, Proc. AAAI-83, pp. 27—30, 1983.
7. Besnard P. and Siegel P. The preferential models approach to non monotonic logics In: Non-Standard Logics for Automated Reasoning (P. Smets et al., eds.), Academic Press, London, 1988, p. 137—161.
8. Bobrow D. and Winograd T. An overview of KRL, a knowledge representation language, Cognitive Science, vol. 1, pp. 3—46, 1977.
9. Brachman R. J., Fikes R. E. and Levesque H. J. KRYPTON: Integrating terminology and assertion. Proc. AAAI-83, pp. 31—35, 1983.
10. Brachman R. J. and Levesque H. J. (eds.). Readings in Knowledge Representation, Morgan Kaufmann, Los Altos, 1985.
11. Bratko I. Prolog Programming for Artificial Intelligence, Addison Wesley, Reading, M. A. 1986. Имеется перевод:

- Братко И. Программирование на языке Пролог для искусственного интеллекта. — М.: Мир, 1990.
12. Ceccato S. *Linguistic Analysis and Programming for Mechanical Translation*, Gordon and Breach, New York, 1961.
 13. Chang C. C. and Keisler H. J. *Model Theory*, North Holland, Amsterdam, 1973. Имеется перевод: Кейслер Г., Чен Ч. Теория моделей. — М.: Мир, 1977.
 14. Chang C. L. and Lee R. C. T. *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973. Имеется перевод: Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. — М.: Наука, 1983.
 15. Chellas B. F. *Modal Logic: an Introduction*, Cambridge University Press, Cambridge, 1980.
 16. Chomsky N. On certain formal properties of grammars, *Information and Control*, vol. 2, no. 2, pp. 137—167, 1959. Имеется перевод: Хомский Н. О некоторых формальных свойствах грамматик. — В кн.: Киберн. сб., вып. 5, — М.: ИЛ, 1962, с. 279—311.
 17. Clark K. L. Negation as Failure: In: *Logic and Data Bases* (N. Gallaire et J. Minker, eds.), Plenum Press, New York, pp. 293—322, 1978.
 18. Clocksin W. F. and Mellish C. S. *Programming in Prolog* (2-ème éd.), Springer-Verlag, Berlin, 1984. Имеется перевод: Клоксин У., Меллиш К. Программирование на языке Пролог. — М.: Мир, 1987.
 19. Colmerauer A., Kanoui H., Roussel P. et Pasero R. *Un Système de Communication Homme-Machine en Français*, Groupe de Recherche et Intelligence Artificielle, Université d'Aix-Marseille, 1973.
 20. Colmerauer A. Metamorphosis grammars, In: *Natural Language Communication with Computers* (L. Bolc, ed.), pp. 139—169, *Lecture Notes in Computer Science*, vol. 63, Springer-Verlag, Berlin, 1978.
 21. Colmerauer A., Kanoui H. et Van Caneghem M. *Prolog, bases théoriques et développements actuels*, T. S. I., vol. 2, no. 4, 1983. Имеется перевод: Колмероз А., Кануи А., ван Канегем М. Пролог — теоретические основы и современное развитие. — В сб.: Логическое программирование. — М.: Мир, 1988, с. 27—133.
 22. Davio M., Deschamps J. P., Thayse A. *Discrete and Switching Functions*, McGraw-Hill, New York, 1978.
 23. Davis M. *Computability and Unsolvability*, McGraw Hill, New York, 1958.
 24. Dijkstra E. W. *A Discipline of Programming*, Academic Press, New York, 1976. Имеется перевод: Дейкстра Э. Дисциплина программирования. — М.: Мир, 1978.
 25. Dowling W. and Gallier J. H. Linear-time algorithms for testing the satisfiability of propositional Horn formulae, *J. of Logic Programmig*, vol. 1, 1984.
 26. Doyle J. A truth maintenance system, *Artificial Intelligence*, vol. 12, no. 3, pp. 231—272, 1979. Имеется перевод:

- Дойл Дж. Система поддержания истинности. — В кн.: Киберн. сб., вып. 20, — М.: Мир, 1983, с. 159—215.
27. Etherington D. W. and Reiter R. On inheritance hierarchies with exceptions, Proc. AAAI-83, pp. 104—108, 1983.
 28. Etherington D. W., Mercer R. E. and Reiter R. On the adequacy of predicate circumscription for closed-world reasoning, Proc. AAAI-Workshop on Non-Monotonic Reasoning, New Paltz, New York, pp. 70—81, octobre 1984.
 29. Etherington D. W. Formalizing nonmonotonic systems, Artificial Intelligence, vol. 31, no. 1, pp. 41—85, janvier 1987.
 30. Even S., Itai A. and Shamir A. On the complexity of timetable and multicommodity flow problems, SIAM J. Comput., vol. 5, pp. 691—703, 1976.
 31. Fikes R. and Kehler T. The role of frame-based representation in reasoning, Comm. ACM, vol. 28, no. 9, pp. 904—920, septembre 1985.
 32. Froidevaux C. JSA hierarchies with exceptions, Proc. 5 ième Congrès AFCET Reconnaissance des Formes et Intelligence Artificielle, Grenoble, pp. 1127—1138, 1985.
 33. Froidevaux C. Taxonomic default theory, Proc. ECAI-86, pp. 123—129, Brighton, juillet 1986.
 34. Froidevaux C. and Kayser D. Inheritance in semantic networks and in default logic, In «Non-Standard Logics for Automated Reasoning» (P. Smets et al., eds.), Academic Press, London, 1988.
 35. Gabbay D. M. Theoretical foundations for non-monotonic reasoning in expert systems, Research Report 84/11, Department of Computing, Imperial College, London, 1984.
 36. Garey M. R. and Johnson D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979. Имеется перевод: Гэри М., Джонсон Д. С. Вычислительные машины и труднорешаемые задачи. — М.: Мир, 1982.
 37. Grégoire E. Raisonnement plausible: inférence non monotone et logiques autoépistémiques, Proc. Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Paris, pp. 376—379, juillet 1986.
 38. Grégoire E. A note on Moore's autoepistemic logic. In: «Non-Standard Logics for Automated Reasoning» (P. Smets et al., eds.), Academic Press, London, 1988, p. 132—133.
 39. Gries D. The Science of Programming, Springer-Verlag, Berlin, 1981. Имеется перевод: Грис Д. Наука программирования. — М.: Мир, 1984.
 40. Halpern J. Y. and Moses Y. Towards a theory of knowledge and ignorance: Preliminary report, Proc. AAAI-Workshop on Non-Monotonic Reasoning, New Paltz, New York, pp. 125—143, octobre 1984.
 41. Halpern J. Y. and Moses Y. A guide to the modal logics of knowledge and belief: Preliminary draft, Proc. IJCAI-85, pp. 480—490, 1985.
 42. Halpern J. Y. (ed.) Proc. Conf. on Theoretical Aspects of

- Reasoning about Knowledge, Morgan Kaufmann, Los Altos, 1986.
43. Hayes P. J. In defense of logic, Proc. IJCAI-77, pp. 428—433, 1977.
 44. Hayes P. J. The logic of frame. In *Frame Conceptions and Text Understanding* (D. Metzger, ed.), de Gruyter, Berlin, pp. 46—61, 1979.
 45. Heffer A. (ed.) *Non-classical logics for expert systems*, CC-AI, vol. 3, no. 1—2, 1986.
 46. Hintikka J. *Knowledge and Belief: an Introduction to the Logic of the Two Notions*, Cornell University Press, Ithaca, New York, 1962.
 47. Hintikka J. Impossible possible worlds vindicated, *J. Philosophical Logic*, vol. 4, pp. 475—484, 1975.
 48. Hobbs J. R. and Moore R. C. (eds.) *Formal Theories of the Commonsense World*, Ablex, Norwood, 1985.
 49. Hopcroft J. and Ullman J. *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.
 50. Hughes G. E. and Cresswell M. J. *An Introduction to Modal Logic*, Methuen, London, 1972.
 51. Israel D. J. What's wrong with non-monotonic logic? Proc. AAAI-80, pp. 99—101, 1980.
 52. Israel D. J. A short companion to the naive physics manifesto. In: *Formal Theories of the Commonsense World* (J. Hobbs et R. C. Moore, eds.), pp. 427—447, Ablex, Norwood, 1985.
 53. Kleene S. C. *Introduction to Metamathematics*, North-Holland, Amsterdam, 1952. Имеется перевод: Клини С. Введение в метаматематику. — М.: ИЛ, 1957.
 54. Kleene S. C. *Mathematical Logic*, Wiley, Chichester, 1967. Имеется перевод: Клини С. Математическая логика. — М.: Мир, 1973.
 55. Knuth D. Semantics of context-free languages, *J. Math. Syst. Theory*, vol. 2, pp. 127—146, 1968.
 56. Konolige K. A deduction model of belief and its logics, Department of Computer Science, Report STAN-CS-84-1022, Stanford, San Francisco, 1984.
 57. Konolige K. On the relation between default theories and autoepistemic logic, Proc. IJCAI-87, Morgan Kaufmann, pp. 394—401, 1987.
 58. Kowalski R. *Logic for Problem Solving*, North-Holland, New York, 1979.
 59. Kramosil I. A note on deduction rules with negative premises, Proc. IJCAI-75, pp. 53—56, 1975.
 60. Kripke S. A. Semantical considerations on modal logic. In: *Reference and Modality* (L. Linsky, ed.), Oxford University Press, London, pp. 63—72, 1971.
 61. Levesque H. J. The interaction with incomplete knowledge bases: a formal treatment, Proc. IJCAI-88, pp. 240—245, 1981.
 62. Levesque H. J. A logic of implicit and explicit belief, Proc. AAAI-84, pp. 198—202, 1984.

63. Lloyd J. W. Foundations of Logic Programming, Springer-Verlag, Berlin, 1984.
64. Lukasiewicz J. Many-valued systems of propositional logic. In: Polish Logic (S. McCall, ed.), Oxford University Press, Oxford, 1967.
65. Manna Z. Mathematical Theory of Computation, McGraw-Hill, New York, 1974. Имеется перевод гл. 5: Манна З. Теория неподвижной точки программ. — В кн.: Кибернетический сборник, вып. 15, — М.: Мир, 1978, с. 38—100.
66. Marchal B. Modal logic: a brief tutorial. In: Non-Standard Logics for Automated Reasoning (P. Smets et al., eds.), Academic Press, London, 1988.
67. McCarthy J. Epistemological problems of artificial intelligence, Proc. IJCAI-77, pp. 1038—1044, 1977.
68. McCarthy J. Circumscription: a form of non-monotonic reasoning, Artificial Intelligence, vol. 13, no. 1—2, pp. 27—39, 1980.
69. McCartny J. Applications of circumscription to formalizing commonsense knowledge, Artificial Intelligence, vol. 28, no. 1, pp. 89—121, 1986.
70. McDermott D. and Doyle J. Non-monotonic logic I, Artificial Intelligence, vol. 13, no. 1—2, pp. 41—72, 1980.
71. McDermott D. Non-monotonic logic II: non-monotonic modal theories, J. ACM, vol. 29, no. 1, pp. 34—57, 1982.
72. Mendelson E. Introduction to Mathematical Logic (2-ème ed.), Van Nostrand, Princeton, 1979. Имеется перевод первого издания: Мендельсон Э. Введение в математическую логику. — М.: Наука, 1976.
73. Minsky M. L. Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, NY, 1967. Имеется перевод: Минский М. Вычисления и автоматы. — М.: Мир, 1971.
74. Minsky M. L. A framework for representing knowledge. In: The Psychology of Computer Vision (P. Winston, ed.), McGraw-Hill, New York, pp. 34—57, 1974. Имеется перевод: Минский М. Структура для представления знания. — В сб. Психология машинного зрения. — М.: Мир, 1978, с. 249—338.
75. Moisil G. Essai sur les Logiques non Chrysippiennes, Editions de L'Académie de la République Socialiste de Roumanie, 1972.
76. Montague R. The proper treatment of quantification in ordinary English. In: Formal Philosophy: Selected Papers of Richard Montague (R. H. Thomason, ed.), Yale University Press, London, 1974, pp. 188—221.
77. Moore R. C. The rôle of logic in knowledge representation and commonsense reasoning, Proc. AAAI-82, pp. 428—433, 1982.
78. Moore R. C. Possible-world semantics for auto-epistemic logic, Proc. AAAI-Workshop on Non-Monotonic Reasoning, New Paltz, New York, pp. 344—354, octobre 1984.
79. Moore R. C. Semantical considerations on non-monotonic logic, Artificial Intelligence, vol. 25, no. 1, pp. 75—94, 1985.

80. Moore R. C. Autoepistemic logic. In: *Non-Standard Logics for Automated Reasoning* (P. Smets et al., eds.), Academic Press, London, 1988, p. 105—136.
81. Murray N. Completely non-clausal theorem proving, *Artificial Intelligence*, vol. 18, pp. 67—86, 1982.
82. Newell A. The knowledge level, *AJ Magazine*, vol. 2, no. 2, pp. 1—20, 1980.
83. Nilsson N. *Principles of Artificial Intelligence*, Tioga Publishing Company, Palo Alto, CA, 1980. Имеется перевод: Нильсон Н. Принципы искусственного интеллекта. — М.: Радио и связь, 1985.
84. Pereira F. and Warren D. Definite Clause grammar for language analysis — A survey of the formalism and a comparison with ATN, *Artificial Intelligence*, vol. 13, pp. 231—278, 1980.
85. Putnam H. The analytic and the synthetic. In: *Mind, Language, and Reality* (H. Putnam, ed.), Cambridge University Press, Cambridge, pp. 33—69, 1962.
86. Quine W. V. *Methods of Logic*, Henry Holt, New York, 1950.
87. Reiter R. On closed-world data base. In: *Logic and Data Bases* (H. Gallaire and J. Minker, eds.), Plenum Press, New York, pp. 55—76, 1978.
88. Reiter R. A logic for default reasoning, *Artificial Intelligence*, vol. 13, no. 1—2, pp. 81—131, 1980.
89. Reiter R. and Criscuolo G. On interacting defaults, *Proc. IJCAI-81*, pp. 270—276, 1981.
90. Reiter R. Circumscription implies predicate completion (sometimes), *Proc. AAAI-Workshop on Non-monotonic Reasoning*, New Paltz, New York, pp. 418—420, octobre 1984.
91. Rescher N. *Many-valued Logic*, McGraw-Hill, New York, 1969.
92. Rich E. *Artificial Intelligence*, McGraw-Hill, New York, 1983.
93. Robinson J. A. A machine-oriented logic based on the resolution principle *J. ACM*, vol. 12, no. 1, pp. 23—41, 1965. Имеется перевод: Робинсон Дж. А. Машинно-ориентированная логика, основанная на принципе резолюции. — В кн.: Кибернетич. сб., нов. сер., вып. 7 — М.: Мир, 1970, с. 194—218.
94. Robinson G. and Wos L. Paramodulation and theorem-proving in first-order theories with equality, *Machine Intelligence 4*, Edinburgh University Press, Edinburgh, pp. 135—150, 1969.
95. Robinson J. A. *Logic: Form and Function*, Edinburgh University Press, Edinburgh, 1979.
96. Rogers H. *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967. Имеется перевод: Роджерс Х. Теория рекурсивных функций и эффективная вычислимость. — М.: Мир, 1972.
97. Sandewall E. An approach to the frame problem and its implementation. In: *Machine Intelligence 7* (B. Meltzer and D. Michie, eds.), Wiley, New York, pp. 195—204, 1972.
98. Schank R. and Abelson R. *Scripts, Plans, Goals and Understanding*, Lawrence Erlbaum Associates, New York, 1977.
99. Schoenfield J. R. *Mathematical Logic*, Addison-Wesley, Read-

- ing, MA, 1967. Имеется перевод: Шенфилд Дж. Математическая логика. — М.: Наука, 1975.
100. Smets Ph., Mandani E. H., Dubois D. and Prade N. (eds.) Non-Standard Logics for Automated Reasoning, Academic Press, London, 1988.
101. Snyers D. and Thayse A. From Logic Design to Logic Programming, Lecture Notes in Computer Science, vol. 271, Springer-Verlag, Berlin, 1987.
102. Sowa J. Conceptual Structures, Addison-Wesley, Reading, MA, 1984.
103. Stalnaker R. A note on non-monotonic modal logic, Note non publiée, Dept. of Philosophy, Cornell University, Ithaca, New York, juin, 1980.
104. Stefik M. and Bobrow D. G. Object-oriented programming: themes and variations, AI Magazine, vol. 6, no. 4, pp. 40—61, 1986.
105. Sterling L. and Shapiro E. The Art of Prolog, MIT Press, Cambridge, MA, 1986. Имеется перевод: Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. — М.: Мир, 1989.
106. Thayse A. Meet and join derivatives and their use in switching theory, IEEE Trans. Computers, vol. C-27, pp. 713—720, 1978.
107. Thayse A. Implementation and transformation of algorithms based on automata, Philips J. of Research, vol. 35, pp. 190—216, 1980.
108. Thayse A. P. Functions and Boolean Matrix Factorization, Lecture Notes in Computer Science, vol. 175, Springer-Verlag, 1984.
109. Tison P. Generalisation of consensus theory and application to the minimization of Boolean functions, IEEE Trans. Electr. Comput., vol. EC-5, pp. 126—132, 1967.
110. Touretzky D. S. The Mathematics of Inheritance Systems, Research Notes in Artificial Intelligence, Pitman, London, 1986.
111. Turner R. Logics for Artificial Intelligence, Ellis-Horwood, Chichester, 1984.
112. Winograd T. Language as a Cognitive Process, Vol. 1: Syntax, Addison-Wesley, Reading, MA, 1983.
113. Winston P. and Horn B. LISP (2-ème ed.), Addison-Wesley, Reading, MA, 1984.
114. Woods W. Transition network grammars for natural language analysis, Comm. ACM, vol. 13, No. 10, pp. 591—606. Octobre 1970. Имеется перевод: Вудс В. А. Сетевые грамматики для анализа естественных языков. — В кн.: Киберн. сб., вып. 13, — М.: Мир, 1976, с 120—158.
115. Woods W. Cascaded ATN grammars, American J. Computational Linguistics, vol. 6, no. 1, 1980.

Предметный указатель

- Абстракция- λ** (abstraction- λ) 161
- Автомат** детерминированный (automate déterministe) 303
- конечный (automate fini) 128, 303
- недетерминированный (automate non déterministe) 303
- стековый (automate à pile) 305
- — детерминированный (automate à pile déterministe) 308
- Автоэпистемическая интерпретация** (interprétation autoépistémique) 255, 261
- **логики** (logique autoépistémique) 227, 253
- **модель** (modèle autoépistémique) 255, 261
- **теория** легальная (théorie autoépistémique légale) 255
- — **основанная** (théorie autoépistémique fondée) 257
- — **полная** (théorie autoépistémique complète) 255
- — **устойчивая** (théorie autoépistémique stable) 256
- Аксиома** (axiome) 113
- Аксиоматическая система** (système axiomatique) 92
- — **немонотонная** (système axiomatique non monotone) 247
- Алгебра булева** (algèbre de Boole) 29
- Алгоритм** (algorithme) 125
- **Девиса и Патнема** (algorithme de Davis et Putnam) 33
- **Куайна** (algorithme de Quine) 24, 83
- **редукции** (algorithme de réduction) 26
- **унификации** (algorithme d'unification) 357
- Алгоритмы Пролога** (algorithmes de Prolog) 351
- Алетическая логика** (logique aléthique) 152
- Антисимметричность** (antisymétrie) 28
- Ассоциативность** (associativité) 28
- Атом** (atome) 59
- Атрибут** (attribut) 198
- Ввод-вывод в Прологе** (entrée-sortie Prolog) 384
- Возврат** (retour en arrière) 299, 343
- Вопрос** (question) 293, 337, 353
- Временная логика** (logique temporelle) 153
- Вывод** (inférence) 103
- **немонотонный** (inférence non monotone) 248
- Выражение- λ** (expression- λ) 161
- **тело** (corps d'une expression- λ) 162
- Высказывание** (proposition) 11
- Гипотеза** (hypothèse) 21
- Грамматика атрибутивная** (grammaire attribuée) 407
- **контекстно-свободная** (grammaire hors-contexte) 49, 270, 305, 399
- **определенных дизъюнктов** (grammaire DCG) 278, 279, 283, 290, 330, 407
- **регулярная** (grammaire régulière) 305
- **структуры фразы** (grammaire à structure de phrase) 300
- **Хомского** (grammaire de Chomsky) 300
- **типа 0** (grammaire de Chomsky de type 0) 311

- — типа 1 (grammaire de *Chomsky* de type 1) 310
- — типа 2 (grammaire de *Chomsky* de type 2) 305
- — типа 3 (grammaire de *Chomsky* de type 3) 300
- Граф канонический (graphe canonique) 181
- концептуальный (graphe conceptuel) 169

- Дедукции метатеорема (métathéorème de la déduction) 96
- принцип (principe de déduction) 22
- Дедукция обратная (déduction inverse) 167
- прямая (déduction directe) 166
- Деонтическая логика (logique déontique) 152
- Дерево-И/ИЛИ (arbre ET/OU) 296
- семантическое (arbre sémantique) 23
- — полное (arbre sémantique complet) 23
- синтаксическое (arbre syntaxique) 274, 329
- Диаграмма переходов (diagramme de transition) 303
- состояний (diagramme d'état) 303
- Дизъюнкт (clause) 30, 72
- резольвентный (clause résolvente) 86
- хорновский (clause de *Horn*) 45, 49, 274, 345
- Дизъюнктивная нормальная форма (forme disjunctive normale) 33, 72
- Дистрибутивность (distributivité) 28
- Доказательство (démonstration) 92, 103
- посредством опровержения (preuve par réfutation) 163
- Дополнительность (complémentarité) 29

- Доступности отношение (accessibilité) 242

- Завершаемость (complétude) 40
- Заголовок (tête) 271
- Заключение (conclusion) 21
- импликации (conséquent) 18
- Законы *де Моргана* (lois de *Morgan*) 28
- Замены правило (règle d'échange) 97
- Замыкание (fermeture) 268
- положительное (fermeture positive) 268
- рефлексивное и транзитивное (fermeture réflexive et transitive) 272
- \forall (fermeture universelle) 69
- \exists (fermeture existentielle) 69
- Знаний представление (représentation de la connaissance) 141
- Значение (valeur) 198
- слота (valeur de la facette) 172

- Идемпотентность (idempotence) 28
- Иерархия (hiérarchie) 185
- типов (hiérarchie des types) 185
- *Хомского* (hiérarchie de *Chomsky*) 299
- Имя индивидуума (nom spécifique) 143
- предикатное (nom de prédicat) 143
- слота (nom de la facette) 147
- совокупности (nom générique) 143
- функциональное (nom de fonction) 143
- Интерпретация (interprétation) 16, 63

- высказываний (interprétation propositionnelle) 255
- область (domaine d'interprétation) 63
- частичная (interprétation partielle) 23
- *Эрбрана* (interprétation de Herbrand) 77
- Исчисление высказываний (calcul des propositions) 11, 12
- предикатов (calcul des prédicats) 54, 57

- Категория синтаксическая (catégorie syntaxique) 270
- Квантификация (quantification) 56
- область действия (portée d'une quantification) 61
- Квантор (quantificateur) 56
- общности (quantificateur universel) 56
- существования (quantificateur existentiel) 56
- Кластер схематический (amas schématique) 190
- Клаузальная форма (forme clausal) 76
- Ключ (clé) 147
- Коммутативность (commutativité) 28
- Компактности теорема (théorème de compacité) 54
- Композиционный метод (principe de compositionnalité) 150
- Конкретизация (instance) 67, 75, 143
- фундаментальная (instance fondamentale) 79
- Константа (constante) 55, 143
- индивидуальная (constante individuelle) 57
- предикатная (constante prédictive) 56, 57, 143
- функциональная (constante fonctionnelle) 57, 143
- Конфигурация (configuration) 302, 306, 311
- Концептуальное отношение (relation conceptuel) 172

- Конъюнкт (cube) 33, 72
- Конъюнктивная нормальная форма (forme conjonctive normale) 30, 72

- Легальность (légalité) 255
- Литера (littéral) 16, 72
- Логика веры (logique des croyances) 153
- возможного (logique du possible) 152
- высказываний (logique des propositions) 211
- немонотонная (logique non monotone) 220, 227, 245
- предикатов (logique des prédicats) 211
- трехзначная (logique ternaire) 158
- умолчаний (logique des défauts) 226, 227
- четырехзначная (logique quaternaire) 160
- Логическое программирование (programmation logique) 89, 266, 333

- Матрица (matrice) 68
- Машина Тьюринга (machine de Turing) 127, 311
- Множество рекурсивно перечислимое (ensemble récursivement énumérable) 137
- рекурсивное (ensemble récursif) 14, 138
- Модальная логика (logique modale) 239
- нормальная система (système modal normal) 239
- теорема (théorème modal) 250
- Модальный оператор (opérateur modal) 153
- Моделей теория (théorie des modèles) 122
- Модель (modèle) 16, 113, 243
- высказываний (modèle propositionnel) 255

- Моноид свободный (monoïde libre) 268
- Монотонность ограниченная (monotonie restreinte) 221
- Наследственное свойство (propriété d'héritage) 183
- Неклаузуальное правило резолюций (résolution non clausale) 44, 89
- — согласия (consensus non clausale) 89
- Область Эрбрана (domaine de Herbrand) 76
- Обоснование (justification) 229
- Общезначимость (valide) 244
- в модели (valide dans un modèle) 244
- в структуре (valide dans une structure) 244
- Общезначимые рассуждения (raisonnement valide) 217
- Объявление оператора (déclaration d'opérateur) 391
- Одновременная постановка (substitution uniforme) 12, 60, 97
- Оператор- λ (opérateur- λ) 160
- Операции над списками (opération sur les listes) 378
- Операция парасочетания (opération de correspondance) 200
- Остановки проблема (problème de l'arrêt) 139
- Отношение порядка (relation d'ordre) 28, 117
- Отрицание в Прологе (négation Prolog) 349
- Отсечение (coupure) 363, 366
- Оценка (valuation) 242
- Переменная (variable) 55, 57, 143, 270
- анонимная (variable anonyme) 344
- квантифицированная (variable quantifiée) 61
- начальная (variable de départ) 270
- свободная (variable libre) 60
- Подстановка (substitution) 66, 354
- Полугруппа свободная (demi-group libre) 268
- Правило (règle) 165, 268, 270, 293, 345
- немонотонного вывода (inférence non monotone) 223
- обобщения (règle de généralisation) 106
- переписывания (règle de réécriture) 271
- рекурсивное (règle récursive) 348
- соединения (règle d'assemblage) 143
- умолчания (règle de défaut) 229
- Modus Ponens (Modus Ponens) 94
- Предваренная форма (forme préfixe) 70
- Предикат (prédicat) 59, 143
- бинарный (predicat binaire) 144
- встроенный (prédicat prédéfini) 362
- унарный (prédicat unaire) 144
- Префикс (préfixe) 70
- Принцип двойственности (principe de dualité) 29, 65
- монотонности (principe de monotonie) 104
- Продукция (production) 268, 270
- Пролог (Prolog) 290
- Прототип (prototype) 187, 189, 191
- Процедура (procédure) 125
- Разрешимая теория (théorie décidable) 120
- Разрешимость (décidabilité) 123
- Рассуждения модифицируемые

- (raisonnement révisable) 217
 — с умолчаниями (raisonnement par défaut) 206, 228
 Расширение (extension) 232
 — устойчивое (extension stable) 257, 258, 262
 Резольвента (résolvante) 38
 Резолюций принцип (résolution) 38, 42, 87, 99
 Резолюция фундаментальная (résolution fondamentale) 84
 Рефлексивность (réflexivité) 28
 Решетка (treillis) 28
 — *Линденбаума* (treillis de *Lindenbaum*) 28
 — множеств (treillis des ensembles) 186
 — типов (treillis des types) 186
- Свойство унаследованное** (propriété héritée) 183
Связанная переменная (variable liée) 60
Связка (connecteur) 57
Семантика (sémantique) 15, 62, 150, 157, 162
 — **возможных миров** (sémantique des mondes possibles) 158, 242
Сеть базовая переходов (réseau BTN) 316
 — **рекурсивная переходов** (réseau RTN) 320
 — **семантическая** (réseau sémantique) 172
 — **усиленная переходов** (réseau ATN) 325
Сигнатура (signature) 113
Символ исходный (symbole initial) 270
 — **нетерминальный** (symbole non terminal) 270
Синтаксис (syntaxe) 15
Система натурального вывода (déduction naturelle) 102, 107
Сколемовская форма (forme de Skolem) 73
Следствие логическое (conséquence logique) 21
- Слот** (facette) 147, 198
Совокупности поле (champ générique) 176
Совокупность — **ссылка** (générique — référent) 175, 188
Согласие (consensus) 43, 89
Сортировка (tri) 380
Список (liste) 275, 376
Ссылки поле (champ référent) 176
Стратегия (stratégie) 298
 — *Девиса и Патнема* (stratégie de *Davis et Putnam*) 83
Структура (structure) 113, 242
Схема (schéma) 190
Сцепка (unité) 197
- Тавтология** (tautologie) 21
Тезис Чёрча (thèse de *Church*) 133
Тело (corps) 271
Теорема (théorème) 92, 113
Теория (théorie) 113
 — **категоричная** (théorie catégorique) 121
 — **нормальная** (théorie normale) 233
 — **первого порядка** (théorie du premier ordre) 111, 113
 — **полная** (théory complète) 121
 — **полунормальная** (théory semi-normale) 236
 — **с умолчаниями** (théorie avec défauts) 229
Терм (terme) 56, 58
 — **индивидуальный** (terme fondamental) 337
 — **константный** (terme constant) 337
Точка неподвижная (point fixe) 250
Транзитивность (transitivité) 28
Требование (prérequis) 229
- Узел** — **индивид** (noeud spécifique) 176

- концепт (noeud concept) 172
- связывающий (noeud relationnel) 172
- совокупность (noeud générique) 176
- ссылка (noeud référent) 176
- Умолчание (défaut) 229
- нормальное (défaut normal) 234
- Универсум (univers) 242
- Унификатор (unificateur) 355
- наиболее общий (unificateur le plus général) 86, 356
- Унификация (unification) 85
- Унифицируемые литеры (littéraux unifiables) 85
- Факт (fait) 46, 165, 201, 293, 337.
- Финитно выполнимое подмножество (sous-ensemble finiment consistant) 51
- Форма предикатная (forme prédicative) 56, 59, 144
- Формализм усиленных сетей переходов (formalisme ATN) 314
- Формула (formule) 13, 59
 - выполняемая (formule consistante) 20, 64
 - невыполнимая (formule inconsistent) 20, 64
 - нейтральная (formule contingente) 21, 64
- общезначимая (formule valide) 20, 64
- Фрейм (cadre) 197
- функциональный (cadre fonctionnel) 199
- явный (cadre explicite) 198
- Фундаментальная форма (forme fondamentale) 78
- Функциональная форма (forme fonctionnelle) 58
- Функция (fonction) 59, 149
 - примитивно рекурсивная (fonction primitive réursive) 136
 - рекурсивная (fonction réursive) 137
- Цель (but) 46, 165, 201, 293, 341, 353
- Элиминация (elimination) 28
- Эпистемическая логика (logique épistémique) 153
- Язык (language) 113
 - алгоритмический Гёделя (language algorithmique de Gödel) 131
 - контекстно-свободный (language hors-contexte) 272

Оглавление

Предисловие редактора перевода	5
Предисловие	7
1. Логика	11
1.1. Исчисление высказываний	11
1.1.1. Введение	11
1.1.2. Словарь	11
1.1.3. Синтаксис исчисления высказываний	12
1.1.4. Семантика исчисления высказываний	15
1.1.5. Исчисление высказываний и естественный язык	16
1.1.6. Выполнимые и общезначимые формулы	20
1.1.7. Алгоритмическая точка зрения	23
1.1.8. Алгоритм редукции	26
1.1.9. Алгебраический подход	27
1.1.10. Дизъюнкты и нормальные формы	30
1.1.11. Алгоритм Девиса и Патиема	33
1.1.12. Принцип резолюций	37
1.1.13. Доказательства невыполнимости, основанные на принципе резолюций	38
1.1.14. Приложения и примеры использования метода резолюций	42
1.1.15. Неклаузальное правило резолюций	44
1.1.16. Хорновские дизъюнкты	45
1.1.17. Хорновские дизъюнкты и КС-грамматики	49
1.1.18. Теорема компактности	51
1.2. Исчисление предикатов	54
1.2.1. Введение	54
1.2.2. Словарь	57
1.2.3. Синтаксис исчисления предикатов	58
1.2.4. Свободные и связанные переменные, область дей- ствия	59
1.2.5. Семантика исчисления предикатов	62
1.2.6. Подстановка и конкретизация	66
1.2.7. Предваренная и нормальные формы	69
1.2.8. Сколемовские и клаузальные формы	72
1.2.9. Эрбранова интерпретация и компактность	76
1.2.10. Два простых примера	80
1.2.11. Алгоритм Куайна, Девиса и Патиема	83
1.2.12. Фундаментальная резолюция	84
1.2.13. Унификация	85
1.2.14. Метод резолюций	87
1.2.15. Принцип логического программирования	89

2. Аксиоматические системы	92
2.1. Аксиоматический подход к логике	92
2.1.1. Введение	92
2.1.2. Свойства аксиоматических систем	93
2.1.3. Простая аксиоматическая система исчисления высказываний	94
2.1.4. Несколько интересных теорем	95
2.1.5. Полнота	98
2.1.6. Польза аксиоматических систем	100
2.1.7. Система натурального вывода	102
2.1.8. Классические аксиомы для квантификации	105
2.1.9. Натуральный вывод в логике предикатов	107
2.1.10. Равенство в исчислении предикатов	108
2.2. Теории первого порядка	111
2.2.1. Введение	111
2.2.2. Неформальные и формальные теории	112
2.2.3. Польза теорий	114
2.2.4. Теория частичного порядка	117
2.2.5. Модели теории	119
2.2.6. Алгоритмы и разрешимость	123
2.2.7. Алгоритмический язык Тьюринга	127
2.2.8. Алгоритмический язык Гёделя	131
2.2.9. Кодирование и тезис Чёрча	133
2.2.10. Класс вычислимых функций	136
2.2.11. Проблема остановки	139
3. Представление знаний и рассуждений	141
3.1. Логическое представление	141
3.1.1. Введение	141
3.1.2. Синтаксис логики предикатов	142
3.1.3. Примеры	144
3.1.4. Преобразование унарных предикатов в бинарные	144
3.1.5. Примеры	145
3.1.6. Преобразование m -арных предикатов в произведение бинарных	146
3.1.7. Явное представление ссылок	148
3.1.8. Представление функциями	149
3.1.9. Примеры	149
3.1.10. Семантика логики предикатов	150
3.1.11. Модальная логика предикатов	152
3.1.12. Модальные операторы	153
3.1.13. Примеры модальных операторов	154
3.1.14. Синтаксис модальной логики предикатов	155
3.1.15. Примеры	155
3.1.16. Трёхзначная семантика для модальной логики предикатов	157
3.1.17. Семантика возможных миров	158
3.1.18. Лямбда-исчисление	160
3.1.19. Рассуждения, использующие логические формулы	162

3.1.20. Пример	163
3.1.21. Рассуждения по поводу знаний	165
3.1.22. Системы прямой дедукции	166
3.1.23. Системы обратной дедукции	167
3.2. Сетевое представление	168
3.2.1. Введение	168
3.2.2. Концептуальные графы	169
3.2.3. Пример и терминология	170
3.2.4. Семантические сети	172
3.2.5. Правила конъюнкции и упрощения	173
3.2.6. Представление контекста	173
3.2.7. Представление «совокупность-ссылка»	175
3.2.8. Пример	177
3.2.9. Пример введения кванторов	177
3.2.10. Временные и модальные операторы	179
3.2.11. Канонические графы	181
3.2.12. Правила построения	182
3.2.13. Унаследованные свойства	183
3.2.14. Решетки типов; иерархии типов	185
3.2.15. Решетки множеств и решетки типов	186
3.2.16. Определение типа посредством рода и различия	188
3.2.17. Прототипы	189
3.2.18. Схемы и схематические кластеры	190
3.2.19. Рассуждения, использующие семантические сети	192
3.2.20. Заключение	194
3.3. Объектное представление	196
3.3.1. Введение	196
3.3.2. Сцепки	197
3.3.3. Фреймы и слоты	197
3.3.4. Явные фреймы	198
3.3.5. Функциональные фреймы	199
3.3.6. V -квантификация	199
3.3.7. Рассуждения, использующие объектное представление	200
3.3.8. Паросочетание	200
3.3.9. Функциональные атрибуты	202
3.3.10. Автоматические рассуждения, использующие фреймы	203
3.3.11. Иерархические рассуждения, использующие фреймы	204
3.3.12. Рассуждения с умолчаниями	206
3.3.13. Заключение	208
4. Логика и модифицируемые рассуждения	209
4.1. Многочисленные роли логики	209
4.1.1. Введение	209
4.1.2. Логика как средство для представления знаний и рассуждений	209
4.1.3. Логика как формализм ссылок	213
4.1.4. Необходимость логики	214
4.1.5. Анализ знаний и рассуждений	215
4.1.6. Заключение	216

4.2. Логика и модифицируемые рассуждения	217
4.2.1. Формализация модифицируемых рассуждений	217
4.2.2. Классическая логика и общезначимые рассуждения	217
4.2.3. Характеристики немонотонных логик	220
4.2.4. Заикливание правил немонотонного вывода	223
4.2.5. Полирасширяемость немонотонной системы	225
4.2.6. Различные формы немонотонных рассуждений	225
4.3. Логики умолчаний	227
4.3.1. Введение	227
4.3.2. Теории с умолчаниями	229
4.3.3. Примеры применения умолчаний	229
4.3.4. Расширения теорий с умолчаниями	231
4.3.5. Примеры расширений теорий с умолчаниями	232
4.3.6. Нормальные теории	233
4.3.7. Теория доказательств для нормальных теорий	234
4.3.8. Полуноормальные теории	236
4.3.9. Наследственные системы с исключениями	237
4.4. Модальные логики знания и веры	239
4.4.1. Введение	239
4.4.2. Некоторые элементарные модальные системы	239
4.4.3. Семантика возможных миров	242
4.5. Немонотонные логики Мак-Дермотта	245
4.5.1. Введение	245
4.5.2. Язык логики Мак-Дермотта	246
4.5.3. Пример немонотонной аксиоматической системы	247
4.5.4. Примеры	252
4.5.5. Ценность логики Мак-Дермотта	252
4.6. Автоэпистемические логики	253
4.6.1. Введение	253
4.6.2. Язык и семантика	254
4.6.3. Характеризация синтаксиса	256
4.6.4. Анализ немонотонной логики	257
4.6.5. Семантика возможных миров	260
4.6.6. Пример	263
4.6.7. Области применения	265
5. Формальные грамматики и логическое программирование	266
5.1. Формальные грамматики и логика	266
5.1.1. Введение	266
5.1.2. КС-грамматики	268
5.1.3. Формальное определение КС-грамматик	270
5.1.4. КС-грамматика и хориовские дизъюнкты	274
5.1.5. ОК-грамматики	278
5.1.6. ОК-грамматики и логика	283
5.1.7. Построение синтаксического дерева	287
5.1.8. ОК-грамматики в Прологе	290
5.1.9. Пример	293
5.1.10. Графическое представление и стратегии	295

5.2. Иерархия Хомского	299
5.2.1. Введение	299
5.2.2. Регулярные грамматики	300
5.2.3. Конечные автоматы	301
5.2.4. Пример	304
5.2.5. КС-грамматики и стековые автоматы	305
5.2.6. Пример	308
5.2.7. Грамматики и языки Хомского типа 1	310
5.2.8. Машины Тьюринга и грамматики типа 0	311
5.3. Формализм усиленных сетей переходов	314
5.3.1. Введение	314
5.3.2. Конечные автоматы и диаграммы переходов	315
5.3.3. Базовые сети переходов	316
5.3.4. Пример	318
5.3.5. Рекурсивные сети переходов	320
5.3.6. Пример	323
5.3.7. Усиленные сети переходов	325
5.3.8. Пример	327
5.3.9. Построение синтаксического дерева	328
5.3.10. УП-сети и ОК-грамматики	330
6. Пролог и логическое программирование	333
6.1. Основы языка	333
6.1.1. Введение	333
6.1.2. Термы и объекты	335
6.1.3. Факты и элементарные вопросы	337
6.1.4. Конъюнкция	340
6.1.5. Переменные	340
6.1.6. Анонимные переменные	344
6.1.7. Правила	345
6.1.8. Рекурсивные правила	348
6.1.9. Дизъюнкция	348
6.1.10. Отрицание	349
6.1.11. Области действия имен	350
6.1.12. Операторы	351
6.2. Алгоритмы Пролога	352
6.2.1. Введение	352
6.2.2. Соответствие и унификация	354
6.2.3. Вычисление ответа	358
6.2.4. Встроенные предикаты	362
6.2.5. Отсечение	366
6.3. Инструментарий и пример	371
6.3.1. Введение	371
6.3.2. Вычислительные процедуры	371
6.3.3. Списки	376
6.3.4. Операции над списками	378
6.3.5. Перестановки и сортировки	380
6.3.6. Представление списка в виде разности списков	384
6.3.7. Ввод-вывод	385
6.3.8. Примеры ввода-вывода	387

6.3.9. Анализ и построение термов	389
6.3.10. Объявление оператора	391
6.3.11. Поиск в пространстве решений	394
6.4. Пролог и КС-грамматики	399
6.4.1. Введение	399
6.4.2. Распознавание КС-фраз	399
6.4.3. Анализ КС-фраз	403
6.4.4. ОК-грамматики и атрибутные грамматики	407
Литература	411
Предметный указатель	418

Научное издание

Андре Тейз, Паскаль Грибомон, Жорж Лум и др.

**ЛОГИЧЕСКИЙ ПОДХОД К ИСКУССТВЕННОМУ ИНТЕЛЛЕКТУ:
ОТ КЛАССИЧЕСКОЙ ЛОГИКИ К ЛОГИЧЕСКОМУ
ПРОГРАММИРОВАНИЮ**

Заведующий редакцией академик В. И. Арнольд

Зам. зав. редакцией А. С. Попов

Ст. научн. редактор А. А. Брядинская

Мл. научн. редактор Р. И. Пяткина

Художник В. С. Потапов

Художественный редактор В. И. Шаповалов

Технический редактор О. Г. Лапко

Корректор А. Ф. Рыбальченко

ИБ № 7329

Сдано в набор 2.04.90. Подписано к печати 26.11.90. Формат 84×108¹/₃₂.
Бумага офсет. № 2. Печать высокая. Гарнитура литературная. Объем
6,75 бум. л. Усл. печ. л. 22,68. Усл. кр.-отт. 22,68. Уч.-изд. л. 19,45.
Иад. № 1/7059. Тираж 20 000 экз. Зак. 493. Цена 2 р. 90 к.

Издательство «Мир» В/О «Совэкспорткнига» Государственного комитета
СССР по печати. 129820, ГСП, Москва, И-110, 1-й Рижский пер., 2

Ленинградская типография № 2 головное предприятие ордена Трудового
Красного Знамени Ленинградского объединения «Техническая книга»
им. Евгении Соколовой Государственного комитета СССР по печати,
198052, г. Ленинград, Л-52, Измайловский проспект, 29.