СОДЕРЖАНИЕ

STUDIES IN COMPUTER SCIENCE
AND ARTIFICIAL INTELLIGENCE 13

PRINCIPLES OF LOGIC AND LOGIC PROGRAMMING

G. METAKIDES

European Commission
Brussels, Belgium

A. NERODE

Mathematical Sciences Institute
Cornell University
Ithaca, NY, USA

With the cooperation of

A. SINACHOPOULOS
Université Libre de Bruxelles
Belgium

1996

ELSEVIER

AMSTERDAM – LAUSANNE – NEW YORK – OXFORD – SHANNON – TOKYO

Г. МЕТАКИДЕС
А. НЕРОУД

ПРИНЦИПЫ ЛОГИКИ И ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Перевод с английского
В. А. Захарова,
С. Н. Селезневой и В. В. Спануполо

под редакцией
В. А. Садовничего и В. А. Захарова

ИЗДАТЕЛЬСТВО «ФАКТОРИАЛ»
МОСКВА, 1998

ББК 22.19
М 54
УДК 519.68

Издание осуществлено по инициативе и при поддержке Московского государственного университета им. М. В. Ломоносова

М 54 **Метакидес Г., Нероуд А.** Принципы логики и логическое программирование / Пер. с англ. под ред. к.ф.м.н В. А. Захарова и акад. В. А. Садовничего. — М.: Изд-во «Факториал», 1998. — 288 с. — ISBN 5-88688-037-2.

Книга известных специалистов в области математической логики и логического программирования (Греция, США). Излагаются основные понятия и принципы математической логики. Основное внимание уделяется вопросам применения языка математической логики для представления знаний и формальным методам доказательства теорем с использованием семантических таблиц и резолютивного вывода На примере языка ПРОЛОГ рассматриваются принципы логического программирования. Подробно анализируется механизм вычислений и методика проектирования логических программ. Изложение отличается методическими достоинствами — книга написана в хорошем стиле, не требует специальных предварительных знаний, содержит большое количество примеров и задач.

Может быть рекомендована в качестве учебного пособия для программистов разной квалификации, специалистов по искусственному интеллекту и для всех интересующихся математической логикой, а также теорией и практикой логического программирования.

Научное издание

Метакидес Г., Нероуд А.

Принципы логики и логического программирования

Формат 60 × 90/16. Гарнитура литературная. Усл. печ. л. 18. Подписано к печати 12.9.1998. Тираж 1000 экз.

Издательство «Факториал», 117449, Москва, а/я 331; ЛР № 063537 от 22.07.1994.

Оригинал-макет подготовлен с использованием макропакета **AP** TEX.

Отпечатано с готовых диапозитивов в ППП «Типография «Наука»
121099 Москва, Шубинский пер., 6. Заказ 31

ISBN 0-444-81644-5 (англ.)
ISBN 5-88688-037-2 (русск.)

ПРЕДИСЛОВИЕ К РУССКОМУ ИЗДАНИЮ

Мне доставляет удовольствие представить российским читателям новый учебник, написанный всемирно известными американскими математиками, профессором Корнельского университета, Анилом Нероудом совместно с признанным специалистом в области логического программирования, профессором Георгием Метакидесом. Созданная ими книга весьма необычна, и поэтому нуждается в кратком предисловии, поясняющем некоторые ее замечательные особенности.

До недавнего времени математическая логика не выходила за пределы «чистой» математики и обеспечивала только корректность построения математических теорий. Традиционно считалось, что полноценное изучение этой математической дисциплины целесообразно начинать лишь тогда, когда объем и разнообразие математических знаний учащихся потребуют их упорядочения для того, чтобы осмыслить структуру и принципы устройства всей математики. Поэтому значительная часть учебников по математической логики изначально ориентированы на круг читателей, имеющих достаточно высокую математическую подготовку и свободно владеющих формальным аппаратом современной математики. Вполне естественно, что при таких условиях круг людей, знакомых с методами математической логики, ограничивался выпускниками математических факультетов.

Но с течением лет положение дел сильно изменилось. По мере развития информатики, математическая логика раскрыла свой огромным потенциал практического применения. Совершенствование вычислительной техники и ее математического обеспечения привело к тому, что круг задач, для решения которых можно использовать компьютеры, существенно расширился. Стало ясно, что вычислительные машины можно применять не только для численных расчетов и чисто математических или статистических исследований, но и для обработки информации и знаний самого разного рода. Возникла потребность в разработке такого языка общения с компьютером, который позволял бы человеку наиболее естественным и удобным способом представлять знания, относящиеся к произвольным областям деятельности, и в создании таких методов обработки данных, которые позволяли бы компьютеру извлекать новые сведения и давать ответы на вопросы исходя из имеющихся знаний.

Проведенные исследования показали, что для этой цели лучше всего подходит язык математической логики. Подобно тому, как новые математические теоремы доказываются на основе изначальных аксиом и ранее доказанных теорем, новые знания о предметной области могут быть извлечены из накопленных знаний чисто автоматически при помощи логических методов вывода. Совершенствование аппарата логического вывода привело к созданию нового направления в программировании — логического программирования. Логическое программирование изменило устоявшиеся представления о

технологии проектирования компьютерных программ. В самом общем виде логическая программа представляет собой собрание сведений о предметной области решаемой задачи, записанных на языке, приближенным к естественному языку человеческого общения. Обращаясь к программе с запросами, пользователь может получать все возможные ответы, логически вытекающие из той совокупности знаний, которыми располагает логическая программа. Чем полнее и детальнее будет описание области знаний, к которой относится решаемая задача, тем точнее будут ответы на поставленные вопросы. Диалог программы с пользователем теперь становится похожим на обычную беседу учителя с послушным и аккуратным учеником: учитель наделяет ученика знаниями, а ученик, опираясь на эти знания, безошибочно решает задачи и отвечает на вопросы. Поэтому в логическом программировании для решения поставленной задачи нужно всего лишь как можно точнее и детальнее описать, *что* именно требуется решить. Если решение задачи существует, то в этом случае компьютер самостоятельно «догадается», *как* достичь искомого решения. Логическое программирование открывает новые возможности использования компьютера. Теперь мы можем общаться с ним как с интеллектуальным, хотя и несколько своеобразным, собеседником, который берет на себя значительную часть работы, связанную с организацией вычислений программы. Поэтому основное внимание программиста может быть сосредоточено на самой задаче как таковой, на формировании тех знаний, которые могут оказаться полезными при ее решении. В чем секрет таких необычных свойств логического программирования? Каким образом логика дает возможность компьютеру решать задачи даже тогда, когда человек не указывает алгоритмов их решения? В какой форме человеку нужно представить свои знания об окружающем мире для того, чтобы компьютер мог их понять и сделать из них выводы, т. е. получить новые знания? Ответы на многие возникающие вопросы такого рода можно получить, прочитав эту книгу.

Ее авторам удалось преодолеть самую большую трудность, возникающую при написании учебников по математической логике: они сумели изложить основные принципы и методы этой необычайно формальной и абстрактной науки на очень простом и общедоступном уровне, не поступившись при этом строгостью и полнотой картины. Основная цель книги — объяснить читателям, как можно использовать аппарат математической логики для решения прикладных задач в самых разнообразных областях знаний. Поэтому наибольшее внимание авторы уделяют изучению выразительных возможностей языка математической логики и эффективных методов доказательства теорем. Один из таких методов — метод резолюций — позволяет реализовать на практике концепцию логического программирования, согласно которой вычислительная программа может быть записана при помощи логических формул, играющих роль аксиом, а ее вычисление представлено в виде доказательства

формулы-запроса. Ознакомившись в первых двух главах с основными понятиями классической логики, с устройством и организацией логического вывода, читатель сможет в третьей главе без труда разобраться в технологии проектирования логических программ и механизме их функционирования.

Для понимания книги не требуется никаких специальных знаний, выходящих за рамки школьной программы, и она вполне доступна большинству учащихся выпускных классов российских средних школ. В то же время в ней в той или иной мере рассмотрены основные понятия классической логики, подробно описаны принципы и наиболее существенные особенности логического программирования. Поэтому эту же книгу можно рекомендовать в качестве учебного пособия по математической логике и логическому программированию всем тем, кто хочет получить предварительные сведения об этих разделах математики и информатики.

Читатель, освоивший эту замечательную книгу, конечно же, не должен пребывать в праздной уверенности, что он получил исчерпывающее представление о математической логике и может написать сколь угодно сложные логические программы. Он овладел лишь ключиком, открывающим доступ к одному из разделов современной математики, в котором ему еще предстоит отыскать немало ценных и полезных знаний. И я надеюсь, что для многих из вас этот ключик окажется поистине золотым.

Академик РАН
В. А. Садовничий

Будущему информатике как математической науке *.

*) Время покажет, в какой мере будет справедливо обратное

ВВЕДЕНИЕ

Логика — это наука, которая изучает, каким образом мы выражаем мысли, делаем умозаключения, и как все это можно представить формально. К логическими высказываниями относятся простые предложения и элементарные умозаключения; они подвергаются анализу как с позиции их *формы*, то есть *синтаксиса*, так и с позиции их *интерпретации*, иначе говоря, *семантики*. Изучается также и взаимосвязь синтаксиса и семантики.

Первые упоминания о логике содержатся в произведениях ионийских и эллинских философов и софистов. Однако основателем настоящей логики как науки считается Аристотель. Первоначальный интерес к логике постепенно утрачивался по мере распространения римского владычества над Средиземноморьем. В средние века большинство трудов античных философов — за исключением Платона и Аристотеля — было утеряно или уничтожено, и силлогистика Аристотеля была доступна лишь немногим избранным монахам.

С открытием неевклидовых геометрий и возникновением потребности в теоретическом обосновании математического анализа интерес к логике возродился. В 1879 г. Фреге впервые разработал формальный язык для математики и логики. Однако наибольший стимул к развитию логика получила в связи обнаружением парадоксов теории множеств и последующим широким обсуждением их в математическом сообществе.

В 1895 и 1897 годах Кантор опубликовал первую и вторую части своих исследований по кардинальным и ординальным числам. В этих исследованиях, составивших первоначальное обоснование теории множеств, каждая совокупность объектов рассматривалась как особая сущность, называемая множеством. Но уже в 1897 году Бурали-Форти обнаружил парадокс в канторовской теории множеств. А в 1899 году сам Кантор опубликовал сообщение о парадоксе, затрагивающем его теорию кардинальных чисел. Рассел опубликовал в 1902 году упрощенную форму того же парадокса, вошедшего в историю математики под названием «парадокс Рассела». Его сущность такова.

В канторовской теории множеств каждое множество определяется характеристическим свойством его элементов. Пусть A — множество всех множеств X, обладающих свойством $X \notin X$,

$$A = \{X \mid X \notin X\}$$

Но тогда

$$A \in A \text{ тогда и только тогда, когда } A \notin A$$

что, очевидно, образует противоречие.

Парадоксы теории множеств вызвали у математиков того времени большие сомнения и неуверенность относительно хорошего обоснования математики. Стало понятно также, что создавать по-настоящему непротиворечивые математические теории, лишенные антиномий, можно только путем использования строгой формализации математических понятий и методов. И вот тогда, благодаря настойчивым усилиям Гильберта (см. Замечание 1.4.1(2)), началось развитие аксиоматического подхода, и логика приобрела свой современный вид.

Использование и совершенствование вычислительной техники в начале 1950-ых годов привело к пониманию того, что компьютеры можно применять не только для арифметических вычислений, но также и для символьных преобразований. Таким образом, первые арифметические вычислительные программы и первые программы, призванные давать ответы на простейшие вопросы и доказывать простые теоремы, были написаны почти одновременно. Основные этапы создания общего метода, основанного на логике и позволяющего проводить как обработку предложений, так и символьную манипуляцию, были впервые осуществлены в 1965 году Робинсоном, разработавшим *метод резолюций*, а позднее Ковальским и Колмероэ, которые предложили использовать логику непосредственно как язык логического программирования.

В этой книге раскрываются элементарные понятия математической логики, рассматриваются взаимоотношения и взаимосвязи между логикой и логическим программированием.

В первой главе описана *логика высказываний* (*PL*). Вначале в ней изучается язык высказываний, а также синтаксис и семантика формул *PL*. Далее рассматриваются методы доказательства: метод аксиоматического доказательства, представленный, главным образом, по причинам исторического характера, затем метод таблиц Бета, в котором доказательства сводятся к математическому анализу высказываний, и, наконец, метод резолюций. В последнем из указанных методов доказательства представляются в простой, но чрезвычайно эффективной форме, которая может быть легко реализована в виде программы. Перечисленные методы доказательств обоснованы соответствующими теоремами корректности и полноты.

Во второй главе изучается *логика предикатов* (*PrL*). В логике предикатов более подробно изучается структура высказываний, поскольку язык логики предикатов существенно разнообразнее, нежели язык логики высказываний. Этот язык содержит кванторы и константы, предикаты и функции, и поэтому семантика *PrL* значительно сложнее. Среди прочих разновидностей семантики языка логики предикатов особо выделяются эрбрановские интерпретации, поскольку они позволяют проверять выполнимость предложений *PrL*,

используя методы *PL*. Далее изучаются аксиоматический метод доказательства, метод таблиц Бета и метод резолюций. Метод резолюций вместе с алгоритмом унификации положен в основу логического программирования. Для обоснования этих методов приводятся соответствующие теоремы корректности и полноты.

В третьей главе речь идет о логическом программировании вообще и о языке ПРОЛОГ в частности. Здесь рассматривается также механизм построения вывода в ПРОЛОГе. Логика обеспечивает логическое программирование и ПРОЛОГ выразительным языком и предоставляет теоретическое обоснование, гарантирующее правильность полученных результатов.

Каждая глава содержит примеры и задачи, предназначенные для лучшего усвоения соответствующих тем. Примеры демонстрируют методику решения задач, а сами задачи призваны стимулировать личную инициативу читателя. Книга является самодостаточной, т. е. в ней подробно раскрываются все основные понятия математической логики, и от читателя требуются лишь элементарные знания алгебры и анализа. В связи с этим ее можно рекомендовать студентам любого курса как для самостоятельного изучения, так и для использования в качестве учебного пособия, подкрепляющего соответствующий курс лекций. Книга может также оказаться полезной тем программистам, кто уже имел практический опыт работы с системами логического программирования, не обладая при этом специальной теоретической подготовкой по логике, а также всем тем, кто просто интересуется логикой и логическим программированием. Все темы, затронутые в этой книге, рассматриваются в перспективе перехода от логики к логическому программированию. Поэтому доказательства некоторых теорем, связанных, как правило, с обоснованием полноты аксиоматического метода, опущены. В тех случаях, когда те или иные понятия и доказательства выходят за рамки книги и обсуждаются в ней лишь поверхностно, мы указываем библиографические источники, где они изложены более полно.

В основу настоящего пособия положена книга, опубликованная первым автором в Греции в 1992 году; эта книга была написана по материалам семинаров, посвященным основам логического программирования, проведенных в Корнелле и Рочестере, США, и Университете Патраса, Греция.

Пользуясь случаем, авторы отмечают значительный вклад, который внесла Анета Синахопулос-Сварна в разделы 1.1, 1.8, 2.3, 2.11, 3.2, 3.3, 3.6, 3.8, 3.9, в размещение и расположение общего содержания книги, а также помощь, которую оказал Георгий Потамиас при написании разделов 3.4, 3.5 и 3.7. Содействие при подготовке всего материала книги оказали также аспиранты университета Патраса Костас Контогианнис, Спирос Пападакис и Петрос Петропулос. Все программы, включенные в книгу, равно как и упражнения, были проверены Георгием Потамиасом в системе программирования TURBO-PROLOG, v.2.0.

Глава 1

ЛОГИКА ВЫСКАЗЫВАНИЙ

Μέτρον πάντων χρημάτων άνθρωποζ τῶν μέν όντων ωζ εστί, τών δέ μή όντων ωζ ουχ εστί.

Человек — мера всех вещей; существующих — что они существуют, несуществующих — что они не существуют.

Протагор

§ 1.1. Введение

До 1850 года математические доказательства основывались на эксперименте и интуиции. В 1666 году Лейбниц впервые отметил необходимость создания формального языка, названного им «универсальной характеристикой», который можно было бы использовать в математических формулировках и методологии. Однако его идеи опережали свое время. Начало развития формализма в математике связывается с публикациями Булем, два века спустя, своих трудов «Математический анализ логики» (1847) и «Законы мышления» (1854). Свой вклад в создание логического формализма в математике внесли работы Де Моргана (1847, 1864), Пирса (1867) и Фреге (1879, 1893, 1903). Публикация же трехтомника Рассела и Уайтхеда «Principia Mathematica» стала кульминацией этих усилий, так как в нем была представлена формализующая математику логическая система, в которой теоремы выводились при помощи логических аксиом и правил.

Расцвет логики как самостоятельной науки пришелся на 20-е годы XX века и связан с такими именами, как Лукашевич (1878–1956), Леви (1883–1964), Гёдель (1906–1978), Тарский (1901–1983), Черч (1903–1995) и Клини (1909–1994).

Начиная с 50-х годов нашего века компьютерная технология столкнулась с проблемой использования компьютера для символьных преобразований. Одно из решений этой задачи предлагает функциональное программирование. Этот стиль программирования был разработан Мак-Карти и получил широкое распространение в

США. Другое решение обеспечивает логическое программирование, созданное, преимущественно, европейскими учеными, в том числе, Колмероэ и Ковальским. Логическое программирование придало логике новый импульс. Оно решило многие проблемы, касающиеся строения логических высказываний, при помощи которых компьютер может выполнять процедуры вывода заключений и суждений. Логическое программирование также раскрывает природу и механизмы логических процедур, выполняемых компьютером.

В первую очередь нам необходимо рассмотреть, каким образом формализуются логические и математические высказывания. Как мы увидим в следующих примерах, логические связки являются основными элементами формализации.

1. Конъюнкция формализуется при помощи символа $\wedge$. Предположим, нам известны два свойства некоторого x:

$$A\colon \quad x>3, \qquad B\colon \quad x<10.$$

То есть мы знаем, что x больше трех и меньше десяти. Другими словами, нам известно высказывание

$$A \wedge B,$$

где логическая связка $\wedge$ соответствует соединительному союзу «и». Таким образом, высказывание $A \wedge B$ утверждает, что $x>3$ и $x<10$, т. е. $3<x<10$.

2. Отрицание формализуется при помощи символа $\neg$. Например, рассмотрим высказывание

$$C\colon \quad \text{50 делится на 7,}$$

$\neg C$ обозначает, что 50 не делится на 7.

3. Дизъюнкция обозначается при помощи символа $\vee$. Если D и E — высказывания

$$D\colon \quad \text{60 делится на 6,} \qquad E\colon \quad \text{60 делится на 5,}$$

то высказывание

$$D \vee E$$

утверждает, что «60 делится на 6 или 60 делится на 5». Символ $\vee$ не является исключающе разделительным, так как 60 является кратным 6 и 5 также, как и 20, не упомянутого в высказываниях D и E.

4. Импликация «если ..., то ...» обозначается в логике символом $\rightarrow$. Если F и G есть высказывания

$$F\colon \text{ число } a \text{ кратно 10,} \qquad G\colon \text{ число } a \text{ кратно 5,}$$

то высказывание

$$F \to G$$

утверждает, что «если число a кратно 10, то оно кратно 5».

5. Равносильность «...тогда и только тогда, когда ...» обозначается символом $\leftrightarrow$. Например, если H и I есть высказывания

H: 16 кратно 2, I: 16 является четным числом,

то формально мы записываем $H \leftrightarrow I$.

Свойства логических связок не одинаковы. В примере о высказываниях A и B запись $A \wedge B$ по существу не отличается от записи $B \wedge A$. Интуитивно понятно, что логическая связка $\vee$ (равно как и $\wedge$) должна быть коммутативной. Однако ответ на вопрос о коммутативности связки $\to$ будет отрицательным: высказывание $G \to F$, «если a кратно 5, то a кратно 10» неверно, так как 15 кратно 5, но не кратно 10. Поэтому нам необходимо изучить свойства логических связок.

Существуют также различные типы высказываний. Например, высказывания

«6 кратно 6»,

«7 — простое число»,

«рациональное число есть либо 0, либо не равно 0»

являются «верными», «правильными», в то время, как высказывания

«50 — нечетное число»,

«50 делится на 7»,

«если $x = 3$, то $x = 5$»,

относятся к числу «неверных», «ошибочных». Наконец, такие высказывания, как

«идет дождь»,

«я голоден»,

«x равно своему абсолютному значению»,

иногда являются «правильными», а иногда — «ошибочными».

Интуитивно понятно, что каждое высказывание имеет некоторую интерпретацию: оно может выражать истину или ложь, иначе говоря, может быть истинным или ложным. Но что делает высказывание истинным? Какова связь между интерпретацией составного высказывания и интерпретациями высказываний, составляющих его? И какую роль играют логические связки в интерпретациях? Нам необходимо изучить интерпретации высказываний и определить роль логических связок в этих интерпретациях.

Наша собственная логика позволяет нам принимать решения и делать выводы. Например, мы говорим «если $x > 3$, то $x > 0$», или, формально,

$$(x > 3) \to (x > 0).$$

Давайте предположим, что $x > 3$. Тогда $x > 0$. Какое правило позволяет нам заключать, что $x > 0$ в случае, когда высказывание $x > 3$ истинно? И обладает ли это правило всеобщностью, другими словами, если $M \rightarrow N$ истинно и M истинно, то всегда ли истинно N?

Обратимся к другому примеру. Мы знаем, что для того, чтобы пойти в кино, нужно иметь деньги на билет. Таким образом,

$$\text{иду в кино} \rightarrow \text{имею деньги на билет.} \qquad (1)$$

Давайте предположим, что мы не имеем достаточно денег на билет, т. е.

$$\neg\,(\text{имею деньги на билет}). \qquad (2)$$

Можем ли мы пойти в кино? Каково влияние отрицания? Какой вид будет иметь (1) с отрицанием?

$$\neg\,(\text{иду в кино}) \rightarrow \neg\,(\text{имею деньги на билет})?$$
$$\text{Или } \neg\,(\text{иду в кино} \rightarrow \text{имею деньги на билет})?$$
$$\text{Или } \neg\,(\text{имею деньги на билет}) \rightarrow \neg\,(\text{иду в кино})?$$

И какое заключение истинно, если истинны (1) и (2)?

Для того, чтобы ответить на эти вопросы, нам нужно изучить синтаксис высказываний и правила, которые определяют, когда и как мы можем выводить истинные заключения, располагая в качестве посылок имеющимися в распоряжении данными. Логика высказываний, равно как и более сложная логика предикатов, исследует эти вопросы.

В главе 1, посвященной логике высказываний (PL), мы изучим высказывания PL с точки зрения их синтаксиса и семантики. Мы также исследуем методы получения заключений из посылок и определим формы высказываний PL, необходимые для представления знаний, разрешающих процедур и автоматического доказательства теорем в логическом программировании.

§ 1.2. Язык логики высказываний

Алфавит, синтаксис и семантика

Язык логики высказываний (PL) является весьма строгим языком. В терминах этого языка мы формализуем те части повседневной речи, которые необходимы для представления логических и математических понятий. Имея формализованные в этом языке высказывания, мы затем исследуем их в соответствии с их структурой и истинностью.

Каждый формальный язык задается:

а) *алфавитом*, содержащим все символы языка;

б) *синтаксисом*, определяющим, каким образом из символов формируются высказывания;

в) *семантикой*, при помощи которой интерпретируется элементы языка путем приписывания значений символам алфавита.

Алфавит языка логики высказываний состоит:

1) из пропозициональных символов: $A, A_1, A_2, \ldots, B, B_1, B_2, \ldots$;

2) из логических связок: $\vee, \wedge, \neg, \rightarrow, \leftrightarrow$;

3) из запятой и скобок: «,» и «(», «)».

Логические связки интуитивно соответствуют частицам и союзам, которые мы используем в повседневной речи. Соответствие между ними таково:

$\vee$, дизъюнкция — или;

$\wedge$, конъюнкция — и;

$\rightarrow$, импликация — если ..., то ...;

$\leftrightarrow$, эквивалентность — ...тогда и только тогда, когда ...;

$\neg$, отрицание — не.

Произвольная последовательность символов, принадлежащих алфавиту языка, называется *выражением*. Например, $A \vee \vee A$, $A \vee B$, $\leftrightarrow A$ — выражения языка логики высказываний. Некоторые *правильно построенные* выражения рассматриваются синтаксисом как высказывания. В следующем определении 1.2.1 вводится понятие высказывания и, тем самым, описываются законы синтаксиса языка логики высказываний.

Определение 1.2.1 (индуктивное определение высказывания). 1. Пропозициональные символы являются высказываниями и называются *атомарными высказываниями* или *атомами*.

2. Если σ и τ — высказывания, то выражения $(\sigma \wedge \tau)$, $(\sigma \vee \tau)$, $(\sigma \rightarrow \tau)$, $(\sigma \leftrightarrow \tau)$, $(\neg\sigma)$ — также высказывания; их называют *составными высказываниями*.

3. Выражения, построенные в соответствии с пунктами 1 и 2, и только они являются высказываниями. □ 1.2.1

Выражения $\vee A \vee B$ и $\leftrightarrow A$, таким образом, не являются высказываниями, так как они построены вопреки пунктам 1 и 2. А вот выражения $(A \vee B)$ и $((\neg A) \vee (B \leftrightarrow (\neg C)))$ в самом деле являются высказываниями.

Приведенное выше определение 1.2.1 использует метод индукции для определения составного высказывания: пункт 1 есть *базис индукции*, пункт 2 есть *индуктивный переход*. Индукция проводится по логической длине, структуре высказывания. Под «логической длиной» высказывания A мы подразумеваем натуральное

число, обозначающее число, тип и порядок появления логических связок, используемых при построении A, начиная с атомарных высказываний, составляющих A. Нам известно, что *принцип математической индукции* справедлив для натуральных чисел.

Определение 1.2.2 (математическая индукция). Если $P(a)$ обозначает свойство натурального числа a и нам известно, что

а) $P(0)$ справедливо,

б) для любого натурального n из истинности $P(n)$ вытекает истинность $P(n+1)$,

то мы можем заключить, что для всех натуральных чисел n истинно $P(n)$. □ 1.2.2

Тот факт, что множество натуральных чисел вполне упорядочено (под этим подразумевается, что каждое непустое подмножество натуральных чисел имеет наименьший элемент), вместе с принципом математической индукции приводят ко второй форме индукции, называемой *полной*.

Определение 1.2.3 (полная индукция). Если $P(a)$ есть свойство натурального числа a, и нам известно, что

а) $P(0)$ справедливо,

б) для всех натуральных чисел m и n, таких, что $m < n$, из $P(m)$ можно вывести $P(n)$,

то для всех натуральных чисел n истинно $P(n)$. □ 1.2.3

Мы использовали полную индукцию в индуктивном определении высказываний (определение 1.2.1). Теперь применим ее в следующем определении 1.2.4.

Определение 1.2.4 (общая схема индукции для высказываний). Пусть P — пропозициональное свойство. Мы будем писать $P(\sigma)$, чтобы обозначить, что *высказывание σ имеет свойство P*. Если мы докажем, что

а) $P(A)$ — истинно для любого атома A языка,

б) если σ_1 и σ_2 — высказывания и $P(\sigma_1)$, $P(\sigma_2)$, то $P((\sigma_1 \wedge \sigma_2))$, $P((\sigma_1 \vee \sigma_2))$, $P((\sigma_1 \to \sigma_2))$, $P((\sigma_1 \leftrightarrow \sigma_2))$, $P((\neg\sigma_1))$ и $P((\neg\sigma_2))$,

то мы можем заключить, что свойство $P(\sigma)$ справедливо для любого высказывания σ нашего языка. □ 1.2.4

Пример 1.2.5.

1. Свойство P «число левых скобок совпадает с числом правых скобок» истинно для всех высказываний PL.

В самом деле, согласно общей схеме индукции мы имеем:

а) в пропозициональных символах число левых и правых скобок равно 0;

б) если в высказываниях σ_1 и σ_2 число левых и правых скобок совпадает и равно для σ_1 и σ_2 соответственно n и m, то в высказываниях $(\sigma_1 \wedge \sigma_2)$, $(\sigma_1 \vee \sigma_2)$, $(\sigma_1 \to \sigma_2)$, $(\sigma_1 \leftrightarrow \sigma_2)$, $(\neg\sigma_1)$ и $(\neg\sigma_2)$ число

левых и правых скобок совпадает и равно $n+m+1$ в первых четырех, $n+1$ и $m+1$ соответственно — в двух последних.

2. Выражение $((\neg(A \wedge B)) \to C)$ является высказыванием.

Доказательство основано на определении 1.2.1:

1) A, B являются высказываниями по определению 1.2.1(1);

2) $(A \wedge B)$ является высказыванием по определению 1.2.1(2);

3) $(\neg(A \wedge B))$ является высказыванием по определению 1.2.1(2);

4) C является высказыванием по определению 1.2.1(2);

5) $((\neg(A \wedge B) \to C)$ является высказыванием по определению 1.2.1(2).

3. Выражения, приведенные ниже, не являются высказываниями:

$\neg$ — символ $\neg$ не является атомом;

$\wedge \to A$ — символ $\wedge$ не является высказыванием;

$(A \wedge B$ — нет правой скобки, соответствующей левой. □ 1.2.5

Пример 1.2.6. Рассмотрим повествовательное предложение обыденной речи:

F: «Если нет дождя, то я пойду на прогулку».

Для того, чтобы формализовать его в логике высказываний, можно ввести вспомогательные пропозициональные символы:

A: «Идет дождь»; B: «Я иду на прогулку».

Тогда предложение F соответствует высказыванию $((\neg A) \to B)$.

Если не возникает путаницы, то скобки можно опустить:

$$F: \quad \neg A \to B. \qquad \square\ 1.2.6$$

Каждый атом, который встречается в правильно построенной формуле[1]) D рассматривается как подформула D. Например, A и B — подформулы F в примере 1.2.6. Более того, каждая правильно построенная составная часть D является также подформулой D. Рассмотрим в качестве примера формулу

$$D\colon (A \wedge \neg B) \to ((C \vee \neg A) \to B).$$

Тогда $\neg A$, $C \vee \neg A$, $A \wedge \neg B$ равно, как и $(C \vee \neg A) \to B$, являются подформулами D. Наконец, сама формула D, т. е. $(A \wedge \neg B) \to ((C \vee \vee \neg A) \to B)$, также рассматривается как подформула D.

Множество $\mathrm{subform}(\sigma)$ всех подформул правильно построенной формулы σ определяется индуктивно:

Определение 1.2.7. 1) Если σ — атом A, то $\mathrm{subform}(\sigma) = \{A\}$;

[1]) Высказывания логики высказываний называются также формулами. — *Прим. перев*

2) если σ имеет вид $\neg\tau$, то $\mathrm{subform}(\sigma) = \mathrm{subform}(\tau) \cup \{\sigma\}$;
3) если σ имеет вид $\tau \circ \varphi$, где $\circ \in \{\vee, \wedge, \to, \leftrightarrow\}$, то $\mathrm{subform}(\sigma) = \mathrm{subform}(\tau) \cup \mathrm{subform}(\varphi) \cup \{\sigma\}$. □ 1.2.7

Замечание 1.2.8. Чтобы избежать неоднозначностей при использовании связок в формулах без скобок, мы будем считать, что $\neg$ связывает сильнее, чем все остальные связки, $\wedge$ и $\vee$ связывают сильнее, чем $\leftrightarrow$ и $\to$, $\leftrightarrow$ связывает сильнее, чем $\to$. Таким образом, высказывания

$$\neg A \to B \vee C, \quad A \wedge B \to C, \quad A \to B \leftrightarrow C$$

обозначают соответственно

$$((\neg A) \to (B \vee C)), \quad ((A \wedge B) \to C), \quad (A \to (B \leftrightarrow C)). \quad \square\ 1.2.8$$

Замечание 1.2.9. Следуя сложившейся традиции, символ $\leftarrow$ мы будем также рассматривать как логическую связку, $B \leftarrow A$ обозначает то же, что и $A \to B$. □ 1.2.9

§ 1.3. Понятие семантики в логике высказываний

Означивания и истинностные означивания

Согласно определению 1.2.1 высказывания являются общим и абстрактным объектом. Мы хотим интерпретировать эти абстрактные объекты при помощи семантики. Это означает, что нас интересует определение условий, при которых высказывание является истинным или ложным. Поэтому мы создадим структуру, область определения которой — $\{t, f\}$, где t, f обозначают истинностные значения «истина» и «ложь» соответственно, и попытаемся приписать одно из этих значений каждому высказыванию. Используемый метод приписывания истинностных значений, подкрепленный необходимыми определениями, и составляет семантику логики высказываний.

Определение 1.3.1. *Означиванием* назовем произвольную функцию

$$F\colon Q \mapsto \{t, f\},$$

где Q — множество атомов языка. □ 1.3.1

Таким образом, означивание приписывает истинностные значения атомам языка.

Теперь мы введем на множестве $\{t, f\}$ внутренние алгебраические операции для того, чтобы преобразовать его в алгебраическую структуру. Внутренние операторы над t, f, которые мы определим, а именно, $\sim$, $\sqcup$, $\sqcap$, $\rightsquigarrow$ и $\leftrightsquigarrow$, будут соответствовать логическим связкам $\neg$, $\wedge$, $\vee$, $\to$ и $\leftrightarrow$. Сходство соответствующих символов не случайна,

она отражает их соответствие. Рассмотрим, например, высказывание $A \vee B$. Приписав некоторые истинностные значения A и B, мы интерпретируем $A \vee B$, используя интерпретации A и B и применяя операцию $\sqcup$, на основании определения 1.3.2.

Определение 1.3.2. Внутренние операции $\sim$, $\sqcup$, $\sqcap$, $\rightsquigarrow$ и $\leftrightsquigarrow$ над $\{t, f\}$ определяются следующими таблицами.

	$\sim$
t	f
f	t

$\sqcup$	t	f
t	t	t
f	t	f

$\sqcap$	t	f
t	t	f
f	f	f

$\rightsquigarrow$	t	f
t	t	f
f	t	t

$\leftrightsquigarrow$	t	f
t	t	f
f	f	t

В приведенных таблицах в первом столбце перечислены значения первого аргумента и в первой строке — значения второго аргумента соответствующих операций. □ 1.3.2

Структура $(\{t, f\}, \sim, \sqcup, \sqcap)$, в которой операции $\sim$, $\sqcup$, $\sqcap$ определены описанными таблицами, является двузначной булевой алгеброй.

Булевы алгебры важны как для семантики логики высказываний (см. [RaSi70, Curr63, Rasi74, Raut79]), так и для изучения теоретических основ информатики вообще.

Определение 1.3.3. Пусть S — множество высказываний нашего языка. *Истинностным означиванием*, или *булевым означиванием* назовем такую функцию

$$V\colon S \mapsto \{t, f\},$$

что для произвольных $\sigma, \tau \in S$ верно:

а) если σ — атом, то $V(\sigma) \in \{t, f\}$;

б) $V(\neg\sigma) = \sim V(\sigma)$;

в) $V(\sigma \vee \tau) = V(\sigma) \sqcup V(\tau)$;

г) $V(\sigma \wedge \tau) = V(\sigma) \sqcap V(\tau)$;

д) $V(\sigma \to \tau) = V(\sigma) \rightsquigarrow V(\tau)$;

е) $V(\sigma \leftrightarrow \tau) = V(\sigma) \leftrightsquigarrow V(\tau)$. □ 1.3.3

Согласно этому определению истинностные значения атомов некоторого высказывания преобразуются алгебраическими операциями $\sim, \sqcup, \sqcap, \rightsquigarrow, \leftrightsquigarrow$ и определяют тем самым истинностное означивание рассматриваемого высказывания (см. [Smul68]).

Означивание приписывает истинностное значение, t или f, атомам языка. Истинностное означивание является расширением означивания на множество высказываний языка. В последующей теореме 1.3.4 будет доказано, что каждое означивание может быть однозначно расширено до истинностного означивания.

Теорема 1.3.4. *Для каждого означивания F существует одно и только одно истинностное означивание V, являющееся расширением F.*

Доказательство. Проведем доказательство индукцией по длине высказывания.

Пусть F — означивание, Q — множество атомов. Индуктивно определим истинностное означивание V, полагая

а) $V(A) = F(A)$ для каждого $A \in Q$.

Пусть σ, φ — высказывания. Если $V(\sigma)$ и $V(\varphi)$ уже определены, то положим

б) $V(\neg\sigma) = \sim V(\sigma)$;

в) $V(\sigma \vee \varphi) = V(\sigma) \sqcup V(\varphi)$;

г) $V(\sigma \wedge \varphi) = V(\sigma) \sqcap V(\varphi)$;

д) $V(\sigma \to \varphi) = V(\sigma) \rightsquigarrow V(\varphi)$;

е) $V(\sigma \leftrightarrow \varphi) = V(\sigma) \leftrightsquigarrow V(\varphi)$.

Очевидно, что V является истинностным означиванием, расширяющим F.

Остается доказать, что если истинностные означивания V_1 и V_2 являются расширениями F, то равенство $V_1(\varphi) = V_2(\varphi)$ выполняется для любого высказывания φ. Докажем по индукции свойство P:

$$P(\varphi): \quad V_1(\varphi) = V_2(\varphi):$$

Базис индукции.

Для каждого пропозиционального символа A верно, что $V_1(A) = V_2(A)$, так как V_1, V_2 — расширения F.

Индуктивный переход.

Предположим, что $V_1(\sigma) = V_2(\sigma)$ и $V_1(\varphi) = V_2(\varphi)$. Тогда справедливы соотношения:

$V_1(\neg\sigma) = \sim V_1(\sigma) = \sim V_2(\varphi) = V_2(\neg\varphi)$;

$V_1(\sigma \vee \varphi) = V_1(\sigma) \sqcup V_1(\varphi) = V_2(\sigma) \sqcup V_2(\varphi) = V_2(\sigma \vee \varphi)$;

$V_1(\sigma \wedge \varphi) = V_1(\sigma) \sqcap V_1(\varphi) = V_2(\sigma) \sqcap V_2(\varphi) = V_2(\sigma \wedge \varphi)$;

$V_1(\sigma \to \varphi) = V_1(\sigma) \rightsquigarrow V_1(\varphi) = V_2(\sigma) \rightsquigarrow V_2(\varphi) = V_2(\sigma \to \varphi)$;

$V_1(\sigma \leftrightarrow \varphi) = V_1(\sigma) \leftrightsquigarrow V_1(\varphi) = V_2(\sigma) \leftrightsquigarrow V_2(\varphi) = V_2(\sigma \leftrightarrow \varphi)$.

Таким образом, V_1 и V_2 имеют одинаковые значения для каждого высказывания, и поэтому совпадают. □ 1.3.4

Следствие 1.3.5 непосредственно вытекает из теоремы 1.3.4.

Следствие 1.3.5. *Пусть σ — высказывание, все атомы которого содержатся в списке $A_1, \ldots, A_k$. Если два истинностных означивания V_1, V_2 имеют одни и те же значения на множестве $\{A_1, \ldots, A_k\}$, то $V_1(\sigma) = V_2(\sigma)$.* □ 1.3.5

Пример 1.3.6. Вычисление истинностного означивания по заданному означиванию.

Пусть S — множество атомарных высказываний, $S = \{A_1, A_2\}$, F — такое означивание, что

$$F(A_1) = t, \qquad F(A_2) = f.$$

По теореме 1.3.4 искомое истинностное означивание V_F, расширяющее F, определено однозначно.

Положим, что

$$V_F(A_1) := t, \qquad V_F(A_2) := f,$$

где $:=$ обозначает «равно по определению».

Теперь мы можем вычислить значения истинностного означивания V_F для каждого множества высказываний, которые содержат только атомы A_1 и A_2:

$V_F(A_1 \wedge A_2) = V_F(A_1) \sqcap V_F(A_2) = t \sqcap f = f$;

$V_F(A_1 \vee A_2) = V_F(A_1) \sqcup V_F(A_2) = t \sqcup f = t$;

$V_F((A_1 \wedge A_2) \to A_2) = V_F(A_1 \vee A_2) \rightsquigarrow V_F(A_2) = t \rightsquigarrow f = f$;

и т. д. □ 1.3.6

Мы можем классифицировать высказывания логики высказываний в соответствии с истинностными значениями, которые они приобретают.

Определение 1.3.7. Высказывание σ называется *логически истинным*, или *тавтологией*, если *для каждого истинностного означивания V* верно, что $V(\sigma) = t$. Мы будем пользоваться записью $\models \sigma$ для обозначения тавтологии. В свою очередь, запись $\not\models$ будет обозначать, что σ не является тавтологией, т. е. *существует истинностное означивание V, такое, что $V(\sigma) = f$.*

Высказывание σ называется *выполнимым* или *подтверждаемым*, если *существует такое истинностное означивание V*, что $V(\sigma) = t$. При этом про истинностное означивание V, для которого $V(\sigma) = t$, говорят, что оно *подтверждает* высказывание σ.

Высказывание σ, такое, что *для каждого истинностного означивания V* верно $V(\sigma) = f$, называется *логически ложным*, или *невыполнимым*, или *противоречием*. □ 1.3.7

Замечание 1.3.8. Пусть V — истинностное означивание, и множество

$$S_V = \{\sigma\colon V(\sigma) = t\}$$

состоит из всех высказываний языка, выполнимых при означивании V. Если высказывание φ принадлежит множеству S_V для каждого истинностного означивания V, то φ является тавтологией. Каждое множество S_V, где V — истинностное означивание, составляет возможный мир (см. [Fitt69, Chel80]) множества высказываний PL. Каждый возможный мир является *аристотелевым* в том смысле, что для каждого высказывания σ только одно из двух высказываний σ и $\neg\sigma$ может быть истинно в этом мире. И в самом деле, мы имеем либо $V(\sigma) = t$, либо $V(\sigma) = f$, что равносильно $V(\neg\sigma) = t$, сообразно аристотелеву принципу *исключенного третьего*, замечание 1.4.4).

Возможные миры — основное понятие *семантики Крипке*. Семантика Крипке используется при изучении модальной логики. В модальной логике кроме логических связок содержатся специальные символы, как, например, $\diamond$, называемые модальными операторами, что расширяет выразительные возможности языка. Поэтому в модальном языке кроме выражения A встречается выражение $\diamond A$, которое может интерпретироваться как «иногда A истинно», или «A будет истинно в будущем».

Модальная логика позволяет нам выражать особые свойства, затрагивающие расположение и порядок выполнения операторов программы.

Рассмотрим, например, программу на языке ФОРТРАН, предназначенную для вычисления $n!$ для всех целых чисел n от 1 до 10.

```
      k = 1
      do 10 i = 1, ..., 10
      k = k * i
10    write(*, *)k
      end
```

Пусть A и B — высказывания модальной логики,

$$A := \text{write}(*, *)k \text{ и } B := \text{end}.$$

В тот момент, когда программа начинает выполняться, высказывания $\diamond A$ и $\diamond B$ истинны. Это означает, что программа когда-то в будущем вычислит $n!$ и когда-то в будущем она остановится. □ 1.3.8

Определение 1.3.9. Два высказывания, σ и τ, такие, что $V(\sigma) = V(\tau)$ для каждого истинностного означивания V, называются *логически эквивалентными*. Для обозначения эквивалентных высказываний будем использовать запись $\sigma \equiv \tau$. □ 1.3.9

Пример 1.3.10. Высказывания $A \vee \neg A$ и $((A \to B) \to A) \to A$ — тавтологии.

Доказательство. Сначала докажем, что $A \vee \neg A$ — тавтология. Пусть V_1 и V_2 — два таких истинностных означивания, что

$$V_1(A) = t, \qquad V_2(A) = f.$$

Заметим, что

$$V_1(A \vee \neg A) = V_1(A) \sqcup V_1(\neg A) = V_1(A) \sqcup (\sim V_2(A)) = t \sqcup (\sim t) = t \sqcup f = t,$$

$$V_2(A \vee \neg A) = V_2(A) \sqcup V_2(\neg A) = V_2(A) \sqcup (\sim V_2(A)) = f \sqcup (\sim f) = f \sqcup t = t.$$

Произвольное истинностное означивание V на A совпадает либо с V_1, либо с V_2.

Таким образом, согласно следствию 1.3.5 выполняется одно из двух:

$$V(A \vee \neg A) = V_1(A \vee \neg A) = t \quad \text{или} \quad V(A \vee \neg A) = V_2(A \vee \neg A) = t.$$

То есть высказывание истинно для любого истинностного означивания V. Следовательно, оно является тавтологией.

Теперь докажем, что $((A \to B) \to A) \to A$ — тавтология. На $Q = \{A, B\}$ имеется четыре различных означивания F_1, F_2, F_3, F_4:

$$F_1(A) = t, \qquad F_1(B) = t,$$
$$F_2(A) = t, \qquad F_2(B) = f,$$
$$F_3(A) = f, \qquad F_3(B) = t,$$
$$F_4(A) = f, \qquad F_4(B) = f.$$

Для каждого из этих означиваний существует единственное истинностное означивание V_k, $k \in \{1, 2, 3, 4\}$, расширяющее его. Несложно проверить, что

$$V_k(((A \to B) \to A) \to A) = t \text{ для каждого } k \in \{1, 2, 3, 4\}.$$

Произвольное истинностное означивание V на пропозициональных символах A и B совпадает с V_k для некоторого $k \in \{1, 2, 3, 4\}$. В силу следствия 1.3.5 справедливо равенство

$$V(((A \to B) \to A) \to A) = V_k(((A \to B) \to A) \to A) = t.$$

Следовательно, высказывание $((A \to B) \to A) \to A$ истинно для любого истинностного означивания и поэтому является тавтологией. □ 1.3.10

Пример 1.3.11. Высказывание $((\neg A \wedge B) \to C) \vee D)$ (обозначим его K) выполнимо. В самом деле, пусть F — такое означивание пропозициональных символов K, что

$$F(A) = t, \quad F(B) = t, \quad F(C) = f, \quad F(D) = f.$$

Истинностное означивание V, расширяющее F, приписывает следующие значения:

$$V(A) = t, \quad V(B) = t, \quad V(C) = f, \quad V(D) = f.$$

Тогда

$$V(\neg A) = \sim V(A) = f,$$
$$V(\neg A \wedge B) = V(\neg A) \sqcap V(B) = f \sqcap t = f,$$
$$V((\neg A \wedge B) \to C) = V(\neg A \wedge B) \rightsquigarrow V(C) = f \rightsquigarrow f = t,$$
$$V(((\neg A \wedge B) \to C) \vee D) = V((\neg A \wedge B) \to C) \sqcup V(D) = t \sqcup f = t,$$

следовательно, $V(K) = t$.

Высказывание K не является тавтологией. И в самом деле, пусть G — такое означивание, что

$$G(A) = t, \quad G(B) = t, \quad G(C) = f, \quad G(D) = f.$$

Предположим, что G' — истинностное означивание, расширяющее G. Применяя метод, аналогичный используемому при доказательстве того, что $V(K) = t$, мы можем доказать, что $G'(K) = f$. Тогда, по определению 1.3.7, высказывание K не является тавтологией. □ 1.3.11

§ 1.4. Таблицы истинности

В предыдущем параграфе мы определили означивание атомов языка и, затем, распространив означивание атомов на составные высказывания, определили истинностные означивания. При помощи этого метода мы можем установить, является ли высказывание тавтологией, противоречием, или оно выполнимо. Однако, чем больше составляющих содержит высказывание, тем более сложным становится этот метод.

Чтобы сделать его несколько проще, мы соберем все возможные означивания атомов высказывания в таблицу, называемую *таблицей истинности* высказывания. Итак, для составных высказываний $\neg A$, $A \vee B$, $A \wedge B$, $A \to B$ и $A \leftrightarrow B$ мы имеем следующие таблицы:

A	$\neg A$
t	f
f	t

A	B	$A \vee B$
t	t	t
f	t	t
t	f	t
f	f	f

A	B	$A \wedge B$
t	t	t
f	t	f
t	f	f
f	f	f

A	B	$A \to B$
t	t	t
f	t	t
t	f	f
f	f	t

A	B	$A \leftrightarrow B$
t	t	t
f	t	f
t	f	f
f	f	t

Каждая строка в этих таблицах соответствует означиванию и его однозначному расширению до истинностного означивания. Например, вторая строка в таблице $A \wedge B$ имеет вид t, f, f. Она означает, что A принимает значение t (это первый элемент строки), B принимает значение f (это второй элемент строки). На основании определения 1.3.2 и определения 1.3.3 мы знаем, что соответствующее истинностное значение $A \wedge B$ есть $t \sqcap f$, а именно f. Таким образом, мы находим f (это третий элемент строки).

В дальнейшем мы будем использовать описанные выше таблицы истинности как определения результирующих значений при использовании логических связок, не ссылаясь на означивания и истинностные означивания.

Замечание 1.4.1. 1. Дизъюнкция логики высказываний не является разделительной, т. е. $A \vee B$ принимает значение t и тогда, когда и A, и B принимают значение t. В повседневной речи, напротив, чаще всего фигурирует разделительная дизъюнкция. Например, мы говорим: «Я останусь дома или пойду в кино», подразумевая при этом выбор только одного варианта, но никак не обоих.

2. Высказывание $A \to B$ принимает значение f только тогда, когда A принимает значение t, а B принимает значение f. Таким образом, если A есть f, то $A \to B$ есть t, независимо от того какое значение принимает B. Эти свойства $\to$ и, как следствие, та часть логики высказываний, которая основывается на этих свойствах, принимаются не всеми логическими школами.

В 1910–1920 гг. в области прикладных наук в Европе преобладала концепция школы Гильберта (Гильберт, 1862–1943) (см. [Boye68, Heij67]):

«Математика и логика должны иметь аксиоматическое основание, и используемые этими науками методы должны быть максимально формальными».

Но в 1918 г. Брауэр (1881–1966) опубликовал суровую критику логики высказываний. Представленная в его работе альтернативная система была названа *интуиционистской логикой*. Это название происходит из утверждаемого Брауэром факта, основанного на мнении Канта, что мы ощущаем натуральные числа (и, соответственно, логику, которая характеризует науку) только интуитивно (см. [Dumm77, Brou75]). Интуиционистская логика никогда целиком не замещала логику, основанную на концепции Гильберта, однако, сегодня она имеет некоторые приложения в информатике.

С другой стороны, логика высказываний предлагает эффективный инструмент (в аристотелевом смысле) для изучения и решения различных задач и составляет основу современной науки и техники.

□ 1.4.1

Резюмируем:

опираясь на индуктивное определение высказываний и на определение значений логических связок, мы можем построить таблицу истинности для любого высказывания, приписав значения всем атомам, составляющим его.

Пример 1.4.2. Таблица истинности высказывания $A \wedge B \to C$:

A	B	C	$A \wedge B$	$A \wedge B \to C$
t	t	t	t	t
t	t	f	t	f
t	f	t	f	t
t	f	f	f	t
f	t	t	f	t
f	t	f	f	t
f	f	t	f	t
f	f	f	f	t

Для тройки атомов (A, B, C) тройка истинностных значений (f, t, f) обращает $A \wedge B \to C$ в истину, тогда как тройка (t, t, f) обращает его в ложь. □ 1.4.2

Сокращенной таблицей истинности высказывания $A \wedge B \to C$ называется его таблица истинности, в которой вспомогательный четвертый столбец пропущен.

Сокращенная таблица истинности высказывания, содержащего n атомов, состоит из 2^n строк и $n+1$ столбца.

При помощи таблицы истинности мы можем устанавливать, когда высказывание истинно, а когда — ложно. В силу следствия 1.3.5, если в таблицах истинности для двух высказываний столбцы значений атомов и последние столбцы совпадают, то эти два высказывания логически эквивалентны.

Пример 1.4.3.

1. Высказывание $((A \to B) \to A) \to A$ является тавтологией.

A	B	$A \to B$	$(A \to B) \to A$	$((A \to B) \to A) \to A$
t	t	t	t	t
t	f	f	t	t
f	t	t	f	t
f	f	t	f	t

2. Высказывание $(P \to Q) \wedge (P \wedge \neg Q)$ невыполнимо.

P	Q	$\neg Q$	$P \to Q$	$P \wedge \neg Q$	$(P \to Q) \wedge (P \wedge \neg Q)$
t	t	f	t	f	f
t	f	t	f	t	f
f	t	f	t	f	f
f	f	t	t	f	f

Замечание 1.4.4. Основываясь на определении 1.3.7 мы имеем:

1) высказывание является тавтологией тогда и только тогда, когда его отрицание невыполнимо;

2) высказывание невыполнимо тогда и только тогда, когда его отрицание является тавтологией;

3) высказывание, являющееся тавтологией, выполнимо, тогда как выполнимое высказывание не обязательно является тавтологией;

4) некоторые часто используемые тавтологии:

1) $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$ — закон де Моргана;

2) $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$ — закон де Моргана;

3) $\neg(\neg A) \equiv A$ — закон двойного отрицания;

4) $(A \to B) \equiv (\neg B \to \neg A)$ — закон контрапозиции;

5) $(B \to C) \to ((A \to B) \to (A \to C))$ — первый закон силлогистики;

6) $(A \to B) \to ((B \to C) \to (A \to C))$ — второй закон силлогистики;

7) $(A \to (B \to C)) \equiv ((A \wedge B) \to C)$ — закон транспозиции;

8) $A \vee \neg A$ — закон исключенного третьего.

Высказывания 5 и 6 называются аристотелевыми законами силлогистики. □ 1.4.4

Метод нахождения значения составного высказывания при помощи таблицы истинности достаточно прост, если высказывание содержит небольшое число атомов. В том случае, когда высказывание имеет три атома, соответствующая таблица истинности состоит из $2^3 = 8$ строк. Таблица истинности высказывания, имеющего четыре атома, состоит уже из $2^4 = 16$ строк, а имеющего десять атомов, — из $2^{10} = 1024$ строк!

Кроме того, нужно иметь ввиду, что мы не можем использовать таблицы истинности в логике предикатов. Поэтому в следующих параграфах мы изучим более совершенные и более экономные методы определения истинностных значений высказываний и множеств высказываний. Эти методы будут составлять основу логического программирования.

§ 1.5. Логические следствия и интерпретации

Из таблицы истинности высказывания $(A \wedge B) \to C$ в примере 1.4.2 можно видеть, что это высказывание принимает значение t, если все три атома A, B, C принимают значения t. В этом случае говорят, что истинность $(A \wedge B) \to C$ следует из того факта, что каждое высказывание из $S = \{A, B, C\}$ имеет значение t (см. [Chur56]). Таким образом, введем следующее понятие.

Определение 1.5.1. Пусть S — множество высказываний. Высказывание σ называется *логическим следствием* S (этот факт

будет обозначаться $S \models \sigma$), если для каждого истинностного означивания V, обладающего свойством $V(\varphi) = t$ для любого $\varphi \in S$, мы можем заключить, что $V(\sigma) = t$.

Обозначим $Con(S) = \{\sigma\colon S \models \sigma\}$ множество всех логических следствий S. Формальное определение этого множества таково:

$$\sigma \in Con(S) \Leftrightarrow (\text{для каждого истинностного означивания } V)$$
$$(\text{для каждого } \varphi \in S \text{ верно } V(\varphi) = t) \Rightarrow (V(\sigma) = t).$$

В дальнейшем вместо выражения «для каждого $\varphi \in S$ верно $V(\varphi) = t$» мы будем использовать запись $V[S] = \{t\}$, или даже еще менее формально $V[S] = t$. □ 1.5.1

Замечание 1.5.2. Символы $\Leftrightarrow$ и $\Rightarrow$, используемые в данном выше определении в значениях «тогда и только тогда» и «влечет», являются символами *метаязыка*. Метаязык — это язык, используемый для рассуждений о формулах логики высказываний и для исследования их свойств. Когда, например, мы говорим, что $\models \varphi$, т. е. высказывание φ — тавтология, мы выражаем суждение о φ. Поэтому $\models \varphi$ является *метавысказыванием* для PL , а именно высказыванием метаязыка PL.

Как и язык логики высказываний, метаязык может быть формализован. Но для того, чтобы избежать чрезмерного и педантичного использования символов, например, $\rightarrow$ в языке и $\Rightarrow$ в метаязыке, в качестве метаязыка мы будем использовать тщательно и точно сформулированные предложения разговорного языка.

Завершим комментарий о метаязыке следующим примером. Пусть A — высказывание PL. Тогда выражение «$A \rightarrow A$» — это высказывание PL, а выражение «если A является высказыванием PL, то A является высказыванием PL» — это высказывание метаязыка PL. □ 1.5.2

Пример 1.5.3. Пусть $S = \{A \wedge B, B \rightarrow C\}$ — множество высказываний. Тогда C — логическое следствие S, т. е. $S \models C$.

Доказательство. Допустим, что V — истинностное означивание, которое обращает все высказывания S в истину:

$$V(A \wedge B) = t, \qquad V(B \rightarrow C) = t.$$

Тогда по определению 1.3.3 мы имеем:

$$V(A) \sqcap V(B) = t, \tag{1}$$

$$V(B) \rightsquigarrow V(C) = t. \tag{2}$$

Затем, из (1) и определения 1.3.2 следует, что

$$V(A) = V(B) = t. \tag{3}$$

и, следовательно, (2) преобразуется к виду

$$t \rightsquigarrow V(C) = t. \tag{4}$$

Из определения 1.3.2 и (4) мы можем заключить, что $V(C) = t$. Это означает, что каждое истинностное означивание, подтверждающее все высказывания из S, подтверждает также C. Таким образом, C является логическим следствием S, т. е. $S \models C$. □ 1.5.3

Обозначив $Taut$ множество всех тавтологий, докажем следствие.

Следствие 1.5.4. *$Con(\varnothing) = Taut$, где $\varnothing$ обозначает пустое множество.*

Доказательство. Рассмотрим $\sigma \in Taut$. Тогда для каждого истинностного означивания V имеем $V(\sigma) = t$, т. е. справедливо соотношение

$$\underline{V[\varnothing] = \{t\}} \Rightarrow V(\sigma) = t.$$

Следовательно, $\sigma \in Con(\varnothing)$.

Верно и обратное. Рассмотрим $\sigma \in Con(\varnothing)$. Тогда для каждого истинностного означивания V имеем:

$$\text{если для каждого } \underline{\varphi \in \varnothing} \text{ справедливо } V(\varphi) = t, \text{ то } V(\sigma) = t. \tag{1}$$

Но в $\varnothing$ нет элементов. Поэтому мы можем записать:

$$\begin{gathered}\text{если для каждого } \underline{\varphi \in \varnothing} \text{ верно,} \\ \text{что } V(\varphi) = t \text{ или } V(\varphi) = f, \text{то } V(\sigma) = t.\end{gathered} \tag{2}$$

То есть $V(\sigma) = t$ независимо от значения $\varphi \in \varnothing$. Это означает, что для всех истинностных означиваний V верно, что $V(\sigma) = t$. Следовательно, σ является тавтологией. □ 1.5.4

Таким образом, тавтологии не являются следствием какого-либо специального множества высказываний[1]), они не зависимы ни от какого множества высказываний и связаны только с таблицами истинности, определенными в § 1.3.

Подчеркнутые части в доказательстве следствия 1.5.4 отмечают технические моменты доказательства. В пустом множестве нет элементов. Как же мы можем записать импликацию с подчеркнутой частью? Давайте вспомним третью и четвертую строки определения таблицы истинности высказывания $A \to B$ (см. стр. 25). Если A принимает значение f и B принимает значение t или f, то $A \to B$ равно t. Это значит, что если A есть ложь, то импликация $A \to B$ всегда истинна! Именно этим свойством мы и воспользовались в метаязыке, в котором доказываются наши теоремы. В (1), например,

[1]) Это означает, что тавтологии логически следуют из любого множества высказываний S вне зависимости от его состава. — *Прим. перев.*

утверждение «$\varphi \in \varnothing$» не может быть истинно, так как в $\varnothing$ нет элементов. Независимо от того, является ли утверждение «$V(\varphi) = t$» истинным или нет, (1) истинно!

Как упоминалось выше, PL изучает множества высказываний. Следующие два определения касаются выполнимости множества высказываний.

Определение 1.5.5. Множество высказываний S (семантически) *непротиворечиво*, если существует истинностное означивание, которое подтверждает каждое высказывание из S. Формально:

$$\text{непротиворечиво}(S) \Leftrightarrow (\text{существует означивание } V)\ [(\text{для каждого } \sigma \in S)\ (V(\sigma) = t)].$$

Тот факт, что S непротиворечиво, обозначается $V(S) = t$.

Непротиворечивое множество S мы будем называть также *подтверждаемым*, или *выполнимым*. Про истинностное означивание V, которое подтверждает каждое высказывание непротиворечивого множества S, говорят, что оно *подтверждает* S.

Соответственно, S *противоречиво*, *неподтверждаемо*, или *невыполнимо*, если каждое истинностное означивание не подтверждает по меньшей мере одно высказывание из S. Формально:

противоречиво $(S) \Rightarrow$

(для каждого означивания V) [(существует $\sigma \in S$) $(V(\sigma) = f)$].

□ 1.5.5

Пример 1.5.6. Множество высказываний $S = \{A \wedge \neg B, A \to B\}$ противоречиво.

Доказательство. Предположим, существует такое истинностное означивание V, что для каждого $C \in S$ верно $V(C) = t$. Тогда

$$V(A \wedge \neg B) = t, \qquad V(A \to B) = t,$$

что означает

$$V(A) \sqcap V(\neg B) = t, \tag{1}$$

$$V(A) \rightsquigarrow V(B) = t. \tag{2}$$

Из (1) и определения 1.3.2 следует, что $V(A) = V(\neg B) = t$, и

$$V(A) = t, \qquad V(B) = f.$$

Тогда из (2) следует, что $t \rightsquigarrow f = t$, что противоречит определению 1.3.2, в котором $t \rightsquigarrow f = f$. Следовательно, никакое истинностное означивание не может подтвердить все высказывания S, т. е. S противоречиво. □ 1.5.6

Определение 1.5.7. Истинностное означивание, подтверждающее множество высказываний S, называется *интерпретацией* S. Множество всех интерпретаций S обозначается $Int(S)$.

$Int(S) = \{V\colon V$ — истинностное означивание и для каждого $\sigma \in S$ верно $V(\sigma) = t\}$. □ 1.5.7

Докажем некоторые полезные свойства логических следствий и интерпретаций.

Следствие 1.5.8. *Для любых множеств высказываний* S, S_1, S_2 *верно:*

1) $S_1 \subseteq S_2 \Rightarrow Con(S_1) \subseteq Con(S_2)$;

2) $S \subseteq Con(S)$;

3) $Taut \subseteq Con(S)$ *для любого* S;

4) $Con(S) = Con(Con(S))$;

5) $S_1 \subseteq S_2 \Rightarrow Int(S_2) \subseteq Int(S_1)$;

6) $Con(S) = \{\sigma\colon V(\sigma) = t$ *для каждого* $V \in Int(S)\}$;

7) $\sigma \in Con(\{\sigma_1, \ldots, \sigma_n\}) \Leftrightarrow \sigma_1 \to (\sigma_2 \to (\ldots(\sigma_n \to \sigma)\ldots)) \in Taut$.

Доказательство. 1. Предположим, что $\sigma \in Con(S_1)$. Пусть V — такое истинностное означивание, что для каждого $\varphi \in S_2$ справедливо $V(\varphi) = t$. Таким образом, верно, что $V(\varphi) = t$ для каждого $\varphi \in S_1$ (так как $S_1 \subseteq S_2$). Тогда $V(\sigma) = t$, поскольку $\sigma \in Con(S_1)$. Следовательно, $Con(S_1) \subseteq Con(S_2)$.

2. Если $\sigma \in S$, то каждое истинностное означивание V, подтверждающее все высказывания S, также подтверждает σ. Таким образом, $\sigma \in Con(S)$. Но тогда $S \subseteq Con(S)$.

3. Предположим, что $\sigma \in Taut$. Пусть V — такое истинностное означивание, что для каждого $\varphi \in S$ верно $V(\varphi) = t$. Тогда $V(\sigma) = t$, и поэтому $\sigma \in Con(S)$. Следовательно, $Taut \subseteq Con(S)$.

4. Согласно пункту 2 мы имеем $S \subseteq Con(S)$, а согласно пункту 1 — $Con(S) \subseteq Con(Con(S))$. Остается только доказать, что

$$Con(Con(S)) \subseteq Con(S).$$

Предположим, что $\sigma \in Con(Con(S))$. Пусть V — истинностное означивание, такое, что для каждого $\varphi \in S$ имеет место $V(\varphi) = t$. Тогда для каждого $\tau \in Con(S)$ верно, что $V(\tau) = t$. Из определения $Con(S)$ следует, что $V(\sigma) = t$. Это означает, что $\sigma \in Con(S)$. Следовательно, $Con(Con(S)) \subseteq Con(S)$.

5. Если $V \in Int(S_2)$, то для каждого $\sigma \in S_2$ справедливо $V(\sigma) = t$. Ввиду того, что $S_1 \subseteq S_2$, для каждого $\sigma \in S_1$ верно, что $V(\sigma) = t$ и, следовательно, $V \in Int(S_1)$.

6. Если $\sigma \in Con(S)$, то для каждого $V \in Int(S)$ верно, что $V(\sigma) = t$. Тогда

$$\sigma \in \{\varphi\colon V(\varphi) = t \text{ для каждого } V \in Int(S)\}.$$

Более того, если

$$\varphi \in \{\sigma\colon V(\sigma) = t \text{ для каждого } V \in Int(S)\},$$

то, очевидно, $\varphi \in Con(S)$.

7. ($\Rightarrow$) Предположим, что $\varphi \in Con(\{\sigma_1, \ldots, \sigma_n\})$. Пусть V — истинностное означивание. Возможны два случая:

а) для каждого σ_i, $1 \leqslant i \leqslant n$, $V(\sigma_i) = t$ и

б) хотя бы для одного σ_j, $1 \leqslant j \leqslant n$, $V(\sigma_j) = f$.

Проанализируем каждый случай в отдельности.

а) Для каждого σ_i, где $1 \leqslant i \leqslant n$, верно, что

$$V(\sigma_i) = t \text{ и } \varphi \in Con(\{\sigma_1, \ldots, \sigma_n\}),$$

поэтому $V(\varphi) = t$. Следовательно,

$$V(\sigma_n \to \varphi) = V(\sigma_n) \rightsquigarrow V(\varphi) = t \rightsquigarrow t = t.$$

Если $V(\sigma_k \to (\sigma_{k+1} \to \ldots(\sigma_n \to \varphi)\ldots)) = t$, то

$$V(\sigma_{k-1} \to (\sigma_k \to (\sigma_{k+1} \to \ldots(\sigma_n \to \varphi)\ldots))) =$$
$$= V(\sigma_{k-1}) \rightsquigarrow V(\sigma_k \to (\sigma_{k+1} \to \ldots(\sigma_n \to \varphi)\ldots)) = t \rightsquigarrow t = t.$$

Следовательно, (по индукции) $\sigma_1 \to (\sigma_2 \to (\ldots(\sigma_n \to \sigma)\ldots)) = t$.

б) Пусть r — наименьшее натуральное число, для которого верно $V(\sigma_r) = f$. Тогда

$$V(\sigma_r \to (\sigma_{r+1} \to \ldots(\sigma_n \to \varphi)\ldots)) =$$
$$= V(\sigma_r) \rightsquigarrow V(\sigma_{r+1} \to \ldots(\sigma_n \to \varphi)\ldots) =$$
$$= f \rightsquigarrow V(\sigma_{r+1} \to \ldots(\sigma_n \to \varphi)\ldots) = t.$$

Поскольку $V(\sigma_{r-1}) = t$, то

$$V(\sigma_{r-1} \to (\sigma_r \to \ldots(\sigma_n \to \varphi)\ldots)) =$$
$$= V(\sigma_{r-1}) \rightsquigarrow V(\sigma_r \to \ldots(\sigma_n \to \varphi)\ldots) = t \rightsquigarrow t = t.$$

Если $V(\sigma_k \to (\sigma_{k+1} \to \ldots(\sigma_n \to \varphi)\ldots)) = t$, где $k \leqslant r-1$, то

$$V(\sigma_{k-1} \to (\sigma_k \to (\sigma_{k+1} \to \ldots(\sigma_n \to \varphi)\ldots))) =$$
$$= V(\sigma_{k-1}) \rightsquigarrow V(\sigma_k \to (\sigma_{k+1} \to \ldots(\sigma_n \to \varphi)\ldots)) = t \rightsquigarrow t = t.$$

Следовательно, $\sigma_1 \to (\sigma_2 \to (\ldots(\sigma_n \to \sigma)\ldots)) = t$.

Как в случае а), так и в случае б) для любого истинностного означивания V справедливо соотношение $\sigma_1 \to (\sigma_2 \to (\ldots(\sigma_n \to \sigma)\ldots)) = t$. Следовательно, $\sigma_1 \to (\sigma_2 \to (\ldots(\sigma_n \to \sigma)\ldots))$ — тавтология.
($\Leftarrow$) Предположим, что $\sigma_1 \to (\sigma_2 \to (\ldots(\sigma_n \to \sigma)\ldots))$ — тавтология. Пусть V — истинностное означивание, такое, что $V(\sigma_i) = t$ для каждого $i \in \{1, 2, \ldots, n\}$. Если мы предположим, что $V(\varphi) = f$, то

$$V(\sigma_n \to \varphi) = V(\sigma_n) \rightsquigarrow V(\varphi) = t \rightsquigarrow f = f.$$

Поэтому, если $V(\sigma_k \to (\sigma_{k+1} \to \ldots(\sigma_n \to \varphi)\ldots)) = t$, то

$$V(\sigma_{k-1} \to (\sigma_k \to (\sigma_{k+1} \to \ldots(\sigma_n \to \varphi)\ldots))) =$$
$$= V(\sigma_{k-1}) \rightsquigarrow V(\sigma_k \to (\sigma_{k+1} \to \ldots(\sigma_n \to \varphi)\ldots)) = t \rightsquigarrow f = f.$$

Следовательно, $V(\sigma_1 \to (\sigma_2 \to (\ldots(\sigma_n \to \sigma)\ldots))) = f$ вопреки тому, что $\sigma_1 \to (\sigma_2 \to (\ldots(\sigma_n \to \sigma)\ldots))$ — тавтология. □ 1.5.8

Следствие 1.5.8 дает нам метод, позволяющий для любой заданной формулы φ установить, является ли она логическим следствием заданного конечного множества высказываний S: мы проверим за 2^n шагов, является ли правая часть пункта 7 тавтологией. Однако для бесконечного множества высказываний S этот метод потребует бесконечного числа шагов. В этом случае более уместно воспользоваться семантическими таблицами.

§ 1.6. Полнота множеств логических связок и нормальные формы

Нахождение истинностного значения высказывания, доказательство противоречивости или непротиворечивости множества высказываний в большинстве случаев зависит от числа и типа связок, содержащихся в высказываниях. Используемые нами пять логических связок наиболее часто встречаются в математических текстах. В этом разделе мы докажем, что произвольное множество логических связок выразимо через множество $\{\neg, \wedge, \vee\}$, и, следовательно, множества логических связок $\{\neg, \wedge, \vee\}$ достаточно, чтобы выразить каждое высказывание логики высказываний (см. [Smu68, Mend64, Schm60]).

Определение 1.6.1. Множество логических связок Q называется *полным*, если для каждого высказывания логики высказываний существует логически эквивалентное ему высказывание, содержащее только связки из Q. □ 1.6.1

Теорема 1.6.2. *Множество* $\{\neg, \wedge, \vee\}$ *является полным.*

Доказательство. Пусть $\sigma = \sigma(A_1, A_2, \ldots, A_k)$ — высказывание, в котором встречаются только атомы $A_1, A_2, \ldots, A_k$. Построим сокращенную таблицу истинности σ:

A_1	...	A_λ	...	A_k	$\sigma(A_1, A_2, \ldots, A_k)$
σ_{l_1}	...	σ_{l_λ}	...	σ_{l_k}	σ_l
...		...		...	...
σ_{ν_1}	...	σ_{ν_λ}	...	σ_{ν_k}	σ_ν
...		...		...	...
$\sigma_{2^k_1}$	...	$\sigma_{2^k_\lambda}$	...	$\sigma_{2^k_k}$	σ_{2^k}

В этой таблице σ_{n_l} обозначает соответствующее истинностное значение в строке n и столбце l, а σ_n обозначает истинностное значение высказывания $\sigma(A_1, \ldots, A_k)$ в строке n.

1. Предположим, что в последнем столбце найдется хотя бы одно значение t. Докажем, что существует высказывание, содержащее только связки $\neg, \wedge, \vee$, последний столбец сокращенной таблицы истинности которого совпадает с последним столбцом указанной выше таблицы.

Для произвольного атома A обозначим A^t сам атом A и A^f — его отрицание $\neg A$. По строке n построим конъюнкцию

$$t_n\colon (A_1^{\sigma_{\nu_1}} \wedge \ldots \wedge A_k^{\sigma_{\nu_k}}),$$

которая содержит только связки $\neg$ и $\wedge$. Тогда высказывание

$$t_{l_1} \vee \ldots \vee t_{l_m}$$

является искомым, так как его сокращенная таблица истинности содержит t в точности в тех же строках, что и сокращенная таблица истинности $\sigma(A_1, \ldots, A_k)$.

2. Если последний столбец не содержит t, то высказывание ложно и, следовательно, логически эквивалентно $A \wedge \neg A$, где A — произвольный пропозициональный символ. □ 1.6.2

Используя технику доказательства теоремы 1.6.2, мы можем выразить произвольное высказывание через связки $\neg$, $\wedge$ и $\vee$. Полученное в результате эквивалентное высказывание называется *дизъюнктивной нормальной формой* (ДНФ).

Пусть F — высказывание, содержащее атомы $A_1, \ldots, A_n$. В общем случае ДНФ формулы F имеет вид

$$\text{ДНФ}(F)\colon (A_{1_1} \wedge \ldots \wedge A_{1_n}) \vee (A_{2_1} \wedge \ldots \wedge A_{2_n}) \vee \ldots \vee (A_{k_1} \wedge \ldots \wedge A_{k_n}),$$

где $A_{i_j} \in \{A_1, \ldots, A_n\}$ или $A_{i_j} \in \{\neg A_1, \ldots, \neg A_n\}$, т. е. A_{i_j} — атомы или отрицание атомов F, и в каждой конъюнктивной компоненте

ДНФ(F) встречается каждый атом F либо с отрицанием, либо без него[1]).

Конъюнктивная нормальная форма (КНФ), двойственная к ДНФ, имеет вид

$$\text{КНФ}(F)\colon (A_{1_1} \vee \ldots \vee A_{1_n}) \wedge (A_{2_1} \vee \ldots \vee A_{2_n}) \wedge \ldots \wedge (A_{k_1} \vee \ldots \vee A_{k_n}).$$

Пример 1.6.3. Найдем ДНФ высказывания F, заданного сокращенной таблицей истинности:

A	B	C	F
t	t	t	t
t	t	f	f
t	f	t	f
t	f	f	f
f	t	t	t
f	t	f	f
f	f	t	f
f	f	f	t

Применим метод, описанный в доказательстве теоремы 1.6.2.

Шаг 1. Находим все строки, содержащие t в последнем столбце. В нашем случае это строки с номерами 1, 5, 8.

Шаг 2. $\text{ДНФ}(F) \equiv t_1 \vee t_5 \vee t_8 \equiv (A \wedge B \wedge C) \vee (\neg A \wedge B \wedge C) \vee (\neg A \wedge \wedge \neg B \wedge \neg C)$.

□ 1.6.3

Теперь докажем два следствия, устанавливающих полноту конкретных множеств логических связок.

Следствие 1.6.4. *Множества связок* $K_1 = \{\neg, \vee\}$ *и* $K_2 = \{\neg, \wedge\}$ *являются полными.*

[1]) Дизъюнктивная нормальная форма, каждый конъюнктивный сомножитель которой содержит все пропозициональные символы, называется *совершенной* ДНФ. Как будет видно в дальнейшем, в общем случае не требуется, чтобы каждый атом входил в состав конъюнктивного сомножителя ДНФ. Аналогичное замечание относится и к устройству конъюнктивной нормальной формы (КНФ) — *Прим. перев.*

Доказательство. Используя таблицы истинности, мы можем доказать, что

$$A \to B \equiv \neg A \vee B, \qquad A \leftrightarrow B \equiv (A \to B) \wedge (B \to A),$$

$$A \wedge B \equiv \neg(\neg A \vee \neg B).$$

Следовательно, каждое высказывание логики высказываний может быть выражено при помощи связок из множества $\{\neg, \vee\}$. Таким образом, K_1 полно.

Аналогично мы можем доказать, что K_2 полно.

$$A \to B \equiv \neg A \vee B, \qquad A \leftrightarrow B \equiv (A \to B) \wedge (B \to A),$$

$$A \vee B \equiv \neg(\neg A \wedge \neg B). \qquad \square\ 1.6.4$$

Пример 1.6.5. Наряду со связками $\neg$, $\wedge$, $\vee$, $\to$ и $\leftrightarrow$ мы можем ввести другие связки, например, $|$ и $:$, определенные следующими таблицами истинности:

A	B	$A\|B$
t	t	f
t	f	t
f	t	t
f	f	t

A	B	$A:B$
t	t	f
t	f	f
f	t	f
f	f	t

$\square$ 1.6.5

Следствие 1.6.6. *Множества связок* $\Sigma_1 = \{|\}$ *и* $\Sigma_2 = \{:\}$ *полны.*

Доказательство. 1. Докажем полноту Σ_1. В сущности, $A|B$ обозначает, что A и B не могут быть одновременно истинными. Тогда $A|B \equiv \neg(A \wedge B)$. (Эту эквивалентность легко проверить при помощи таблиц истинности.) Соотношения $A|A \equiv \neg(A \wedge A) \equiv \neg A$ показывают, что отрицание может быть выражено при помощи связки $|$. Для конъюнкции верно, что $A \wedge B \equiv \neg\neg(A \wedge B) \equiv \neg(A|B) \equiv \equiv (A|B)|(A|B)$.

2. Докажем полноту Σ_2. Мы можем выразить отрицание: $\neg A \equiv \equiv \neg(A \vee A) \equiv A : A$, и конъюнкцию: $A \wedge B \equiv \neg(\neg A \vee \neg B) \equiv \equiv (A : A) : (B : B)$. $\square$ 1.6.6

Заметим, что множествами $\{|\}$ и $\{:\}$ исчерпываются все одноэлементные полные множества связок (см. упражнение 20). Кроме того, каждое множество логических связок с двумя элементами, один из которых — $\neg$, а другой — $\wedge$, $\vee$ или $\to$, является полным (см. [Schm60, Smul68, Mend64]).

Замечание 1.6.7. Преобразование высказывания σ в высказывание φ, логически эквивалентное ему, но содержащее другие связки, часто бывает полезным. Например, в методе резолюций, который исследуется в следующих параграфах, мы выражаем импликации, т. е. высказывания вида $\sigma \to \varphi$, через связки $\neg$, $\wedge$ и $\vee$. При этом мы используем эквивалентность

$$A \to B \equiv \neg A \vee B,$$

проверить которую можно при помощи таблиц истинности.

□ 1.6.7

В последующих параграфах мы опишем используемые в логике высказываний методы доказательств. Их описание упростит дальнейший переход к логическому программированию.

§ 1.7. Семантические таблицы

Методы доказательства — это алгоритмические процедуры, следуя которым мы можем устанавливать, является ли высказывание тавтологией, и выполнимо ли множество высказываний. Эти методы разрабатываются в теории автоматического доказательства теорем и составляют основу логического программирования.

Первый из описанных методов алгоритмического доказательства использует семантические таблицы. Генцен (1909–1945) в 1934 г. впервые доказал, что все тавтологии могут быть получены применением некоторых правил, то есть существует некоторая процедура проверки тавтологичности формул (см. [Klee52, Rault79]). Используя теорию доказательств Бет и Хинтикка в 1955 г. построили алгоритм, устанавливающий, является высказывание тавтологией или нет.

При помощи семантических таблиц Бета (далее мы будем их называть просто семантическими таблицами) можно исследовать возможность того, что данное высказывание принимает значение t или значение f.

Семантическая таблица составного высказывания K строится индуктивно, исходя из семантических таблиц высказываний, составляющих K. Таким образом, мы определим атомарные семантические таблицы (см. [Smul68, Fitt90]).

Определение 1.7.1. 1. Пусть σ — высказывание. Обозначим через $f\sigma$ утверждение «σ ложно», а через $t\sigma$ — утверждение «σ истинно». При этом $t\sigma$ и $f\sigma$ называются *помеченными формулами*.

2. Атомарные семантические таблицы для высказываний A, σ_1, σ_2 и высказываний, составленных из A, σ_1 и σ_2, представлены в следующей таблице.

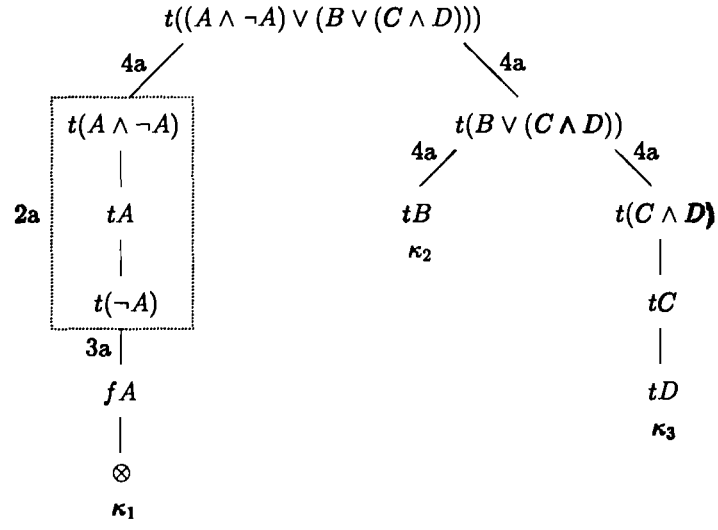
1*a* tA	1*б* fA	2*a* $t(\sigma_1 \wedge \sigma_2)$ $t\sigma_1$ $t\sigma_2$	2*б* $f(\sigma_1 \wedge \sigma_2)$ $f\sigma_1$ $f\sigma_2$
3*a* $t(\neg\sigma)$ $f\sigma$	3*б* $f(\neg\sigma)$ $t\sigma$	4*a* $t(\sigma_1 \vee \sigma_2)$ $t\sigma_1$ $t\sigma_2$	4*б* $f(\sigma_1 \vee \sigma_2)$ $f\sigma_1$ $f\sigma_2$
5*a* $t(\sigma_1 \to \sigma_2)$ $f\sigma_1$ $t\sigma_2$	5*б* $f(\sigma_1 \to \sigma_2)$ $t\sigma_1$ $f\sigma_2$	6*a* $t(\sigma_1 \leftrightarrow \sigma_2)$ $t\sigma_1$ $f\sigma_1$ $t\sigma_2$ $f\sigma_2$	6*б* $f(\sigma_1 \leftrightarrow \sigma_2)$ $t\sigma_1$ $f\sigma_1$ $f\sigma_2$ $t\sigma_2$

Интуитивно, tA (или fA) можно рассматривать как утверждение о том, что A истинно (ложно). Как следует из атомарной семантической таблицы 4а, истинность утверждения $\sigma_1 \vee \sigma_2$ требует истинности σ_1 или истинности σ_2 (т. е. имеет место ветвление), в то время как из атомарной семантической таблицы 5б следует, что ложность утверждения $\sigma_1 \to \sigma_2$ требует истинности σ_1 и ложности σ_2 (т. е. в данном случае мы имеем последовательное соединение). Таким образом, в семантических таблицах ветвление обозначает дизъюнкцию, а последовательное соединение — конъюнкцию утверждений. □ 1.7.1

Построение семантической таблицы составного высказывания K мы начинаем с того, что записываем помеченную формулу tK или fK в корень семантической таблицы. Затем мы разворачиваем семантическую таблицу K в соответствии с определением 1.7.1.

Прежде, чем определить общие правила построения семантических таблиц, рассмотрим пример.

Пример 1.7.2. Пусть $K \equiv (A \wedge \neg A) \vee (B \vee (C \wedge D))$. В K присутствуют атомы A, B, C и D. Построим семантическую таблицу с корнем tK.



Это замкнутая семантическая таблица, содержащая три ветви, а именно, три последовательности $\varkappa_1$, $\varkappa_2$ и $\varkappa_3$. Ветви произрастают из корня. Левая ветвь $\varkappa_1$, противоречива, так как она содержит противоречащие друг другу помеченные формулы tA и fA. Противоречивая ветвь помечается внизу символом $\otimes$. Две другие ветви не являются противоречивыми.

Из семантической таблицы K мы можем заключить, что гипотеза tK правильна при некоторых условиях, например, при выполнении tB (ветвь $\varkappa_2$) или выполнении tC и tD (ветвь $\varkappa_3$). Однако иногда она ложна.

Несложно выделить из семантической таблицы K составляющие ее атомарные семантические таблицы. Например, ограниченная прямоугольником из точек семантическая таблица является атомарной семантической таблицей 2а. □ 1.7.2

Теперь мы сформулируем понятия, необходимые для построения семантических таблиц.

Определение 1.7.3. 1. *Вершинами* семантической таблицы называются все помеченные формулы, встречающиеся в этой таблице.

2. Вершина семантической таблицы называется *особой*, если она встречается как корень некоторой атомарной семантической таблицы. В противном случае, вершина называется *обычной*.

3. Ветвь семантической таблицы называется *противоречивой*, если для некоторого высказывания σ помеченные формулы $t\sigma$ и $f\sigma$ являются вершинами этой ветви.

4. Семантическая таблица называется *замкнутой*, если каждая непротиворечивая ее ветвь не содержит обычных вершин. В противном случае, семантическая таблица называется *незамкнутой*.

5. Семантическая таблица *противоречива*, если каждая ее ветвь противоречива. □ 1.7.3

Определение 1.7.4 (индуктивное определение семантической таблицы). Построим *семантическую таблицу* высказывания K следующим образом.

Вначале помещаем помеченную формулу tK (или fK) в корень. Затем продолжаем построение по индукции.

Шаг n. Пусть мы уже построили семантическую таблицу T_n, $n \geqslant 1$.

Шаг $n+1$. Расширим семантическую таблицу T_n до семантической таблицы T_{n+1}. При этом мы пользуемся некоторой вершиной семантической таблицы T_n, которую в дальнейшем не будем использовать. Из всех обычных вершин T_n, ближайших к корню, выбираем самую левую. Обозначим выбранную вершину X.

К концу каждой непротиворечивой ветви семантической таблицы T_n мы присоединяем атомарную семантическую таблицу, имеющую корнем X. (При этом вершина X становится особой вершиной.) В результате получаем семантическую таблицу T_{n+1} (как правило, мы не записываем саму вершину X, так как она уже присутствует в каждой из рассматриваемых непротиворечивых ветвей).

Построение заканчивается, если каждая непротиворечивая ветвь не содержит обычных вершин. □ 1.7.4

Поясним описанную методику на примере построения противоречивой семантической таблицы.

Пример 1.7.5. Рассмотрим закон Пирса $((A \to B) \to A) \to A$.

1. В корень таблицы мы помещаем утверждение о том, что высказывание $((A \to B) \to A) \to A$ ложно. (См. рис. 1.1) Предположение о ложности закона Пирса приводит нас к построению противоречивой семантической таблицы. Следовательно, соответствующая формула истинна.

2. Если бы мы поместили в корень семантической таблицы утверждения о том, что закон Пирса истинен, то заключение осталось бы таким же. (См. рис. 1.2.) Заметим, что в этой семантической таблице нет противоречивых ветвей. Для каждой из трех альтернатив fA, или tB и fA, или tA утверждение $((A \to B) \to A) \to A)$

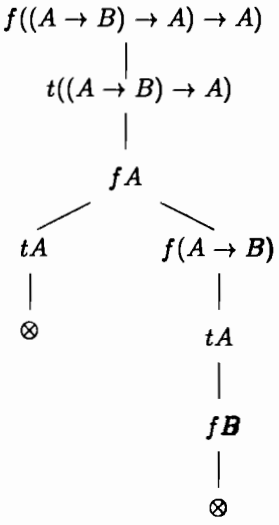


Рис 1.1

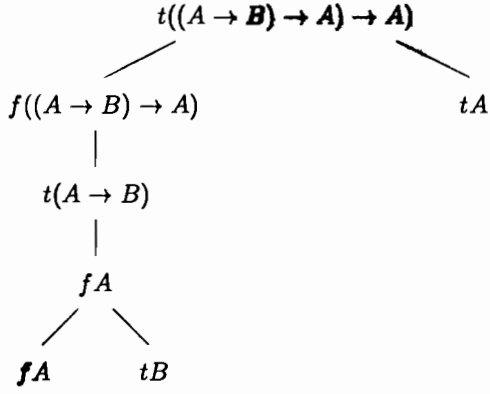


Рис. 1.2

истинно. Даже в случае fB одно из утверждений tA или fA имеет место. Следовательно, закон Пирса логически истинен. □ 1.7.5

Итак: *если замкнутая семантическая таблица с fK в корне противоречива (а это означает, что мы пытались всеми возможными способами сделать высказывание K ложным и не сумели), то K — тавтология.*

В следующем определении 1.7.6 этот замысел представлен более формально.

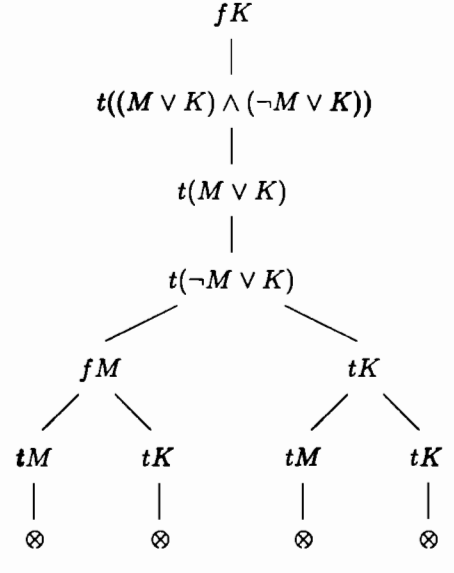
Определение 1.7.6. *Доказательством*, или *выводом по Бету* высказывания K называется замкнутая противоречивая семантическая таблица, в корне которой помещена помеченная формула fK. Замкнутая противоречивая таблица, имеющая в качестве корня tK называется *опровержением по Бету* высказывания K.

Говорят, что высказывание K *доказуемо*, или *выводимо по Бету*, если оно имеет *доказательство по Бету*. Высказывание K называется *опровержимым по Бету*, если существует *опровержение K по Бету*. Тот факт, что K доказуемо по Бету, обозначается $\vdash_B K$. □ 1.7.6

Как мы докажем в теореме 1.10.7 и теореме 1.10.9, каждая тавтология доказуема по Бету (полнота доказательств по Бету) и, наоборот, каждое доказуемое по Бету высказывание есть тавтология (корректность доказательств по Бету).

Пример 1.7.7. Предположим, что истинны высказывания:
1: Джордж любит Марию, или Джордж любит Екатерину;
2: Если Джордж любит Марию, то он любит Екатерину.
Кого же Джордж любит?

Обозначим символом M высказывание «Джордж любит Марию» и символом K высказывание «Джордж любит Екатерину». Тогда высказывание 1 эквивалентно $M \vee K$, а высказывание 2 эквивалентно $M \to K$ и, следовательно, $\neg M \vee K$. Конъюнкцию высказываний 1 и 2, то есть $(M \vee K) \wedge (\neg M \vee K)$, обозначим A. По условию примера предполагается, что A истинно. Мы хотим узнать, любит ли Джордж Екатерину, или, говоря формально, верно ли tK. Предположим, что он ее не любит, то есть верно fK. Мы можем построить семантическую таблицу с корнем fK.



На шаге 2 мы присоединили $t((M \vee K) \wedge (\neg M \vee K))$, так как 1 и 2 даны истинными. Начав построение таблицы с корнем fK, мы получили противоречивую таблицу. Это означает, что высказывание K всегда истинно, другими словами, Джордж любит Екатерину!

Если бы мы построили семантическую таблицу с помеченной формулой fM в корне, то получили бы непротиворечивую таблицу, и, следовательно, не могли бы заключить, любит Джордж Марию или нет! □ 1.7.7

§ 1.8. Аксиоматическая система вывода

Логика высказываний, подобно другим математическим системам, может быть представлена как аксиоматическая система с логическими аксиомами и правилами вывода. Аксиомы — это некоторые тавтологии, правило вывода R *выводит* высказывание σ из последовательности высказываний $\sigma_1, \sigma_2, \ldots, \sigma_n$. Мы опишем вкратце аксиоматическое представление логики высказываний (см. [Schm60, RaSi70, Mend64]).

Определение 1.8.1 (аксиомы). Каждое высказывание следующего вида является аксиомой.

1. $\varphi \to (\tau \to \varphi)$.
2. $(\varphi \to (\tau \to \sigma)) \to ((\varphi \to \tau) \to (\varphi \to \sigma))$.
3. $(\neg\varphi \to \neg\tau) \to (\tau \to \varphi)$.

Заметим, что высказывания φ, τ и σ могут быть произвольными. Таким образом, мы описали схемы аксиом, из которых можно получить неограниченное число аксиом. Легко проверить, что все эти аксиомы являются правильно построенными формулами логики высказываний и, конечно, тавтологиями. □ 1.8.1

Определение 1.8.2 (правило вывода Modus Ponens). Мы будем использовать только одно правило вывода, правило Modus Ponens, которое утверждает, что высказывание τ выводится из высказываний φ и $\varphi \to \tau$. Правило Modus Ponens[1]) своим названием обязано Диогену Лаэртскому (Diogenes Laertius), см. [Zeno]. Оно обозначается следующим образом:

$$\frac{\varphi \qquad \varphi \to \tau}{\tau} \tag{1}$$

или

$$\varphi, \varphi \to \tau \vdash \tau \tag{2}$$

□ 1.8.2

В (1), типичном определении логического правила, горизонтальная линия отделяет гипотезы от заключения. В (2) символ $\vdash$ обозначает выводимость в аксиоматической системе. Мы рассматриваем три аксиомы определения 1.8.1 как формулы, выводимые в аксиоматической системе. Новые высказывания получаются при помощи этих трех аксиом и правила Modus Ponens. В следующем примере 1.8.3 демонстрируется, как можно применить аксиомы и правило Modus Ponens для вывода формулы логики высказываний $A \to A$.

[1]) В русской математической литературе это правило называют *правилом заключения*. — *Прим. перев.*

Пример 1.8.3. Докажем, что $\vdash A \to A$.

Доказательство. Сначала возьмем первую аксиому

$$\vdash A \to ((B \to A) \to A). \tag{1}$$

Вторую аксиому запишем в виде

$$\vdash (A \to ((B \to A) \to A) \to (((A \to (B \to A)) \to (A \to A)). \tag{2}$$

Из (1) и (2) по правилу Modus Ponens получаем, что

$$\vdash (A \to (B \to A)) \to (A \to A). \tag{3}$$

Используя вновь первую аксиому $A \to (B \to A)$, по правилу Modus Ponens заключаем, что $\vdash A \to A$. Следовательно, высказывание $A \to A$ выводимо в нашей аксиоматической системе. □ 1.8.3

Следующая теорема 1.8.4 позволяет производить в высказываниях замену одних подформул на другие, логически им эквивалентные. Доказательство теоремы можно найти в упражнении 22.

Теорема 1.8.4 (подстановка эквивалентных формул). *Пусть высказывание $\sigma \leftrightarrow \sigma_1$ выводимо в PL, σ — подформула высказывания φ, и высказывание φ_1 получено в результате подстановки в высказывание φ вместо некоторых вхождений формулы σ эквивалентной ей формулы σ_1. Тогда высказывание $\varphi \leftrightarrow \varphi_1$ также выводимо в PL. Формально записываем:*

$$\vdash \sigma \leftrightarrow \sigma_1 \Rightarrow \vdash \varphi \leftrightarrow \varphi_1. \qquad \square\ 1.8.4$$

Теперь мы приведем формальное определение доказательства высказывания аксиоматическим методом.

Определение 1.8.5. Пусть S — множество высказываний.

1. *Доказательством*, или *выводом* из S называется такая конечная последовательность высказываний $\sigma_1, \sigma_2, \ldots, \sigma_n$, что для каждого $1 \leqslant i \leqslant n$ верно:

а) σ_i принадлежит S, или

б) σ_i — аксиома, или

в) σ_i получено из σ_j, σ_k, где $1 \leqslant j, k \leqslant i$, по правилу Modus Ponens.

2. Высказывание σ называется *доказуемым*, или *выводимым из множества высказываний* S, если существует такое доказательство $\sigma_1, \sigma_2, \ldots, \sigma_n$ из S, что σ_n совпадает с σ. Для обозначения выводимости высказывания σ из множества высказываний S будем использовать запись $S \vdash \sigma$.

3. Высказывание σ называется *доказуемым*, или *выводимым*, если $\vdash \sigma$, то есть σ выводимо в аксиоматической системе определения 1.8.1 при помощи правила Modus Ponens. Отметим, что если

$S = \varnothing$, то понятие выводимого из множества S высказывания совпадает с понятием выводимого высказывания. □ 1.8.5

Пример 1.8.6. Выведем $\neg B \to (C \to A)$ из $S = \{A\}$. Имеем:

1) A — $A \in S$;

2) $A \to (C \to A)$ — аксиома 1;

3) $C \to A$ — Modus Ponens к 1 и 2;

4) $(C \to A) \to (\neg B \to (C \to A))$ — аксиома 1;

5) $\neg B \to (C \to A)$ — Modus Ponens к 3 и 4. □ 1.8.6

Заметим, что если σ выводимо из S и S — бесконечное множество, то σ выводимо из некоторого конечного подмножества S, так как доказательство всегда конечно.

Следующая теорема 1.8.7 является основной в теории доказательств.

Теорема 1.8.7 (теорема дедукции). *Пусть S — множество высказываний; K, L — высказывания. Тогда*

$$S \cup \{K\} \vdash L \quad \Leftrightarrow \quad S \vdash K \to L.$$

Доказательство. ($\Leftarrow$) Предположим $S \vdash K \to L$. Рассмотрим доказательство $\sigma_1, \sigma_2, \ldots, \sigma_n$ формулы $K \to L$, в котором σ_n есть сама формула $K \to L$, и для каждого $i \in \{1, \ldots, n\}$ либо σ_i — аксиома, либо $\sigma_i \in S$, либо σ_i выводится из двух предшествующих высказываний с использованием правила Modus Ponens. Тогда последовательность $\sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_{n+1}, \sigma_{n+2}$, где σ_{n+1} есть K, а σ_{n+2} есть L, является доказательством L, так как для каждого $i \in \{1, \ldots, n, n+1\}$ либо σ_i — аксиома, либо $\sigma_i \in S \cup \{K\}$, либо σ_i выводится из двух предшествующих высказываний с использованием правила Modus Ponens, σ_{n+2} выводится из σ_n и σ_{n+1} с использованием правила Modus Ponens.

($\Rightarrow$) Мы знаем, что существует доказательство L из $S \cup \{K\}$. Обозначим его через $L_1, L_2, \ldots, L_n$, где L_n есть L. Рассмотрим последовательность высказываний

$$K \to L_1, \quad K \to L_2, \quad \ldots, \quad K \to L_n.$$

Эта последовательность не является доказательством. Однако, она превратится в доказательство, если мы дополним ее высказываниями, полученными следующим индуктивным методом.

Для L_1 возможны три случая: L_1 — аксиома, либо L_1 принадлежит S, либо L_1 есть K.

Высказывание $L_1 \to (K \to L_1)$ — аксиома 1. Поэтому в первых двух случаях мы запишем L_1 и $L_1 \to (K \to L_1)$ перед $K \to L_1$. В последнем случае теорема верна (пример 1.8.3).

Предположим, что мы дополнили последовательность вплоть до высказывания $K \to L_m$ таким образом, что первые m элементов

последовательности вместе с дополняющими ее элементами составляют доказательство $K \to L_m$.

Тогда мы запишем между $K \to L_m$ и $K \to L_{m+1}$ еще ν высказываний так, что первые $m+1$ элементов последовательности вместе дополняющими ее элементами составят доказательство $K \to L_{m+1}$. Высказывание L_{m+1} либо аксиома, либо принадлежит S, либо есть K, либо получено из L_j, L_k $(1 \leqslant j, k \leqslant m)$ по правилу Modus Ponens. Другими словами, L_k имеет вид $L_j \to L_{m+1}$.

В первых трех случаях мы поступаем аналогично тому, как поступали выше, то есть записываем между $K \to L_m$ и $K \to L_{m+1}$ высказывания L_{m+1} и $L_{m+1} \to (K \to L_{m+1})$. В четвертом случае по индуктивному предположению первые m элементов последовательности вместе с дополняющими ее элементами составляют доказательства $K \to L_j$ и $K \to (L_j \to L_{m+1})$. Дополним последовательность высказыванием

$$(K \to (L_j \to L_{m+1})) \to ((K \to L_j) \to (K \to L_{m+1})),$$

представляющим собой аксиому 2.

Тогда $K \to L_{m+1}$ можно вывести, если дважды последовательно применить правило Modus Ponens. Заметим, что при доказательстве теоремы дедукции мы использовали только две из трех аксиом аксиоматической системы. □ 1.8.7

Аксиомы логики высказываний часто называют логическими аксиомами. Обычно мы определяем теорию, расширяя аксиоматизацию логики высказываний множеством S дополнительных аксиом, которые характеризуют теорию. Теоремы теории S являются элементами множества $\{\varphi\colon S \vdash \varphi\}$ (см. замечание 1.8.8).

Замечание 1.8.8. 1. *Аксиоматическая система.* Аксиомы PL и правило Modus Ponens составляют аксиоматическую систему Фреге–Лукашевича (см. [Heij67, Boye68, Schm60]). Фреге (Frege, 1898 – 1925) стал первым, кто определил формальный язык, подходящий для логики. Лукашевич (Lukasiewicz, 1878 – 1926) разработал аксиоматизацию логики высказываний.

2. *Modus Ponens.* Если σ и $\sigma \to \tau$ доказуемы по Бету, то σ и $\sigma \to \tau$ логически истинны, следовательно, τ также логически истинно (почему?). Известен алгоритмический метод, позволяющий строить доказательство по Бету высказывания τ из доказательств по Бету высказываний σ и $\sigma \to \tau$. Этот метод применяется в теореме Генцена, однако его доказательство выходит за рамки этой книги.

3. *Теоремы.* Теоремой является любое высказывание, присутствующее в доказательстве. Это означает, что теоремы теории S — это в точности элементы множества высказываний $\{\sigma\colon S \vdash \sigma\}$. Обычно мы считаем заключением последнее высказывание доказательства, однако, фактически, каждый начальный фрагмент доказательства также является доказательством.

4. *Выбор аксиом.* В следующих параграфах мы покажем, что методы аксиоматических доказательств корректны и полны. Таким образом, выбранная нами аксиоматическая система является полной. Это означает, что каждая тавтология может быть доказана из аксиом последовательными применениями правила Modus Ponens.

5. *Доказуемость по Бету.* Наши аксиомы, будучи логически истинными высказываниями, доказуемы по Бету. Используя правило Modus Ponens, мы из логически истинных высказываний получаем также логически истинное высказывание. Следовательно, каждая теорема доказуема по Бету.

6. *Аксиомы и правила.* Любую из аксиом аксиоматической системы можно заменить правилом. Например, третью аксиому

$$(\neg\varphi \to \neg\tau) \to (\tau \to \varphi)$$

можно было бы заменить правилом

$$\frac{\neg\varphi \to \neg\tau}{\tau \to \varphi}$$

Выбор между аксиомами и правилами — это обычно вопрос субъективной оценки характерных требований теории.

7. *Вывод из аксиом.* При доказательстве высказывания из аксиом мы рассматриваем различные комбинации высказываний, пытаясь определить такую, которая позволяет подходящим образом применить правило Modus Ponens и аксиомы. В результате, вывод некоторых высказываний, например,

$$\vdash (A \to B) \to ((C \to A) \to (C \to B)),$$

оказывается сложным и продолжительным, даже если бы нам было заранее известно, что и первая, и вторая аксиомы будут применяться дважды.

В отличие от этого, доказательства по Бету, введенные в определении 1.7.6, предоставляют систематический алгоритмический метод, всегда приводящий к некоторому результату. По этой причине мы предпочитаем работать с доказательствами по Бету. □ 1.8.8

§ 1.9. Метод резолюций

Терминология и обозначения в логическом программировании

Метод резолюций — наиболее эффективный способ алгоритмического доказательства как в логике высказываний, так и в логике предикатов. Именно этот метод построения доказательств составляет основу языка логического программирования Пролог. Метод

резолюций, как и метод семантических таблиц Бета, строит доказательство путем опровержения, но при этом он более удобен для написания логических программ в языке программирования, который устроен почти так же, как язык логики высказываний.

Для того, чтобы ввести резолюцию, нам необходимо определить некоторые важные понятия современного логического программирования.

Определение 1.9.1. *Литерал* — это произвольный атом или его отрицание. □ 1.9.1

Например, $\neg A$, B, $\neg C$ — литералы.

Нам известно, что произвольную формулу логики высказываний можно привести к конъюнктивной нормальной форме (КНФ), эквивалентной исходному высказыванию. КНФ — это по сути конъюнкция дизъюнкций литералов, причем в каждой дизъюнкции никакой литерал не встречается более одного раза.

Теперь мы опишем алгоритм построения КНФ для заданного высказывания, который работает немного быстрее, чем алгоритм, основанный на построении таблицы истинности высказывания, выборе строк и т. д.

Этот алгоритм состоит в применении:

1) законов де Моргана:

$$\neg(A \wedge B) \leftrightarrow \neg A \vee B, \quad \neg(A \vee B) \leftrightarrow \neg A \wedge B;$$

2) свойств ассоциативности $\wedge$ и $\vee$:

$$(A \wedge B) \wedge C \leftrightarrow A \wedge (B \wedge C), \quad (A \vee B) \vee C \leftrightarrow A \vee (B \vee C);$$

3) свойств коммутативности $\wedge$ и $\vee$:

$$A \wedge B \leftrightarrow B \wedge A, \quad A \vee B \leftrightarrow B \vee A;$$

4) свойств дистрибутивности $\wedge$ относительно $\vee$ и $\vee$ относительно $\wedge$:

$$A \wedge (B \vee C) \leftrightarrow (A \wedge B) \vee (A \wedge C), \quad A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C);$$

5) свойств:

$$A \wedge A \leftrightarrow A, \quad A \vee A \leftrightarrow A, \quad A \wedge (B \vee \neg B) \leftrightarrow A;$$

$$A \vee (B \vee \neg B) \leftrightarrow A, \quad \neg\neg A \leftrightarrow A$$

и теоремы о подстановке эквивалентных формул. (В качестве упражнения докажите, что перечисленные высказывания являются тавтологиями.)

Продемонстрируем на примере описанный метод построения КНФ.

Пример 1.9.2. Представим высказывание S в КНФ, где

$$S\colon \neg((A \vee B) \wedge (\neg A \vee B)) \wedge C.$$

Шаг 1. Используя законы де Моргана, вносим отрицания внутрь скобок:

$$S \leftrightarrow ((\neg(A \vee B) \vee \neg(\neg A \vee \neg B)) \wedge C,$$
$$S \leftrightarrow ((\neg A \wedge \neg B) \vee (\neg\neg A \vee \neg\neg B)) \wedge C.$$

Шаг 2. Используя свойства коммутативности и ассоциативности, собираем вместе литералы одного и того же атома. Затем, используя теорему о подстановке эквивалентных формул, мы можем упростить двойные отрицания, выражения вида $A \vee A$ и $A \wedge A$, а также устранить излишние выражения вида $A \vee \neg A$ и $A \wedge \neg A$:

$$S \leftrightarrow ((\neg A \wedge \neg B) \vee (A \vee B)) \wedge C.$$

Шаг 3. По свойству дистрибутивности имеем:

$$S \leftrightarrow (((\neg A \wedge \neg B) \vee A) \wedge (\neg A \wedge \neg B) \vee B)) \wedge C.$$

Затем мы повторяем второй и третий шаги до тех пор, пока не определим окончательную КНФ.

Шаг 4. $S \leftrightarrow ((\neg A \wedge \neg B) \vee A) \wedge (\neg A \wedge \neg B) \vee B) \wedge C,$

Шаг 5. $S \leftrightarrow (\neg A \vee \neg A) \wedge (\neg B \vee A) \wedge (\neg A \vee B) \wedge (\neg B \vee B) \wedge C,$

Шаг 6. $S \leftrightarrow (\neg B \vee A) \wedge (\neg A \vee B) \wedge C.$

В результате получена искомая КНФ. □ 1.9.2

Последняя форма S в примере 1.9.2 есть конъюнкция дизъюнкций литералов и эквивалентна исходной формуле. В общем случае описанный алгоритм заканчивает работу, когда получена следующая форма высказывания S:

$$(A_1^1 \vee A_2^1 \vee \ldots A_k^1) \wedge \ldots \wedge (A_1^\nu \vee A_2^\nu \vee \ldots \vee A_k^\nu), \qquad (*)$$

где $A_1^1, A_2^1, \ldots, A_k^1, \ldots, A_1^\nu, A_2^\nu, \ldots, A_k^\nu$ — литералы.

В контексте построения доказательств методом резолюций оказывается целесообразным формулировать высказывания как множества литералов. Например, высказывание в первых скобках в (*) переходит в множество

$$\{A_1^1, A_2^1, \ldots, A_k^1\}.$$

Мы считаем, что такое множество обозначает дизъюнкцию литералов, т. е. высказывание логики высказываний.

Теперь мы формально определим понятие теоретико-множественной формы высказываний.

Определение 1.9.3. Дизъюнкция конечного множества *литералов* может быть представлена *в теоретико-множественном виде* как множество, элементами которого являются рассматриваемые литералы. Это множество называется *дизъюнктом*. Таким образом, дизъюнкт эквивалентен дизъюнктивному высказыванию логики высказываний.

Мы также введем понятие *пустого дизъюнкта*, то есть дизъюнкта, который не содержит ни одного литерала и является всегда *неподтверждаемым*. Пустой дизъюнкт обозначается значком $\square$.

$\square$ 1.9.3

Определение 1.9.4. Конъюнкция конечного множества *дизъюнктов* может быть представлена *в теоретико-множественном виде* как множество, элементами которого являются эти дизъюнкты. Это множество называется *множеством дизъюнктов*. Таким образом, множество дизъюнктов представляет собой конъюнкцию дизъюнкций, то есть конъюнктивное высказывание логики высказываний. $\square$ 1.9.4

Пример 1.9.5. Множество дизъюнктов

$$\{\underbrace{\{A, B\}}_{1}, \quad \underbrace{\{\neg B, \neg C\}}_{2}, \quad \underbrace{\{D\}}_{3}\}$$

представляет высказывание

$$(\underbrace{(A \vee B)}_{1} \wedge \underbrace{(\neg B \vee \neg C)}_{2} \wedge \underbrace{D}_{3}).$$

$\square$ 1.9.5

Замечание 1.9.6. 1. Очевидно, что истинностное означивание подтверждает множество дизъюнктов, если оно подтверждает каждый дизъюнкт этого множества. Например, пусть $S = \{\{A, B\}, \{\neg C\}\}$ — множество дизъюнктов и V — такое истинностное означивание, что

$$V(A) = V(B) = V(C) = t.$$

Тогда V не подтверждает S, так как оно не подтверждает один из элементов S, а именно, $\{\neg C\}$.

2. Конечно, мы можем также рассматривать *пустое множество дизъюнктов* $\{\varnothing\}$, которое не следует путать с пустым дизъюнктом $\square$. Формально каждое истинностное означивание подтверждает пустое множество дизъюнктов, так как оно подтверждает каждый из его элементов (в дизъюнктах $\{\varnothing\}$ нет ни одного высказывания, см. доказательство следствия 1.5.4).

В отличие от этого, каждое множество дизъюнктов, содержащее пустой дизъюнкт, не может быть подтверждено никаким истинностным означиванием, так как □ неподтверждаем.

В сущности, пустое множество дизъюнктов обозначает, что к «миру» (множеству высказываний) не предъявляются *никакие «требования»* (высказывания), в то время как пустой дизъюнкт □, обозначая противоречие, тем самым вносит это противоречие в наш мир, делая его противоречивым, то есть неподтверждаемым. □ 1.9.6

В логическом программировании, а также в большинстве версий ПРОЛОГа, преобладает следующая система обозначений.

Пусть S — высказывание

$$A_1 \vee \ldots \vee A_k \vee (\neg B_1) \vee \ldots \vee (\neg B_l),$$

где $A_1, \ldots, A_k, B_1, \ldots, B_l$ — атомы. Тогда, согласно законам де Моргана и эквивалентности $(\neg B \vee A) \leftrightarrow (B \rightarrow A)$, мы имеем

$$S \leftrightarrow A_1 \vee \ldots \vee A_k \vee \neg(B_1 \wedge \ldots \wedge B_l) \leftrightarrow$$
$$\leftrightarrow (B_1 \wedge \ldots \wedge \neg B_l) \rightarrow (A_1 \vee \ldots \vee A_k)$$

и, в итоге,

$$S \leftrightarrow (A_1 \vee \ldots \vee A_k) \leftarrow (B_1 \wedge \ldots \wedge B_l). \qquad (1)$$

Относительно использования $\leftarrow$ как логической связки см. также замечание 1.2.9. В дальнейшем вместо логических связок $\wedge$, $\vee$ и $\leftarrow$ мы будем использовать символы «,» (запятую), «;» (точку с запятой) и символьную комбинацию «:-» соответственно. В результате S может быть эквивалентно записано в виде

$$A_1; \ldots; A_k :- B_1, \ldots, B_l. \qquad (2)$$

При $k = 1$ высказывание (1) примет вид:

$$A_1 :- B_1, \ldots, B_l. \qquad (3)$$

Определение 1.9.7. Дизъюнкт S вида (3) называется *хорновским дизъюнктом.*

Атом A_1 называется *заголовком*, или *целью* S; конъюнктивные компоненты $B_1, \ldots, B_l$ называются *хвостом*, *телом*, или *подцелями* S. □ 1.9.7

Содержательное истолкование хорновского дизъюнкта таково: для того, чтобы цель A была истинна, достаточно, чтобы подцели $B_1, \ldots, B_l$ были также истинны.

Определение 1.9.8. Если в дизъюнкте вида (2) $k = 0$, то дизъюнкт

$$:- B_1, \ldots, B_l \qquad (4)$$

называется *целью программы*, или *положительной целью.*

Если $l = 0$, то дизъюнкт

$$A_1 : - \tag{5}$$

называется *унитарным дизъюнктом*, или *фактом* □ 1.9.8

Замечание 1.9.9. 1. Множество дизъюнктов S может неформально рассматриваться как база данных (см. также раздел 3.1.1), так как дизъюнкты в S представляют информацию о взаимосвязи между атомами, из которых они состоят.

2. Подтверждение цели A в дизъюнкте $A : -B_1, \ldots, B_l$ вытекает из подтверждения подцелей $B_1, \ldots, B_l$. В этом случае говорят, что цель A *успешна*, или что существует доказательство A. В противном случае говорят, что цель A *неуспешна*, или *неудачна*[1]).

3. Хорновский дизъюнкт (4), обозначающий отсутствие цели, утверждает, что по меньшей мере одна из подцелей B_i, $i = 1, \ldots, l$, неуспешна. Хорновский дизъюнкт (5) обозначает, что цель A_1 всегда успешна. В этом случае A_1 составляет требование, факт нашей базы данных. □ 1.9.9

Вообще говоря, *резолюция* представляет собой дедуктивное правило, позволяющее нам выводить дизъюнктивное высказывание из двух других дизъюнктивных высказываний. Прежде, чем описать формальный метод, рассмотрим пример.

Пример 1.9.10. Рассмотрим следующие дизъюнкты: $\{\neg A, B\}$, $\{A, C\}$. Используя резолюцию, мы можем вывести дизъюнкт $\{B, C\}$.

Применение такого правила становится интуитивно понятным, если мы переформулируем вышеозначенные дизъюнкты в классической формулировке логики высказываний:

исходные высказывания: $(\neg A \vee B)$, $(A \vee C)$;

заключение: $(B \vee C)$.

В этом правиле применяется тавтология

$$(\neg A \vee B) \vee (A \vee C) \rightarrow (B \vee C).$$

Как свидетельствует теорема о полноте (теорема 1.10.9), тавтологии выводимы из аксиом с использованием правила Modus Ponens:

$$\vdash (\neg A \vee B) \vee (A \vee C) \rightarrow (B \vee C).$$

По теореме дедукции (теорема 1.8.7) мы получаем, что

$$\{(\neg A \vee B), (A \vee C)\} \vdash (B \vee C).$$

Таким образом, правило резолюции выводимо в логике высказываний. □ 1.9.10

[1]) Если нет других дизъюнктов с заголовком A. — *Прим. перев.*

Обобщим предыдущий пример, рассматривая рассматривая в качестве исходных данных следующие дизъюнкты:

$$C_1 = \{A_1, A_2, \ldots, A_{k_1}, \neg B_1, \ldots, \neg B_{l_1}\},$$
$$C_2 = \{D_1, D_2, \ldots, D_{k_2}, \neg F_1, \ldots, \neg F_{l_2}\},$$

где $A_1, A_2, \ldots, A_{k_1}, B_1, \ldots, B_{l_1}, D_1, D_2, \ldots, D_{k_2}, F_1, \ldots, F_{l_2}$ — атомы. Предположим, что A_1 совпадает с F_1.

Мы можем переписать эти дизъюнкты следующим образом:

$$C_1 = \{A_1\} \cup C_1', \text{ где } C_1' = \{A_2, \ldots, A_{k_1}, \neg B_1, \ldots, \neg B_{l_1}\},$$
$$C_2 = \{\neg A_1\} \cup C_2', \text{ где } C_2' = \{D_1, D_2, \ldots, D_{k_2}, \neg F_2, \ldots, \neg F_{l_2}\}.$$

Тогда по правилу резолюции можно вывести дизъюнкт $C = C_1' \cup C_2'$. Другими словами,

исходные высказывания: $C_1 = \{A_1\} \cup C_1'$, $C_2 = \{\neg A_1\} \cup C_2'$;

заключение: $C_1' \cup C_2' = (C_1 - \{A_1\}) \cup (C_2 - \{\neg A_1\})$. (*)

Мы можем считать, что дизъюнкты C_1 и C_2 «вступают в противоречие», так как C_1 содержит литерал A_1, а C_2 — литерал $\neg A_1$. Устранение причины противоречия приводит к дизъюнкту (*), который разрешает конфликт. Своим названием метод обязан этому разрешению[1]). Теперь мы формально определим метод резолюций.

Определение 1.9.11 (резолюция, формальное описание). Пусть C_1 и C_2 — дизъюнкты, L — такой литерал, что $L \in C_1$ и $\neg L \in C_2$. Тогда *резольвентой* дизъюнктов C_1 и C_2 называется дизъюнкт

$$D = (C_1 - \{L\}) \cup (C_2 - \{\neg L\}).$$

□ 1.9.11

Пример 1.9.12. Исходные высказывания: $C_1 = \{P, Q\}$, $C_2 = \{\neg P, \neg Q\}$; заключение: $D = \{Q, \neg Q\}$. □ 1.9.12

Если мы имеем множество, содержащее более двух дизъюнктов, то можно ввести понятие множества резольвент.

Определение 1.9.13. Пусть $S = \{C_1, \ldots, C_n\}$. Множество $R(S) = S \cup \{D: D$ — резольвента дизъюнктов $C_i, C_j, 1 \leqslant i, j \leqslant n\}$ называется *резольвентой* S. □ 1.9.13

Пример 1.9.14. Пусть S — множество дизъюнктов,

$$S = \{\underbrace{\{A, \neg B, \neg C\}}_{1}, \underbrace{\{B, D\}}_{2}, \underbrace{\{\neg A, \neg D\}}_{3}\}.$$

[1]) От английского to resolve (разрешать). — *Прим. перев*

Применяя правило резолюции для пар дизъюнктов S, мы получим дизъюнкты 4, 5, 6:

1	$\{A, \neg B, \neg C\}$	2	$\{B, D\}$	3	$\{\neg A, \neg D\}$
2	$\underline{\{B, D\}}$	3	$\underline{\{\neg A, \neg D\}}$	1	$\underline{\{A, \neg B, \neg C\}}$
4	$\{A, D, \neg C\}$	5	$\{B, \neg A\}$	6	$\{\neg B, \neg C, \neg D\}$

И, в итоге,

$$R(S) = \{\underbrace{\{A, \neg B, \neg C\}}_{1}, \underbrace{\{B, D\}}_{2}, \underbrace{\{\neg A, \neg D\}}_{3}, \underbrace{\{A, D, \neg C\}}_{4}, \underbrace{\{B, \neg A\}}_{5}, \underbrace{\{\neg B, \neg C, \neg D\}}_{6}\}.$$

Мы можем продолжить применять этот метод, последовательно получая множества:

$$R^0(S) = S, \quad R^1(S) = R^0(S) \cup R(S),$$

$$R^2(S) = R(S) \cup R(R(S)), \ldots, R^n(S) = R^{n-1}(S) \cup R(R^{n-1}(S)).$$

И, затем,

$$R^*(S) = \bigcup_{n=1}^{\infty} R^n(S) = \{C_i : C_i \in R^j(S) \text{ и } j \in N\},$$

где C_i — дизъюнкт, содержащийся в j-й резольвенте S.

Заметим, что $R^*(S)$ конечно тогда и только тогда, когда S конечно. □ 1.9.14

Замечание 1.9.15. 1. В примере 1.9.12 мы применили резолюцию по литералу P. Мы могли бы применить резолюцию и по литералу Q, так как, очевидно, Q также является причиной противоречия.

2. Содержательная подоплека метода резолюции такова: если истинностное означивание подтверждает C_1 и C_2, то оно также подтверждает их резольвенту D. Аналогично, если истинностное означивание подтверждает S, то оно также подтверждает $R(S)$.

3. Заметим, что резольвента D дизъюнктов C_1 и C_2 содержит меньше информации, чем C_1 и C_2. Это поясняет следующий пример 1.9.16. □ 1.9.15

Пример 1.9.16. Рассмотрим множество дизъюнктов $S = \{\{A, B\}, \{\neg B\}\}$.

Исходные высказывания: $C_1 = \{A, B\}$, $C_2 = \{\neg B\}$; заключение: $D = \{A\}$.

Применяя резолюцию к S, мы получаем дизъюнкт $D=\{A\}$, который не содержит никакой информации о литерале B. □ 1.9.16

Теперь мы дадим формальное определение доказательства методом резолюций.

Определение 1.9.17. Пусть S — множество дизъюнктов. *Резолютивным выводом* из S назовем такую конечную последовательность дизъюнктов $C_1, \ldots, C_n$, что для каждого C_i, $i=1, \ldots, n$, либо $C_i \in S$, либо $C_i \in R(\{C_j, C_k\})$, $1 \leqslant j, k \leqslant n$. Дизъюнкт C считается *резолютивно выводимым из множества дизъюнктов* S, и этот факт обозначается $S \vdash_R C$, если существует резолютивный вывод из S, последний дизъюнкт которого — C. Очевидно, что $C \in R^*(S)$. □ 1.9.17

Пример 1.9.18. Найдем резольвенту множества дизъюнктов

$$S=\{\{A, B\}, \{\neg A, \neg B\}\}.$$

Перенумеруем дизъюнкты S: $1 — \{A, B\}$, $2 — \{\neg A, \neg B\}$. Из дизъюнктов 1 и 2 получаем дизъюнкты $3 — \{B, \neg B\}$ и $4 — \{A, \neg A\}$. Затем

$$R^1(S)=\{\{A, B\},\ \{\neg A, \neg B\},\ \{B, \neg B\},\ \{A, \neg A\}\}.$$

В итоге получаем

$$R^*(S)=R^0(S) \cup R^1(S)=\{\{A, B\}, \{\neg A, \neg B\}, \{B, \neg B\}, \{A, \neg A\}\}.$$

Дизъюнкт вида $\{A, \neg A\}$, а именно $A \vee \neg A$, является тавтологией. □ 1.9.18

Пример 1.9.19. Дано высказывание

$$S\colon ((A \leftrightarrow (B \to C)) \wedge (A \leftrightarrow B) \wedge (A \leftrightarrow \neg C)).$$

Докажем, что S неподтверждаемо.

Доказательство. Шаг 1. Определим КНФ высказывания S:

$$S \leftrightarrow ((A \to (B \to C)) \wedge ((B \to C) \to A) \wedge (A \to B) \wedge$$
$$\wedge (B \to A) \wedge (A \to \neg C) \wedge (\neg C \to A) \leftrightarrow$$
$$\leftrightarrow \underbrace{(\neg A \vee \neg B \vee C)}_{1} \wedge \underbrace{(B \vee A)}_{2} \wedge \underbrace{(\neg C \vee A)}_{3} \wedge \underbrace{(\neg A \vee B)}_{4} \wedge$$
$$\wedge \underbrace{(\neg B \vee A)}_{5} \wedge \underbrace{(\neg A \vee \neg C)}_{6} \wedge \underbrace{(C \vee \neg A)}_{7}$$

Шаг 2. Сформируем соответствующее множество дизъюнктов:

$$S=\{\underbrace{\{\neg A, \neg B, C\}}_{1}, \underbrace{\{B, A\}}_{2}, \underbrace{\{\neg C, A\}}_{3}, \underbrace{\{\neg A, B\}}_{4},$$
$$\underbrace{\{\neg B, A\}}_{5}, \underbrace{\{\neg A, \neg C\}}_{6}, \underbrace{\{C, \neg A\}}_{7}\}.$$

Шаг 3. Определим различные резольвенты:

8) $\{A\}$ — резольвента 2 и 5;

9) $\{\neg A, \neg B\}$ — резольвента 1 и 6;

10) $\{\neg A\}$ — резольвента 4 и 9;

11) $\square$ — резольвента 8 и 10.

(См. также определение 1.9.8. Литерал $\neg A$ уничтожается, дизъюнкт 11 не содержит литералов.)

Так как мы получили пустой дизъюнкт (резольвента 11), то множество дизъюнктов S неподтверждаемо. Следовательно, высказывание S невыполнимо. $\square$ 1.9.19

Пример 1.9.20. Доказать, что высказывание $\neg B$ резолютивно выводимо из множества

$$S = \{\{A, \neg B\}, \{\neg A, \neg B, \neg C\}, \{\neg A, \neg B, C\}\}.$$

Доказательство. Построим резолютивный вывод дизъюнкта $\{\neg B\}$.

1) $\{A, \neg B\}$ — дизъюнкт из S;

2) $\{\neg A, \neg B, \neg C\}$ — дизъюнкт из S;

3) $\{\neg A, \neg B, C\}$ — дизъюнкт из S;

4) $\{\neg A, \neg B\}$ — резольвента 2 и 3;

5) $\{\neg B\}$ — резольвента 4 и 1. $\square$ 1.9.20

Замечание 1.9.21. Доказательство в примере 1.9.20 можно было провести следующим образом.

Применяя резолюцию к множеству

$$S_1 = \{\underbrace{\{A \neg B\}}_{1}, \underbrace{\{\neg A, \neg B, \neg C\}}_{2}, \underbrace{\{\neg A, \neg \boldsymbol{B}, \boldsymbol{C}\}}_{3}, \underbrace{\{B\}}_{4}\} = \boldsymbol{S} \cup \{\boldsymbol{B}\},$$

построим вывод

5) $\{A\}$ — резольвента 1 и 4;

6) $\{\neg A, \neg B\}$ — резольвента 2 и 3;

7) $\{\neg A\}$ — резольвента 4 и 6;

8) $\square$ — резольвента 5 и 7.

То есть, верно $S \cup \{B\} \vdash_R \square$. Так как правило резолюции — выводимое правило логики высказываний (как мы видели в примере 1.9.10), то верно также и соотношение $S \cup \{B\} \vdash \square$. Но в этом случае, по теореме дедукции (теорема 1.8.6) мы имеем $S \vdash \{B\} \to \square$. Используя тавтологию $(B \to \square) \leftrightarrow \neg B$, приходим к заключению о том, что $S \vdash \neg B$, или, другими словами, $\neg B$ выводимо из S.

$\square$ 1.9.21

§ 1.10. Корректность и полнота метода семантических таблиц

В следующих параграфах мы докажем теоремы корректности и полноты описанных методов доказательств. Мы начнем с доказательств корректности и полноты системы вывода по Бету.

Для всех последующих определений и теорем, включающих понятия «доказательство по Бету» и «логически истинно», имеются соответствующие двойственные определения и теоремы, относящиеся к двойственным понятиям «опровержение по Бету» и «логически ложно». Формулировку этих определений и теорем мы оставляем читателю в качестве упражнения.

Мы докажем, что все выводимые по Бету высказывания истинны (корректность) и, наоборот, каждое логически истинное высказывание выводимо по Бету (полнота) (см. [Smu168]). Доказательства проведем индукцией по длине высказывания или длине семантической таблицы. Мы уже описали схему индукции для высказываний. Теперь опишем общую схему индукции для семантических таблиц.

Определение 1.10.1 (схема индукции для семантических таблиц). Пусть R — некоторое свойство семантических таблиц. Тот факт, что таблица T обладает свойством R, обозначим $R(T)$. Как только мы докажем, что

а) каждая атомарная семантическая таблица обладает свойством R,

б) если T обладает свойством R, и T' — новая семантическая таблица, полученная присоединением некоторой атомарной семантической таблицы к концу одной из ветвей T, то T' также обладает свойством R,

то мы можем заключить, что R — свойство всех семантических таблиц, то есть $R(T)$ имеет место *для каждой семантической таблицы* T. □ 1.10.1

Для семантической таблицы T индукция проводится по *длине* T, то есть числу атомарных семантических таблиц в ней.

Аналогия между схемой индукции для высказываний (определение 1.2.2) и схемой индукции для семантических таблиц очевидна.

Пример 1.10.2. Пусть R — свойство «Число вершин в семантической таблице не превосходит числа ветвей». Покажем, что все таблицы обладают этим свойством.

Доказательство. 1. Во всех атомарных семантических таблицах число вершин не превосходит числа ветвей.

2. Предположим, что в семантической таблице T число вершин не превосходит числа ветвей. Присоединим к концу одной из ветвей T некоторую атомарную семантическую таблицу. Если присоединенная вершина — это $t(\neg\sigma)$ или $f(\neg\sigma)$, то число вершин увеличилось

на одну в то время, как число ветвей осталось без изменения. Если присоединенная вершина — это вершина любого другого типа, то, согласно определению 1.7.1, появятся не более двух новых ветвей и не менее двух новых вершин. Следовательно, в новой семантической таблице T' число вершин также не превосходит числа ветвей. □ 1.10.2

Теперь мы рассмотрим некоторые вспомогательные определения и леммы, относящиеся к теоремам корректности и полноты доказательств по Бету.

Определение 1.10.3. Пусть $\varkappa$ — непротиворечивая ветвь в семантической таблице T, и $P = \{P_1, \ldots, P_n\}$ — множество вершин $\varkappa$. *Истинностное означивание* V называется *согласованным с ветвью* $\varkappa$, если для каждой вершины P_i из P выполняются соотношения

$$P_i = t\sigma \Rightarrow V(\sigma) = t \quad \text{и} \quad P_i = f\sigma \Rightarrow V(\sigma) = f. \qquad \square\ 1.10.3$$

Лемма 1.10.4. *Пусть V — истинностное означивание, согласованное с корнем $t\sigma$ $(f\sigma)$ семантической таблицы, то есть $V(\sigma) = t$ $(V(\sigma) = f)$. Тогда V согласовано с некоторой ветвью семантической таблицы.*

Доказательство. Воспользуемся индукцией.

1. Лемма, очевидно, верна для атомарных семантических таблиц.

2. Предположим, что лемма верна для семантической таблицы T. Пусть T' — семантическая таблица, полученная из T присоединением атомарной семантической таблицы с корнем X к концу ветви $\varkappa$. Мы докажем, что лемма верна для семантической таблицы T'.

Случай 1. Предположим, что V согласовано со всеми вершинами ветви $\varkappa$. Тогда V согласовано с вершиной X и, следовательно, согласовано с одной из ветвей атомарной семантической таблицы с X в корне. Обозначим $\varkappa_1$ ветвь атомарной семантической таблицы, с которой согласовано V. Ветвь $\varkappa\varkappa_1$, полученная конкатенацией $\varkappa$ и $\varkappa_1$, является ветвью T'. Следовательно, мы нашли ветвь $\varkappa\varkappa_1$ семантической таблицы T', с которой согласовано V.

Случай 2. Предположим теперь, что V не согласовано с $\varkappa$, но согласовано с корнем T (иначе нам не нужно ничего доказывать). Тогда найдется другая ветвь $\varkappa'$ семантической таблицы T, с которой V согласовано. Но $\varkappa'$ — также ветвь T'. Следовательно, V согласовано с $\varkappa'$. □ 1.10.4

Лемма 1.10.5 (Хинтикки). *Пусть $\varkappa$ — непротиворечивая ветвь замкнутой семантической таблицы. Определим истинностное означивание V, полагая для каждого атома A:*

$V(A) = t$, если $\varkappa$ содержит вершину tA,
$V(A) = f$, если $\varkappa$ не содержит вершину tA.
Тогда истинностное означивание V согласовано с ветвью $\varkappa$.

Доказательство. Проведем индукцию по длине высказываний в вершинах этой ветви.

1. Если A — атом, и tA — вершина $\varkappa$, то $V(A) = t$ и V согласовано с $\varkappa$. Если fA — вершина $\varkappa$, то наша ветвь не содержит вершину tA, так как $\varkappa$ непротиворечива. Следовательно, $V(A) = f$.

2. Допустим $\varkappa$ содержит вершину $t(\sigma_1 \wedge \sigma_2)$. Ввиду того, что семантическая таблица замкнута, эта вершина уже была однажды востребована, и, следовательно, соответствующий фрагмент

$$t\sigma_1$$
$$|$$
$$t\sigma_2$$

был присоединен к концу ветви $\varkappa$.

Поэтому вершины $t\sigma_1$ и $t\sigma_2$ также содержаться в ветви $\varkappa$. По предположению индукции $V(\sigma_1) = t$ и $V(\sigma_2) = t$, то есть $V(\sigma_1 \wedge \sigma_2) = t$. Предположим теперь, что ветвь $\varkappa$ содержит вершину $f(\sigma_1 \wedge \sigma_2)$. Поскольку семантическая таблица замкнута, вершина $f(\sigma_1 \wedge \sigma_2)$ уже была однажды рассмотрена, и, следовательно, фрагмент

$$f\sigma_1 \qquad f\sigma_2$$

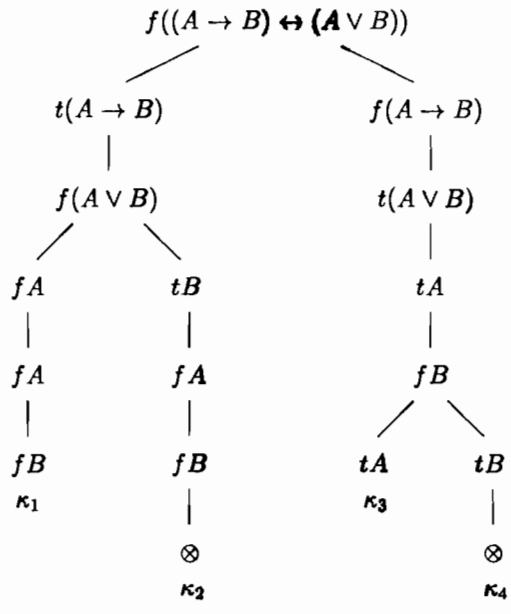
был присоединен к концу ветви $\varkappa$. Но в этом случае либо $f\sigma_1$, либо $f\sigma_2$ является вершиной $\varkappa$. Учитывая индуктивное предположение, это означает, что справедливо хотя бы одно из двух соотношений $V(\sigma_1) = f$ или $V(\sigma_1) = f$ Отсюда следует, что $V(\sigma_1 \wedge \sigma_2) = f$.

Остальные возможные случаи анализируются аналогично. Мы их оставляем читателю в качестве упражнения. □ 1.10.5

Лемма Хинтикки дает нам алгоритм построения контрпримера, опровергающего предположение об истинности заданного высказывания. Предположим, что нам дано высказывание σ. Тогда мы строим замкнутую семантическую таблицу с $f\sigma$ в корне. Если семантическая таблица противоречива, то высказывание логически истинно. Если семантическая таблица непротиворечива, то она содержит по меньшей мере одну непротиворечивую ветвь $\varkappa$. Лемма Хинтикки указывает нам, как, основываясь на $\varkappa$, построить истинностное означивание, такое, что $V(\sigma) = f$.

Рассмотрим описанные действия на примере.

Пример 1.10.6. Найти такое истинностное означивание V, что $V((A \to B) \leftrightarrow (A \vee B)) = f$. Построим семантическую таблицу с $f((A \to B) \leftrightarrow (A \vee B))$ в корне:



Ветви $\varkappa_2$ и $\varkappa_4$ этой семантической таблицы противоречивы. Однако мы можем применить лемму Хинтикки к непротиворечивым ветвям $\varkappa_1$ и $\varkappa_3$. В результате получим, например, такие истинностные означивания V_1 и V_2:

$$V_1(A) = f, \quad V_1(B) = f \qquad V_3(A) = t, \quad V_3(B) = f,$$

что

$$V_1((A \to B) \leftrightarrow (A \vee B)) = V_3((A \to B) \leftrightarrow (A \vee B)) = f.$$

Таким образом, мы отыскали два истинностных означивания, при которых высказывание $(A \to B) \leftrightarrow (A \vee B)$ принимает истинностное значение f. □ 1.10.6

Теорема 1.10.7 (корректности). *Если высказывание σ доказуемо по Бету, то оно логически истинно. Формальная запись:*

$$\vdash_B \sigma \Rightarrow \models \sigma.$$

Доказательство. Если высказывание σ не является логически истинным, то найдется истинностное означивание V, для которого $V(\sigma) = f$. По лемме 1.10.4 каждая семантическая таблица с помеченной формулой $f\sigma$ в корне имеет хотя бы одну ветвь $\varkappa$, с которой согласовано с V, и поэтому, $\varkappa$ непротиворечива (почему?). Следовательно, σ не доказуемо по Бету. □ 1.10.7

Замечание 1.10.8. При доказательстве теоремы 1.10.7 мы воспользовались третьей аксиомой

$$(\neg\mathcal{P}_1 \Rightarrow \neg\mathcal{P}_2) \Rightarrow (\mathcal{P}_2 \Rightarrow \mathcal{P}_1)$$

на уровне нашего метаязыка. Вместо того, чтобы доказывать $\mathcal{P}_2 \Rightarrow \mathcal{P}_1$, т. е. что доказуемость σ по Бету влечет логическую истинность σ, мы доказали, что $\neg\mathcal{P}_1 \Rightarrow \neg\mathcal{P}_2$. Другими словами, если σ не является логически истинным, то σ не доказуемо по Бету. Такой метод доказательства используется довольно часто и называется *непрямым* доказательством или доказательством *от противного*. (Прямой метод доказательства подразумевает непосредственное обоснование следствия $\mathcal{P}_2 \Rightarrow \mathcal{P}_1$). □ 1.10.8

Теорема 1.10.9 (полноты). *Если высказывание σ логически истинно, то оно доказуемо по Бету. Формальная запись:*

$$\models \sigma \Rightarrow \vdash_B \sigma.$$

Доказательство. Если высказывание σ логически истинно, то для каждого истинностного означивания V верно $V(\sigma) = t$. Предположим, что вывод по Бету для σ не существует. Мы построим замкнутую семантическую таблицу с помеченной формулой $f\sigma$ в корне. Эта семантическая таблица должна иметь непротиворечивую ветвь. По лемме 1.10.4 найдется соответствующе определенное истинностное означивание V, которое согласовано с этой ветвью, а значит, и с корнем $f\sigma$. Тогда $V(\sigma) = f$, а это противоречит тому факту, что σ — логически истинная формула. Следовательно, доказательство по Бету для σ обязано существовать. □ 1.10.9

Из доказательства теоремы полноты можно заметить, что если мы не можем построить доказательство по Бету для высказывания σ (а именно, замкнутую семантическую таблицу с $f\sigma$ в корне), то есть построенная замкнутая семантическая таблица имеет хотя бы одну непротиворечивую ветвь, то мы можем точно также, как в лемме 1.10.5, определить истинностное означивание, которое будет контрпримером, опровергающим утверждение о том, что σ логически истинно. Другими словами, *построение семантической таблицы для σ гарантирует, что мы получим либо доказательство σ, либо контрпример для σ.*

§ 1.11. Дедуктивный вывод из гипотез

Теорема компактности

Автоматический вывод теорем из предположений и данных — одна из важных задач логического программирования. В параграфе 1.5 мы рассматривали логические следствия из множества высказываний S. Высказывание σ было названо логическим следствием S, то есть $S \models \sigma$, если каждое истинностное означивание, делающее истинными все высказывания S, приписывает также значение t высказыванию σ. Теперь мы определим, что означает вывод высказывания из множества высказываний и предположений.

Определение 1.11.1. Пусть

$$\sigma, \varphi_1, \varphi_2, \ldots, \varphi_n, \ldots$$

— конечная или бесконечная последовательность высказываний. Высказывание σ называется *дедуктивно выводимым по Бету* из высказываний $\varphi_1, \varphi_2, \ldots, \varphi_n, \ldots$, если существует противоречивая семантическая таблица, построенная следующим образом.

Шаг 0. Помещаем $f\sigma$ в корень.

Шаг S_{2n}. Присоединяем $t\varphi_n$ к концу каждой непротиворечивой ветви.

Шаг S_{2n+1}. Применяем правила расширения (см. определение 1.7.1) к семантической таблице предыдущего шага T_{2n}. □ 1.11.1

Если последовательность высказываний бесконечна, то такое построение может никогда не завершиться. Высказывание σ дедуктивно выводимо по Бету, только в том случае, если построение заканчивается, и в результате получена противоречивая семантическая таблица. Тогда интуитивно понятно, что не существует истинностного означивания, которое приписывает истинностное значение t всем высказываниям φ_n, участвовавшим в построении семантической таблицы, и одновременно с этим приписывает истинностное значение f высказыванию σ. Если исходная последовательность конечна, то указанная процедура когда-нибудь завершится построением замкнутой семантической таблицы.

Проиллюстрируем описанное построение на примере.

Пример 1.11.2. Докажем, что высказывание A дедуктивно выводимо по Бету из высказываний $\neg B$ и $(A \vee B)$. Строим семантическую таблицу с fA в корне, присоединяя утверждения $t(\neg B)$ и $t(A \vee B)$ (см. рис. 1.3).

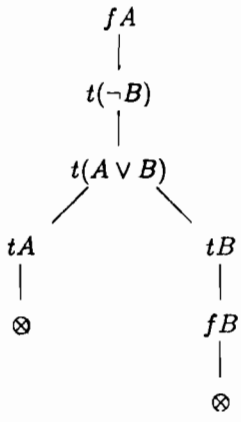


Рис. 1.3

Рассмотрим $t(A \vee B)$. Левая ветвь этой таблицы противоречива, и далее ее можно рассматривать. При анализе правой ветви нужно обратить внимание на вершину $t(\neg B)$. Правая ветвь таблицы оказывается также противоречивой. Следовательно, A дедуктивно выводимо по Бету из $\neg B$ и $(A \vee B)$. □ 1.11.2

Две последующие теоремы 1.11.3 и 1.11.4, относящиеся к конечным множествам предположений, соответствуют теоремам корректности и полноты из предыдущего параграфа 1.10.

Теорема 1.11.3 (теорема корректности дедуктивного вывода). *Если высказывание σ дедуктивно выводимо по Бету из высказываний $\varphi_1, \varphi_2, \ldots, \varphi_n$, то σ является логическим следствием высказываний $\varphi_1, \varphi_2, \ldots, \varphi_n$. Формальная запись:*

$$\{\varphi_1, \varphi_2, \ldots, \varphi_n\} \vdash_B \sigma \Rightarrow \{\varphi_1, \varphi_2, \ldots, \varphi_n\} \models \sigma.$$

Доказательство. Докажем теорему от противного.

Предположим, что высказывание σ не является логическим следствием $\varphi_1, \varphi_2, \ldots, \varphi_n$ Тогда найдется такое истинностное означивание V, что

$$V(\varphi_1) = \ldots = V(\varphi_n) = t, \quad V(\sigma) = f.$$

По лемме 1.10.4 каждая семантическая таблица с корнем $f\sigma$ содержит хотя бы одну ветвь, с которой согласовано V, и, следовательно, непротиворечива. Это означает, что σ не может быть доказуемым по Бету из $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$. □ 1.11.3

Теорема 1.11.4 (теорема полноты дедуктивного вывода). *Если высказывание σ является логическим следствием высказываний $\varphi_1, \varphi_2, \ldots, \varphi_n$, то σ дедуктивно выводимо по Бету из высказываний $\varphi_1, \varphi_2, \ldots, \varphi_n$. Формальная запись:*

$$\{\varphi_1, \varphi_2, \ldots, \varphi_n\} \models \sigma \Rightarrow \{\varphi_1, \varphi_2, \ldots, \varphi_n\} \vdash_B \sigma.$$

Доказательство. Предположим, что высказывание σ не является дедуктивно выводимым по Бету из $\varphi_1, \varphi_2, \ldots, \varphi_n$. Тогда мы можем построить полную непротиворечивую семантическую таблицу с корнем $f\sigma$, каждая ветвь которой содержит вершины $t\varphi_1, t\varphi_2, \ldots, t\varphi_n$. В этой семантической таблице найдется непротиворечивая ветвь. По лемме 1.10.5 существует истинностное означивание V, согласованное с этой ветвью. Следовательно, $V(\varphi_1) = \ldots = V(\varphi_n) = t$ и $V(\sigma) = f$. Однако это противоречит условию теоремы, согласно которому σ — логическое следствие высказываний $\varphi_1, \varphi_2, \ldots, \varphi_n$. □ 1.11.4

Замечание 1.11.5. Теоремы 1.11.3 и 1.11.4 корректности и полноты остаются верны, если последовательность $\{\varphi_k: k \in N\}$ содержит бесконечное число элементов. □ 1.11.5

Теперь мы можем сформулировать и доказать теорему компактности для логики высказываний. Введем основное определение.

Определение 1.11.6. Последовательность $\sigma_1, \sigma_2, \ldots, \sigma_n$ называется *выполнимой*, если существует такое истинностное означивание V, что $V(\sigma_1) = V(\sigma_2) = \ldots = V(\sigma_n) = t$. При этом говорят, что V *подтверждает* последовательность $\sigma_1, \sigma_2, \ldots, \sigma_n$. □ 1.11.6

Пример 1.11.7. Если $A_1, A_2, \ldots$ — последовательность атомов, то бесконечная последовательность

$$A_1, \quad A_2, \quad A_1 \wedge A_2, \quad A_3, \quad A_1 \wedge A_3, \quad A_2 \wedge A_3, \quad \ldots$$

— выполнима. Истинностное означивание V такое, что

$$V(A_1) = V(A_2) = \ldots = V(A_n) = \ldots = t,$$

подтверждает эту последовательность. Напротив, конечная последовательность высказываний

$$A_1, \quad A_2, \quad (A_1 \to A_3), \quad (\neg A_3)$$

может служить примером невыполнимой последовательности. В самом деле, если мы предположим, что она выполнима, то найдется такое истинностное означивание V, что

$$V(A_1) = V(A_2) = V(A_1 \to A_3) = V(\neg A_3) = t.$$

Отсюда следует, что справедливо соотношение $V(A_3) = f$. Но в таком случае из равенства $V(A_1) \rightsquigarrow V(A_3) = t$ вытекает $V(A_1) = f$, что противоречит предположению $V(A_1) = t$. □ 1.11.7

Прежде, чем сформулировать теорему компактности, приведем определение и лемму, необходимые для ее доказательства.

Определение 1.11.8. 1. Пусть X и Y — две вершины семантической таблицы. Назовем Y *последователем* X, если существует ветвь, проходящая через X и Y, в которой X расположена

ближе к корню, чем Y. Назовем Y *непосредственным последователем* X, если X — последователь Y, и в ветви, проходящей через X и Y, между X и Y нет ни одной вершины. Назовем X *подходящей* вершиной, если она имеет бесконечное число последователей.

2. Мы говорим, что семантическая таблица имеет *конечную степень ветвления*, если для каждой ее вершины найдется только *конечное* число ее непосредственных последователей. □ 1.11.8

Лемма 1.11.9 (лемма Кенига). *В семантической таблице, имеющей конечную степень ветвления и содержащей бесконечное число вершин, найдется хотя бы одна бесконечная ветвь.*

Доказательство. Заметим, что корень семантической таблицы с бесконечным числом вершин является подходящей вершиной. Более того, если X — подходящая вершина, то хотя бы один его непосредственный последователь также является подходящей вершиной, ибо семантическая таблица имеет конечную степень ветвления. Следовательно, если взять корень X_0, а затем в качестве X_1 того непосредственного последователя X_0, который является подходящей вершиной, а затем в качестве X_2 того непосредственного последователя X_1, который является подходящей вершиной, а затем... и т. д., то получим ветвь $X_0X_1X_2\ldots$, имеющую бесконечное число вершин. □ 1.11.9

Теорема 1.11.10 (теорема компактности). *Пусть $\sigma_1, \sigma_2, \ldots$ — такая бесконечная последовательность высказываний, что для каждого n конечная последовательность $\sigma_1, \sigma_2, \ldots, \sigma_n$ выполнима. Тогда бесконечная последовательность $\sigma_1, \sigma_2, \ldots$ также выполнима.*

Доказательство. Опишем индуктивное построение семантической таблицы, которая, вообще говоря, может быть бесконечной.

Шаг 1. Помещаем $f(\neg\sigma_1)$ в корень таблицы.

Шаг $2n$. Присоединяя $t\sigma_n$ к концу каждой непротиворечивой ветви таблицы T_{2n-1}, получаем таблицу T_{2n}.

Шаг $2n+1$. Выбираем в T_{2n} самую близкую к корню обычную вершину, расположенную как можно левее, и применим к ней правила развертывания определения 1.7.1.

При этом будем полагать, что построение завершается, если на нечетном шаге $2n+1$ в семантической таблице все ветви противоречивы. (Если они непротиворечивы, то мы можем продолжить построение на шаге $2n+1$.) Если построение завершилось, то мы получили противоречивую семантическую таблицу с корнем $f(\neg\sigma_1)$ и гипотезами $\sigma_1, \sigma_2, \ldots, \sigma_n$. По теореме 1.11.3 это означает, что $\neg\sigma_1$ является логическим следствием $\sigma_1, \sigma_2, \ldots, \sigma_n$, и, следовательно, $\sigma_1, \sigma_2, \ldots, \sigma_n$ невыполнима вопреки условию теоремы.

Таким образом, построение никогда не завершается, т. е. семантическая таблица содержит бесконечное число вершин. Построенная семантическая таблица имеет конечную степень ветвления, так как число непосредственных последователей у каждой ее вершины — конечно. Следовательно, по лемме Кенига в ней найдется бесконечная ветвь $\varkappa$, в которой для каждого высказывания σ_i, $i = 1, 2\ldots$, содержится вершина $t\sigma_i$.

Определим[1]) истинностное означивание V:

$V(A) = t \Leftrightarrow A$ — пропозициональный символ и tA — вершина $\varkappa$.

Тогда по лемме Хинтикки мы можем вывести, что $V(\sigma_i) = t$ для каждого i, и последовательность $\sigma_1, \sigma_2, \ldots, \sigma_n$ выполнима. □ 1.11.10

§ 1.12. Корректность и полнота аксиоматической системы вывода

Теперь мы сформулируем теоремы корректности, полноты и компактности аксиоматических методов. Доказательства этих теорем читатель может найти в большинстве книг по классической логике. См., например, [Klee52, Rasi74, RaSi70].

Теорема 1.12.1 (корректности и полноты). *Высказывание σ выводимо из множества высказываний S тогда и только тогда, когда σ — логическое следствие S. Формальная запись:*

$$S \vdash \sigma \Leftrightarrow S \models \sigma. \qquad \square\ 1.12.1$$

Следствие 1.12.2. *Высказывание σ выводимо из $S = \varnothing \Leftrightarrow$ $\Leftrightarrow \sigma$ — логически истинно.* □ 1.12.2

Теорема 1.12.3 (компактности). *Множество высказываний S выполнимо тогда и только тогда, когда выполнимо каждое его конечное подмножество.* □ 1.12.3

§ 1.13. Корректность и полнота метода резолюций

Следующие теоремы касаются корректности и полноты метода резолюций.

Теорема 1.13.1 (корректности и полноты метода резолюций). *Множество дизъюнктов S невыполнимо тогда и только тогда, когда $R^*(S)$ содержит пустой дизъюнкт. Формальная запись:*

$$\square \in R^*(S) \Leftrightarrow S \text{ невыполнимо.} \qquad \square\ 1.13.1$$

Предпошлем доказательству корректности метода резолюций вспомогательную лемму 1.13.2.

---

[1]) Используемая стратегия построения семантической таблицы гарантирует, что каждое высказывание σ_i будет «разобрано» на атомы. — *Прим. перев.*

Лемма 1.13.2. *Если* $\{C_1, C_2\}$ *— выполнимое множество дизъюнктов, и* C *— резольвента* C_1 *и* C_2*, то* C *выполнимо.*

Доказательство. По определению правила резолюции имеем $C_1 = \{p\} \cup C_1'$, $C_2 = \{\neg p\} \cup C_2'$ и $C = C_1 \cup C_2$ для некоторого литерала p. Если V — истинностное означивание, подтверждающее $\{C_1, C_2\}$, то либо $V(p) = t$, либо $V(\neg p) = t$. Предположим, что $V(p) = t$. Так как $V(C_2) = t$ и $V(\neg p) = f$, то $V(C_2') = t$. Поэтому $V(C) = t$. Если $V(\neg p) = t$, то $V(C_1') = t$, и мы вновь приходим к заключению $V(C) = t$. □ 1.13.2

Теорема 1.13.3 (корректности метода резолюций). *Если* $R^*(S)$ *содержит пустой дизъюнкт, то то* S *— невыполнимое множество дизъюнктов. Формальная запись:*

$$\square \in R^*(S) \Rightarrow S \text{ невыполнимо.}$$

Доказательство. Пусть $C_1, \ldots, C_k$ — резолютивный вывод из S. Тогда, применяя предыдущую лемму, по индукции можно доказать, что любое истинностное означивание, подтверждающее S, подтверждает также C_k. Если заключение — это пустой дизъюнкт, то C_k *совпадает* с □. Так как пустой дизъюнкт невыполним, то S невыполнимо. □ 1.13.3

При доказательстве полноты метола резолюций воспользуемся следующей вспомогательной леммой 1.13.4.

Лемма 1.13.4. *Пусть* S *— невыполнимое множество дизъюнктов, в котором встречаются только литералы* $A_1, A_2, \ldots, A_k$*. Пусть* S^{k-1} *— конечное множество дизъюнктов, резолютивно выводимое из* S*, в котором встречаются только атомарные высказывания* $A_1, A_2, \ldots, A_{k-1}$*. Тогда* S^{k-1} *невыполнимо.*

Доказательство. Предположим, что S^{k-1} выполнимо. Тогда найдется истинностное означивание V множества литералов $\{A_1, A_2, \ldots, A_{k-1}\}$, подтверждающее S^{k-1}. Пусть V_1 и V_2 — два таких расширения V, что

$$V_1(A_k) = t, \qquad V_2(A_k) = f.$$

Множество дизъюнктов S невыполнимо, поэтому найдется дизъюнкт $C_1 \in S$, который не подтверждается V_1. Тогда $\neg A_k \in C_1$, ибо в противном случае обе альтернативные возможности приводят к противоречию:

а) $A_k \notin C_1$, и V_1 подтверждает C_1;

б) $A_k \in C_1$, и V_1 подтверждает C_1.

Аналогично мы можем заключить, что найдется дизъюнкт $C_2 \in S$, который не подтверждается V_2, и содержит атом A_k.

Рассмотрим $D = (C_1 - \{\neg A_k\}) \cup (C_2 - \{A_k\})$. Дизъюнкт D является резольвентой C_1 и C_2, и $D \in S^{k-1}$. Следовательно, V подтверждает D. Другими словами, имеет место один из следующих вариантов:

а) V подтверждает $C_1 - \{\neg A_k\}$. Но тогда V_1 подтверждает C_1. Получаем противоречие.

б) V подтверждает $C_2 - \{A_k\}$. Но тогда V_2 подтверждает C_2. Получаем противоречие.

Следовательно, S^{k-1} невыполнимо. □ 1.13.4

Теорема 1.13.5 (полноты метода резолюций). *Если S — невыполнимое множество дизъюнктов, то из S резолютивно выводим пустой дизъюнкт. Формальная запись:*

$$S \text{ невыполнимо} \Rightarrow \square \in R^*(S).$$

Доказательство. По определению 1.9.3 множество S содержит конечное число литералов. Обозначим их $A_1, \ldots, A_k$. Применяя лемму 1.13.5, можно заключить, что S^{k-1} невыполнимо. Применив эту же лемму k раз, мы заключим, что S^0 невыполнимо. Так как никакое атомарное высказывание не содержится в дизъюнктах S^0, и $S^0 \subseteq R^*(S)$, то $\square \in S^0 \subseteq R^*(S)$. □ 1.13.5

Замечание 1.13.6. Если σ — высказывание, резолютивно выводимое из множества высказываний S, то σ выводимо из S (определение 1.8.5). Тогда по теореме полноты (теорема 1.12.1) можно заключить, что

$$S \vdash_R \sigma \Leftrightarrow S \vdash \sigma \Leftrightarrow S \models \sigma. \qquad (1)$$

По теоремам корректности и полноты (теоремы 1.10.6 и 1.10.8) соотношение (1) может быть обобщено следующим образом:

$$S \vdash_R \sigma \Leftrightarrow S \vdash_B \sigma \Leftrightarrow S \vdash \sigma \Leftrightarrow S \models \sigma. \qquad \square\ 1.13.6$$

§ 1.14. Упражнения

1.14.1. Докажите, что согласно определению **высказывания** следующие выражения являются высказываниями:

а) p: $(A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C))$;

б) q: $((A \vee B) \rightarrow C) \rightarrow (\neg A \wedge C))$;

в) r: $((A_1 \wedge A_2) \vee (\neg A_3))$.

Решение. а) Мы докажем, что выражение p является высказыванием, построив p индуктивно в соответствии с определением 1.2.1:

1) A, B, C — высказывания по определению 1.2.1, 1;

2) $(B \wedge C)$ — высказывание по определению 1.2.1, 2 и п. 1;

3) $A \vee (B \wedge C)$ — высказывание по определения 1.2.1, 2, а также пп. 1 и 2;

4) $(A \vee B)$ и $(A \vee C)$ — высказывания по определению 1.2.1, 2 и п. 1;

5) $(A \vee B) \wedge (A \vee C)$ — высказывание по определению 1.2.1, 2 и п. 4;

6) p: $(A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C))$ — высказывание по определению 1.2.1, 2 и 5.

Задания б) и в) решаются аналогично.

1.14.2. Определите, какие из следующих выражений являются высказываниями, а какие — нет:

а) $(A \wedge B) \vee \neg$; б) $((A \wedge B) \vee (\neg C)) \to D$;
в) $(A \vee B) \vee \to C$; г) $A \leftrightarrow B) \to (A \vee B)$;
д) $(A \wedge B) \to A$; е) $(A_1 \wedge A_2) \leftrightarrow \neg A_3$.

Р е ш е н и е. а) Это не высказывание, так как за $\vee$ следует логическая связка, к которой не относится никакое высказывание.

б) Это высказывание согласно определению 1.2.1 (этот факт обосновывается точно также, как в упражнении 1).

в) Это не высказывание, так как за $\vee$ следует логическая связка, а не высказывание.

г) Это не высказывание, неправильно расставлены скобки.

д) Это высказывание согласно определению 1.2.1 (для обоснования этого факта см. упражнение 1).

е) Это высказывание согласно определению 1.2.1 (для обоснования этого факта см. упражнение 1).

1.14.3. Представьте следующие высказывания атомарными символами и постройте из них составные высказывания:

а) «12 делится на 2», «9 делится на 3», «11 делится на 2»;

б) «все квадраты — параллелограммы», «все ромбы — параллелограммы», «во всяком параллелограмме диагонали точкой пересечения делятся пополам»;

в) «Джордж является отцом», «У Джорджа есть ребенок», «Мэри является отцом», «У Мэри есть ребенок».

Р е ш е н и е. а) Введем следующие обозначения:

A: «12 делится на 2»,

B: «9 делится на 3»,

C: «11 делится на 2».

Тогда $A \vee B$, $(A \wedge B) \to C$, $(A \vee B) \wedge (\neg C)$ и $A \to (B \wedge \neg C)$ — примеры составных высказываний, в которых используются символы A, B и C. Выражение $(A \wedge B) \to C$, конечно, является высказыванием,

однако, по всей видимости, соответствующее ему утверждение будет неверным.

Задания б) и в) решаются аналогично.

1.14.4. Даны атомарные высказывания:

A_1: «3 — простое число»; A_3: «2 делится на 3»;

A_2: «15 делится на 3»; A_4: «13 делится на 3».

а) Определите некоторое означивание F этих высказываний.

б) Пусть W_F — истинностное означивание, являющееся расширением F. Вычислите $W_F((A_1 \wedge A_2) \to (A_3 \vee A_4))$.

Р е ш е н и е. а) Рассмотрим, например, означивание:

$$F(A_1) = f, \qquad F(A_2) = t,$$
$$F(A_3) = f, \qquad F(A_4) = t.$$

б)

$$\begin{aligned} W_F((A_1 \wedge A_2) \to (A_3 \vee A_4)) &= W_F(A_1 \wedge A_2) \rightsquigarrow W_F(A_3 \vee A_4) = \\ &= [W_F(A_1) \sqcap W_F(A_2)] \rightsquigarrow [W_F(A_3) \sqcup W_F(A_4)] = \\ &= [F(A_1) \sqcap F(A_2)] \rightsquigarrow [F(A_3) \sqcup F(A_4)] = \\ &= (f \sqcap t) \rightsquigarrow (f \sqcup t) = \\ &= f \rightsquigarrow t \;=\; t. \end{aligned}$$

1.14.5. Докажите, что следующие высказывания являются тавтологиями:

а) $(A \wedge \neg A) \to A$; в) $A \to \neg\neg A$;

б) $(A \to B) \vee (A \to \neg B)$; г) $[(A \vee B) \to C) \leftrightarrow [A \to (B \to C)]$.

Р е ш е н и е. а) Рассмотрим означивание $W(A) = t$, а также его расширение W'. Тогда

$$\begin{aligned} W'[(A \wedge \neg A) \to A] &= W'(A \wedge \neg A) \rightsquigarrow W'(A) = \\ &= [W'(A) \sqcap W'(\neg A)] \rightsquigarrow W'(A) = \\ &= [W'(A) \sqcap (\sim W'(A))] \rightsquigarrow W'(A) = \\ &= [W(A) \sqcap (\sim W(A))] \rightsquigarrow W(A) = \\ &= [t \sqcap (\sim t)] \rightsquigarrow t = \\ &= [t \sqcap f] \rightsquigarrow t = \\ &= f \rightsquigarrow t \;=\; t. \end{aligned}$$

Рассмотрим теперь означивание $W(A) = f$ и его расширение W'. Таким же способом мы можем заключить, что

$$W'[(A \wedge \neg A) \to A] \;=\; t.$$

Итак, для любого истинностного означивания высказывание а) истинно. Значит оно является тавтологией.

Задания б), в) и г) решаются аналогично.

1.14.6. Докажите, что следующие высказывания логически ложны:

а) $(A \wedge \neg A)$; в) $(A \to B) \wedge (B \to C) \wedge (A \wedge \neg C)$;

б) $(\neg A \vee (B \wedge \neg B)) \leftrightarrow A$; г) $\neg(A \wedge B) \wedge (A \to B) \wedge A$.

Р е ш е н и е. г) Выпишем всевозможные означивания атомов из г):

1) $W(A) = t$, $W(B) = t$;
2) $W(A) = t$, $W(B) = f$;
3) $W(A) = f$, $W(B) = t$;
4) $W(A) = f$, $W(B) = f$.

Затем для каждого из указанных означиваний W найдем соответствующие истинностные означивания W', расширяющие W. Например, для 1) имеем

$$\begin{aligned} W'[\neg(A \wedge B) \wedge (A \to B) \wedge A] &= \\ &= W'[\neg(A \wedge B)] \sqcap W'[(A \to B) \wedge A] = \\ &= (\sim W'(A \wedge B) \sqcap W'(A \to B) \sqcap W'(A) = \\ &= (\sim (W'(A) \sqcap W'(B)) \sqcap (W'(A) \rightsquigarrow W'(B)) \sqcap W'(A) = \\ &= (\sim (W(A) \sqcap W(B)) \sqcap (W(A) \rightsquigarrow W(B)) \sqcap W(A) = \\ &= (\sim [t \sqcap t]) \sqcap [t \rightsquigarrow t] \sqcap t = \\ &= \sim t \sqcap t \sqcap t = f \sqcap t = f. \end{aligned}$$

Аналогично вычисляются истинностные означивания для случаев 2), 3) и 4). В каждом из этих случаев мы получаем $W'[\neg(A \wedge B) \wedge (A \to B) \wedge A] = f$. Таким образом, формула г) противоречива.

Пункты а), б) и в) решаются аналогично.

1.14.7. Заполните следующую таблицу истинности:

A	B	$\neg A$	$\neg B$	$A \to B$	$\neg A \vee B$	$(A \to B) \leftrightarrow (\neg A \vee B)$
t	t					
t	f					
f	t					
f	f					

1.14.8. Докажите, что следующие высказывания логически эквивалентны:

а) $\neg(A \wedge B)$ и $\neg A \vee \neg B$;

б) $A \vee (B \wedge C)$ и $(A \vee B) \wedge (A \vee C)$;

в) $A \vee B$ и $B \vee A$;

г) $A \to B$ и $\neg B \to \neg A$.

Р е ш е н и е. б) Построим таблицы истинности для обоих высказываний и убедимся, что последние колонки этих таблиц совпадают:

A	B	C	$B \wedge C$	$A \vee B$	$A \vee C$	$A \vee (B \wedge C)$	$(A \vee B) \wedge (A \vee C)$
t	t	t	t	t	t	t	t
t	t	f	f	t	t	t	t
t	f	t	f	t	t	t	t
t	f	f	f	t	t	t	t
f	t	t	t	t	t	t	t
f	t	f	f	t	f	f	f
f	f	t	f	f	t	f	f
f	f	f	f	f	f	f	f

Итак, $A \vee (B \wedge C)$ и $(A \vee B) \wedge (A \vee C)$ логически эквивалентны. Пункты а), в) и г) решаются аналогично.

1.14.9. С использованием таблиц истинности **докажите, что** следующие высказывания являются тавтологиями:

а) $(A \vee B) \vee C \leftrightarrow A \vee (B \vee C)$;

б) $(A \wedge B) \wedge C \leftrightarrow A \wedge (B \wedge C)$;

в) $A \vee B \leftrightarrow B \vee A$;

г) $A \leftrightarrow \neg\neg A$;

д) $A \vee (B \wedge \neg B) \leftrightarrow A$;

е) $A \wedge (B \vee \neg B) \leftrightarrow A$;

ж) $A \wedge (B \vee C) \leftrightarrow (A \wedge B) \vee (A \wedge C)$;

з) $A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C)$;

и) $\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$;

к) $\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$.

1.14.10. Пусть $S = \{A \vee B, A \to C\}$. Докажите, что $S \models B \vee C$.

Р е ш е н и е. Построим таблицу истинности высказываний из множества S:

A	B	C	$A \vee B$	$A \to C$
t	t	t	t	t
t	t	f	t	f
t	f	t	t	t
t	f	f	t	f
f	t	t	t	t
f	t	f	t	t
f	f	t	f	t
f	f	f	f	t

Из таблицы видно, что существуют четыре означивания, при которых все высказывания из S становятся истинными. Проверим, что в каждом из этих случаев высказывание $B \vee C$ также обращается в истину. Имеем

$$W(A) = t, \quad W(B) = t, \quad W(C) = t,$$
$$W(B \vee C) = W(B) \sqcup W(C) = t \sqcup t = t.$$

Точно так же можно обнаружить, что $W(B \vee C) = t$ справедливо и для трех других означиваний, при которых одновременно выполняются $W(A \wedge B) = t$ и $W(A \to C) = t$. Таким образом, $S \models B \vee C$

1.14.11. Пусть $S = \{A \leftrightarrow C, B \leftrightarrow D, (A \vee B) \wedge (C \vee D)\}$. Докажите, что $S \not\models (A \wedge B) \vee (C \wedge D)$.

Р е ш е н и е. Постройте таблицу истинности для высказываний из множества S и определите означивания, при которых эти высказывания обращаются в истину. Далее докажите, что имеются в точности три таких истинностных означивания. Затем надлежит вычислить соответствующие истинностные значения для $(A \wedge B) \vee (C \wedge D)$. В двух из этих трех случаев $(A \wedge B) \vee (C \wedge D)$ становится ложным.

1.14.12. Докажите, что если $\{A, \neg B\} \models C \wedge \neg C$, то $\{A\} \models B$.

Р е ш е н и е. Предположим, что $\{A\} \not\models B$. Тогда существует такое истинностное означивание W, что $W(A) = t$ и $W(B) = f$

и, следовательно, $W(\neg B) = f$. Таким образом, W позволяет обратить в истину как A, так и B. Значит высказывание $C \wedge \neg C$ тоже должно быть истинным при означивании W. Другими словами, $W(C \wedge \neg C) = t$. Но тогда $W(C) \sqcap W(\neg C) = t$. Следовательно, $W(C) = W(\neg C) = t$ и $W(\neg C) = \sim W(C) = \sim t = f$, что является противоречием. Поэтому исходное предположение неверно, и $\{A\} \models B$.

1.14.13. Пусть S_1 и S_2 — два множества высказываний. Выясните, какие из следующих утверждений верны:

а) $Con(S_1 \cup S_2) = Con(S_1) \cup Con(S_2)$;

б) $Con(S_1 \cap S_2) = Con(S_1) \cap Con(S_2)$.

Верные утверждения докажите, неверные — соответственно опровергните контрпримером.

Р е ш е н и е. а) Докажем, что $Con(S_1) \cup Con(S_2) \subseteq Con(S_1 \cup S_2)$. Предположим, $\sigma \in Con(S_1) \cup Con(S_2)$. Тогда либо $\sigma \in Con(S_1)$, либо $\sigma \in Con(S_2)$. Поскольку $S_1 \subseteq S_1 \cup S_2$, мы имеем $Con(S_1) \subseteq Con(S_1 \cup S_2)$ (см. следствие 1.5.8). Аналогично, из $S_2 \subseteq S_1 \cup S_2$ вытекает $Con(S_2) \subseteq Con(S_1 \cup S_2)$. Таким образом, $\sigma \in Con(S_1 \cup S_2)$, или, другими словами,

$$Con(S_1) \cup Con(S_2) \subseteq Con(S_1 \cup S_2).$$

Однако обратное включение $Con(S_1 \cup S_2) \subseteq Con(S_1) \cup Con(S_2)$, вообще говоря, несправедливо. Рассмотрим $S_1 = \{A\}$, $S_2 = \{A \to B\}$. Легко заметить, что для каждого означивания W, подтверждающего одновременную выполнимость A и $A \to B$, высказывание B также обращается в истину. Значит,

$$B \in Con(S_1 \cup S_2). \tag{1}$$

Однако, $B \notin Con(S_1)$. Чтобы убедиться в этом, достаточно рассмотреть означивание W_1, при которой $W_1(A) = t$ и $W_1(B) = f$. С другой стороны, $B \notin Con(S_2)$. Подтверждением тому может служить истинностное означивание W_2, при котором $W_2(A) = f$, $W_2(B) = f$. Очевидно, при этом $W_2(A \to B) = t$. Следовательно $B \notin Con(S_1 \cup S_2)$. Учитывая (1), мы получаем

$$Con(S_1 \cup S_2) \not\subseteq Con(S_1) \cup Con(S_2).$$

Таким образом, утверждение а) в общем случае неверно.

б) Приведем контрпример. Предположим $S_1 \cap S_2 = \varnothing$. Согласно следствию 1.5.4 мы имеем $Con(S_1 \cap S_2) = Con(\varnothing)$. Последнее множество формул образовано всевозможными тавтологиями. Рассмотрим далее случай $S_1 = \{A\}$ и $S_2 = \{B\}$. Тогда $A \vee B$ принадлежит как $Con(S_1)$, так и $Con(S_2)$, поскольку каждое означивание, обращающее в истину элементы S_1 или элементы S_2, также делает истинным и высказывание $A \vee B$. Однако $A \vee B$ не является тавтологией (почему?). Значит утверждение б) неверно.

1.14.14. Докажите, что из $S \cup \{A\} \models B$ следует $S \models A \to B$.

Р е ш е н и е. Допустим $S \cup \{A\} \models B$. Тогда каждое означивание W, подтверждающее все элементарные высказывания из $S \cup \{A\}$, также обращает в истину и B. Иными словами, для каждого означивания W, при котором $W(C) = t$ для каждого $C \in S$, равенство $W(A) = t$ неизбежно влечет $W(B) = t$. Тогда $W(A \to B) = W(A) \rightsquigarrow W(B) = t \rightsquigarrow t = t$. Значит, $S \models A \to B$.

1.14.15. Для произвольного непротиворечивого множества высказываний S докажите, что

$$S \cup \{\varphi\} \text{ противоречиво} \Leftrightarrow S \models \neg\varphi.$$

Р е ш е н и е. ($\Rightarrow$) Предположим, $S \cup \{\varphi\}$ — противоречивое множество. Поскольку S непротиворечиво, существует по крайней мере одно означивание W, такое, что для всякого высказывания $\sigma \in S$ выполняется $W(\sigma) = t$. Если $W(\varphi) = t$, то $S \cup \{\varphi\}$ — непротиворечивое множество вопреки предположению. Значит $W(\varphi) = f$, что влечет $W(\neg\varphi) = t$ для всякого означивания W, при котором $W(\sigma) = t$ для каждого $\sigma \in S$. Таким образом, $S \models \neg\varphi$.

($\Leftarrow$) Верно и обратное. Предположим, что $S \models \neg\varphi$. Тогда для каждого истинностного означивания W такого, что $W(\sigma) = t$ для любого высказывания σ из S, будет справедливо $W(\neg\varphi) = t$, а следовательно и $W(\varphi) = f$. Это означает, что всякий раз, когда все элементы из S принимают значение t, высказывание φ имеет значение f. Значит множество $S \cup \{\varphi\}$ противоречиво.

1.14.16. Представьте приведенные ниже утверждения как множества высказываний PL и определите, какие из получившихся множеств являются непротиворечивыми.

S_1: Свидетель был испуган, или в том случае, если Джон совершил самоубийство, записка была обнаружена. Если свидетель испуган, то Джон убил себя сам.

S_2: Любовь слепа, но счастье нам доступно, или любовь слепа, а женщины гораздо умнее мужчин. Если счастье нам доступно, то любовь совсем не слепа. Женщины не более умны, чем мужчины.

Р е ш е н и е. Выразите утверждения из S_1 и S_2 формулами и постройте для них таблицы истинности. Далее воспользуйтесь определением 1.5.5.

1.14.17. Докажите, что если $S_1 \subseteq Con(S_2)$, то $Con(S_1 \cup S_2) = Con(S_2)$.

1.14.18. Пусть Q — множество высказываний[1]), а S — непротиворечивое подмножество Q. Назовем S ***максимальным непро-***

[1]) Содержащее вместе с каждым высказыванием σ и его отрицание $\neg\sigma$. — *Прим. перев.*

тиворечивым множеством, если всякое множество S' такое, что $S \subset S' \subseteq Q$, является противоречивым.

Докажите, что непротиворечивое подмножество S является максимальным непротиворечивым множеством тогда и только тогда, когда для каждого высказывания $\sigma \in Q$ справедливо в точности одно из следующих утверждений: $\sigma \in S$, $\neg\sigma \in S$.

Р е ш е н и е. ($\Rightarrow$) Допустим, что S — максимальное непротиворечивое множество высказываний. Рассмотрим $\sigma \in Q$ такое, что $\sigma \notin S$. Тогда S — это собственное подмножество множества $S' = S \cup \{\sigma\}$, которое противоречиво по определению S. Таким образом, если означивание W подтверждает все высказывания из S, оно опровергнет σ. Так как $W(\sigma) = f$, мы имеем $W(\neg\sigma) = t$. Следовательно, $\neg\sigma$ принадлежит S; иначе S было бы собственным подмножеством $S'' = S \cup \{\neg\sigma\}$, и мы получили бы, что $W(\varphi) = t$ для каждого $\varphi \in S''$, в том числе и $W(\neg\sigma) = t$. Другими словами, S'' было бы непротиворечивым, вопреки предположению о том, что S — максимальное непротиворечивое множество.

Точно так же мы можем доказать, что если $\neg\sigma \notin S$, то $\sigma \in S$.

($\Leftarrow$) Предположим, что S непротиворечиво, и для каждого $\sigma \in Q$ справедливо только одно из двух включений ($\sigma \in S$) и ($\neg\sigma \in S$). Мы докажем, что S — максимальное непротиворечивое множество высказываний.

Пусть S' — такое множество, что $S \subset S' \subseteq Q$. Рассмотрим $\varphi \in (S' - S)$. Тогда $\varphi \in S'$, $\varphi \notin S$ и $\neg\varphi \in S$ (почему?). Напомним, что S непротиворечиво. Следовательно, для любого означивания W, при котором истинны все высказывания из S, мы получим $W(\neg\varphi) = t$, или $W(\varphi) = f$. Но в таком случае не существует означивания, при котором истинны все высказывания из S', поскольку $\neg\varphi \in S \subset S'$ и $\varphi \in S'$. Следовательно, ввиду того, что противоречиво любое множество S' такое, что $S \subset S' \subseteq Q$, приходим к заключению, что S — максимальное непротиворечивое множество.

1.14.19. Даны высказывания: A: «$KLMN$ — параллелограмм»; B: «диагонали $KLMN$ делятся точкой пересечения пополам»; C: «противоположные углы $KLMN$ равны»; D: «противоположные стороны $KLMN$ равны».

Определите, является ли множество $S = \{A, A \leftrightarrow B, A \leftrightarrow C, A \rightarrow (B \wedge D)\}$ непротиворечивым.

1.14.20. Доказать, что не существует логической связки o, соединяющей два высказывания языка PL, отличающейся от | и : (см. пример 1.6.5), и такой, что множество связок $\{o\}$ образует полную систему.

Р е ш е н и е. Пусть o — такая логическая связка, что $A\ o\ B$ синтаксически корректная формула языка логики высказываний. Существует столько различных возможностей для определения таблицы

истинности $A\ o\ B$, сколько существует четырехместных наборов, образованных истинностными значениями t и f, т. е. 16.

A	B	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8
t	t	t	f	t	f	t	f	t	f
t	f	t	t	f	f	t	t	f	f
f	t	t	t	t	t	f	f	f	f
f	f	t	t	t	t	t	t	t	t

A	B	o_9	o_{10}	o_{11}	o_{12}	o_{13}	o_{14}	o_{15}	o_{16}
t	t	t	f	t	f	t	f	t	f
t	f	t	t	f	f	t	t	f	f
f	t	t	t	t	t	f	f	f	f
f	f	f	f	f	f	f	f	f	f

Содержательный смысл o_i, $1 \leqslant i \leqslant 16$, станет более ясным, если мы воспользуемся следующими равносильными формулами

$A\ o_1\ B \equiv \neg A \vee A$, (истина);
$A\ o_2\ B \equiv \neg(A \vee B)$, ($A|B$ из примера 1.6.5);
$A\ o_3\ B \equiv A \to B$;
$A\ o_4\ B \equiv \neg A$;
$A\ o_5\ B \equiv B \to A$;
$A\ o_6\ B \equiv \neg B$;
$A\ o_7\ B \equiv A \leftrightarrow B$;
$A\ o_8\ B \equiv \neg(A \vee B)$, ($A$: B из примера 1.6.5);
$A\ o_9\ B \equiv \neg(A \vee B)$;
$A\ o_{10}\ B \equiv \neg(A \leftrightarrow B)$;
$A\ o_{11}\ B \equiv B$;
$A\ o_{12}\ B \equiv \neg(A \to B)$;
$A\ o_{13}\ B \equiv A$;
$A\ o_{14}\ B \equiv \neg(B \to A)$;
$A\ o_{15}\ B \equiv (A \wedge B)$;
$A\ o_{16}\ B \equiv \neg A \wedge A$, (противоречие).

Указанная таблица истинности содержит логические связки, в том числе, относящиеся только к одному высказыванию, например o_4 и o_6. Эта задача, таким образом, позволит нам сделать более общее заключение, касающееся множества всех связок.

Для каждой связки $o \in \{o_1, \ldots, o_{16}\}$ можно отметить следующее: если $A\ o\ B$ принимает значение t в случае, когда оба высказывания A и B истинны, то множества $\{o\}$ недостаточно для нашей цели. Дело в том, что отрицание $\neg$ не может быть выражено посредством o: сколько бы раз не встречался символ o в формуле, выражающей отрицание, значение формулы всегда будет t и никогда не будет f, если и A и B имеют значение t. Значит $o \notin \{o_1, o_3, o_5, o_7, o_9, o_{11}, o_{13}, o_{15}, \}$.

По той же причине если $A\ o\ B$ принимает значение f в случае, когда оба высказывания A и B означены f, то множества $\{o\}$ также недостаточно для полноты: отрицание $\neg$ нельзя выразить при помощи o. Значит $o \notin \{o_9, o_{10}, o_{11}, o_{12}, o_{13}, o_{14}, o_{15}, o_{16}, \}$.

Если $o = o_4$, то множества $\{o\}$ недостаточно, так как $\neg\neg A \equiv A$. Мы всегда будем иметь либо A, либо $\neg A$, сколько раз бы ни встречалось отрицание, так что нам не удастся выразить $\wedge$ или $\vee$. Следовательно, $o \neq o_4$, и по аналогичной причине $o \neq o_6$.

Если же $o = o_2$ или $o = o_8$, то $\{o\}$ достаточно, как мы уже видели в примере 1.6.5.

1.14.21. При помощи метода семанических таблиц докажите истинность следующих высказываний:

а) $\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$;

б) $(A \wedge (A \rightarrow B)) \rightarrow B$;

в) $((A \wedge B) \rightarrow C) \leftrightarrow (A \rightarrow (B \rightarrow C))$;

г) $((A \rightarrow B) \wedge (A \rightarrow C)) \rightarrow (A \rightarrow (B \wedge C))$;

д) $((A \rightarrow C) \wedge (B \rightarrow C)) \rightarrow ((A \vee B) \rightarrow C)$;

е) $(A \wedge (B \vee C)) \leftrightarrow ((A \wedge B) \vee (A \wedge C))$;

ж) $(A \rightarrow (B \rightarrow C)) \leftrightarrow (B \rightarrow (A \rightarrow C))$;

з) $(A \leftrightarrow B) \leftrightarrow ((A \wedge B) \vee (\neg A \wedge \neg B))$.

Р е ш е н и е. в) Построим семантическую таблицу с $f[(A \wedge B) \rightarrow \rightarrow C) \leftrightarrow (A \rightarrow (B \rightarrow C))]$ в корне (см. рис. 1.4).

е) Построим семантическую таблицу с $f[(A \wedge (B \vee C)) \leftrightarrow ((A \wedge \wedge B) \vee (A \wedge C))]$ в корне (см. рис. 1.5).

1.14.22. Докажите теорему 1.8.4 о подстановке эквивалентных формул.

Р е ш е н и е. Проведем доказательство теоремы по индукции.

1. Б а з и с и н д у к ц и и. Пусть количество логических связок, встречающихся в формуле φ, равно нулю. Следовательно, φ —

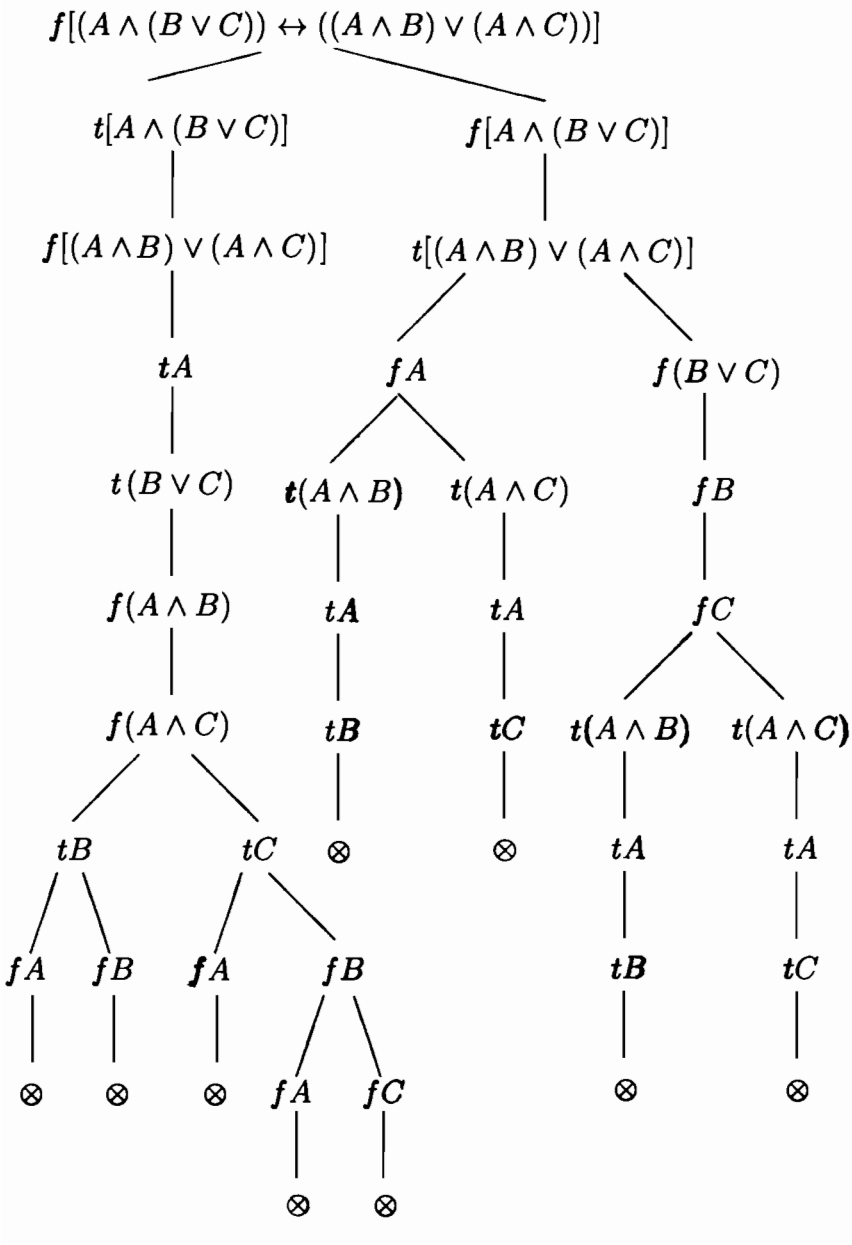

f[(A ∧ (B ∨ C)) ↔ ((A ∧ B) ∨ (A ∧ C))]
t[A ∧ (B ∨ C)]
f[A ∧ (B ∨ C)]
f[(A ∧ B) ∨ (A ∧ C)]
t[(A ∧ B) ∨ (A ∧ C)]
tA
fA
f(B ∨ C)
t(B ∨ C)
t(A ∧ B)
t(A ∧ C)
fB
f(A ∧ B)
tA
tA
fC
f(A ∧ C)
tB
tC
t(A ∧ B)
t(A ∧ C)
tB
tC
⊗
⊗
tA
tA
fA
fB
fA
fB
tB
tC
⊗
⊗
⊗
fA
fC
⊗
⊗

Рис. 1.4

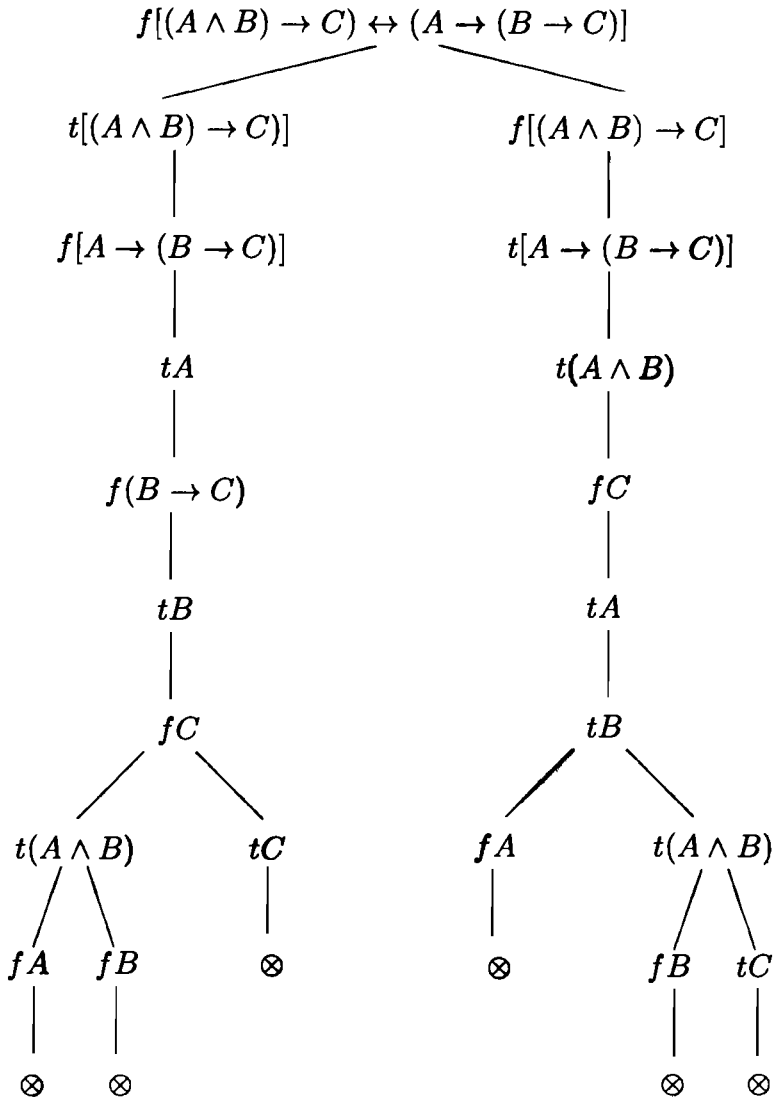


Рис. 1.5

атомарное высказывание, и φ и σ должны совпадать. Тогда формула φ' — это φ или σ'. Итак, мы получили $\vdash \varphi \leftrightarrow \varphi'$.

2. Индуктивный переход. Предположим, что теорема верна, если φ содержит n логических связок. Докажем, что из это-

го предположения можно заключить, что теорема верна, если в φ содержится $n+1$ логическая связка.

Рассмотрим отдельно два случая.

а) Пусть формула φ имеет вид $\neg\varphi_1$. По индуктивному предположению верно, что $\vdash \varphi_1 \leftrightarrow \varphi_1'$, где формула φ_1' получена из φ путем замены некоторых вхождений подформулы σ на формулу σ'. По теореме полноты тавтология $(C \leftrightarrow D) \to (\neg C \leftrightarrow \neg D)$ выводима в нашей аксиоматической системе, то есть

$$\vdash \neg\varphi_1 \leftrightarrow \neg\varphi_1', \quad \varphi \leftrightarrow \varphi'.$$

б) Пусть формула φ имеет вид $\varphi_1 \circ \varphi_2$, где символом $\circ$ обозначена одна из логических связок $\wedge, \vee, \to, \leftrightarrow$. Тогда по индуктивному предположению верно, что $\vdash \varphi_1 \leftrightarrow \neg\varphi_1'$ и $\vdash \varphi_2 \leftrightarrow \neg\varphi_2'$, где формулы φ' и φ'' получены из φ заменой некоторых вхождений подформулы σ на формулу σ'. Используя тавтологию $C \to (D \to (C \wedge D))$ и теорему полноты, мы получаем

$$\vdash (\varphi_1 \leftrightarrow \varphi_1') \wedge (\varphi_2 \leftrightarrow \varphi_2').$$

Результат следует из тавтологии

$$(C \leftrightarrow D_1) \wedge (C_2 \leftrightarrow D_2) \to (C_1 \circ C_2 \leftrightarrow D_1 \circ D_2).$$

1.14.23. Предполагая, что $\vdash (A \wedge B) \leftrightarrow \neg(A \to \neg B)$ и $\vdash (A \vee B) \leftrightarrow (\neg A \to B)$, выведите формулы:

а) $\vdash (\neg B \to \neg A) \leftrightarrow (A \to B)$;

б) $\vdash (\neg A \to B) \leftrightarrow (\neg B \to A)$ и $\vdash (A \to \neg B) \leftrightarrow (B \to \neg A)$;

в) $\vdash (A \wedge B) \to A$;

г) $\vdash (A \wedge \neg A) \to B$;

д) $\vdash A \vee \neg A$.

Р е ш е н и е. а) и б) Воспользуемся аксиомой 3, законом отрицания и теоремой о подстановке эквивалентных формул.

в) По аксиоме 1 верно, что

$$\vdash \neg A \to (B \to \neg A); \tag{1}$$

по (1), б) и теореме 1.8.4 получим

$$\vdash \neg A \to (A \to \neg B); \tag{2}$$

по (2), б) и теореме 1.8.4 имеем

$$\vdash \neg(A \to \neg B) \to A. \tag{3}$$

Но согласно нашим предположениям это — то, что мы искали, а именно $(A \wedge B) \to A$.

Обычно мы определяем шаги вывода аналитическим путем: сначала следует попытаться определить (3), затем определить (2), и продолжать до тех пор, пока не будет найдено высказывание, о котором известно, что оно выводимо, например (1).

1.14.24. Используя метод резолюции, докажите, что следующие множества формул противоречивы:

а) $S=\{\neg A\vee B\vee D, \neg B\vee D\vee A, \neg D\vee A, A\vee B, B\vee\neg D, \neg A\vee\neg B\}$;

б) $S=\{\neg A\vee B, \neg B\vee C, \neg C\vee A, A\vee C, \neg A\vee\neg C\}$;

в) $S=\{A\vee B\vee C, A\vee B\vee\neg C, A\vee\neg B, \neg A\vee\neg C, \neg A\vee C\}$.

Решение. в) Рассмотрим резолютивный вывод:

1) $A\vee B\vee C$;

2) $A\vee B\vee\neg C$;

3) $A\vee\neg B$;

4) $\neg A\vee\neg C$;

5) $\neg A\vee C$;

6) $A\vee B$ — резольвента 1 и 2;

7) A — резольвента 3 и 6;

8) $\neg A$ — резольвента 4 и 5;

9) $\square$ — резольвента 7 и 8.

Следовательно, множество S противоречиво.

1.14.25. Используя метод резолюции, докажите, что:

а) $\{A\to B,\ C\to D,\ D\to B,\ B\vee C\vee D\}\vdash A\wedge B$;

б) $\{A\wedge B\to C,\ A\to B\}\vdash A\to C$.

Решение. б) Нам достаточно доказать, что множество $\{A\wedge B\to C,\ A\to B,\ \neg(A\to C)\}$ противоречиво. Представим высказывания множества $\{A\wedge B\to C,\ A\to B,\ \neg(A\to C)\}$ в КНФ:

$A\wedge B\to C\leftrightarrow\neg(A\wedge B)\vee C\ \leftrightarrow\ \neg A\vee\neg B\vee C$,

$A\to B\leftrightarrow\neg A\vee B$,

$\neg(A\to C)\leftrightarrow\neg(\neg A\vee C)\ \leftrightarrow\ \neg\neg A\wedge\neg C\ \leftrightarrow\ A\wedge\neg C$.

Применим метод резолюции к множеству дизъюнктов $\{\neg A\vee\neg B\vee C,\ \neg A\vee B,\ A,\ \neg C\}$:

1) $\neg A\vee\neg B\vee C$;

2) $\neg A\vee B$;

3) A;

4) $\neg C$;

5) $\neg A\vee\neg B$ — резольвента 1 и 4;

6) $\neg B$ — резольвента из 3 и 5;

7) B — резольвента 3 и 2;

8) $\square$ — резольвента 3 и 2.

1.14.26. Представьте следующие высказывания как множества дизъюнктов:

$$\text{а)}\ \ \neg(A \wedge B \wedge \neg C); \qquad \text{б)}\ \ A \leftrightarrow (\neg B \wedge \neg C).$$

Р е ш е н и е. а) Определим КНФ высказывания $\neg(A \wedge B \wedge \neg C)$:

$$[\neg(A \wedge B \wedge \neg C)] \leftrightarrow [\neg A \vee \neg B \vee \neg\neg C] \leftrightarrow [\neg A \vee \neg B \vee C]$$

В результате получаем состоящее из одного дизъюнкта множество $\{\{\neg A, \neg B, C\}\}$.

1.14.27. Какие из следующих множеств дизъюнктов выполнимы и почему? Для каждого непротиворечивого множества определите означивание, которое подтверждает его выполнимость.

$$\text{а)}\ \{\{A, B\},\ \{\neg A, B\}\}, \qquad \text{в)}\ \{\{A\},\ \square\},$$
$$\text{б)}\ \{\{\neg A\},\ \{A, \neg B\},\ \{B\}\}, \qquad \text{г)}\ \{\square\}.$$

Р е ш е н и е. б) Соответствующая КНФ имеет вид $\neg A \wedge (A \vee \neg B) \wedge B$. Это высказывание логически ложно. Если бы оно было выполнимо, то существовало бы такое означивание W, что $W(B) = t$ (чтобы подтверждалось высказывание B) и $W(A) = f$ (чтобы подтверждалось высказывание A). Но это означивание высказывание $A \vee \neg B$ не подтверждает.

г) Пустой дизъюнкт $\square$ по определению невыполним.

1.14.28. Определите резольвенты $R(S)$ множества дизъюнктов S, если

а) $S = \{\{A, \neg B\},\ \{A, B\},\ \{\neg A\}\}$;

б) $S = \{\{A\},\ \{B\},\ \{A, B\}\}$.

Р е ш е н и е. а) Для множества дизъюнктов

$$S = \{\underbrace{\{A, \neg B\}}_{1}, \underbrace{\{A, B\}}_{2}, \underbrace{\{\neg A\}}_{3}\}$$

множество резольвент

$$R(S) = \{\underbrace{\{A\}}_{1,2}, \underbrace{\{\neg B\}}_{1,3}, \underbrace{\{B\}}_{2,3}\} \cup S.$$

б) В данном случае $R(S) = S$. Резольвент нет.

1.14.29. Дано высказывание Q: $(A \wedge B \to C) \wedge A \wedge (\neg B \to C)$.

а) Представьте Q в теоретико-множественном виде.

б) Докажите, что $Q \vdash_R C$.

в) Предположим, что высказывания Q и $A \to \neg C$ истинны. Что можно сказать об истинностном значении B?

Р е ш е н и е. а) $[A \wedge B \to C] \leftrightarrow [\neg(A \wedge B) \vee C] \leftrightarrow [\neg A \vee \neg B \vee C]$; $[\neg B \to C] \leftrightarrow [\neg\neg B \vee C] \leftrightarrow [B \vee C]$.

Следовательно, $Q = \{\{\neg A, \neg B, C\}, \{A\}, \{B, C\}\}$.

б) Рассмотрим резолютивный вывод:

1) $\{\neg A, \neg B, C\}$;

2) $\{A\}$;

3) $\{B, C\}$;

4) $\{\neg B, C\}$ — резольвента 1 и 2;

5) $\{C\}$ — резольвента 3 и 4.

в) Пусть $Q' = \{\{\neg A, \neg B, C\}, \{A\}, \{B, C\}, \{\neg A, \neg C\}, \{\neg B\}\}$. Множество Q' противоречиво (почему?). Таким образом, любое означивание, которое подтверждает все высказывания из Q и высказывание $A \to \neg C$, приписывает истинностное значение f высказыванию $\neg B$. Другими словами, оно подтверждает B.

1.14.30. Найдите означивания, опровергающие высказывания:

а) $(A \to B) \leftrightarrow (A \vee B)$; б) $(A \vee \neg B) \leftrightarrow (A \wedge B)$.

Р е ш е н и е. а) Построим семантическую таблицу с $f[(A \to B) \leftrightarrow (A \vee B)]$ в корне (см. рис. 1.6).

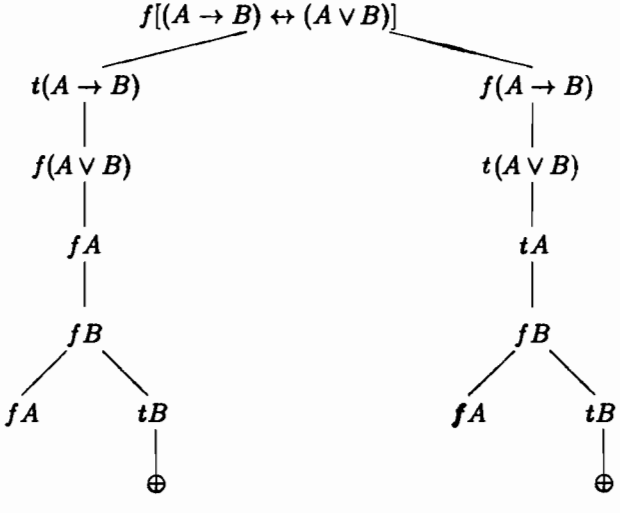


Рис. 1.6

Каждая непротиворечивая ветвь дает означивание W, которое согласуется с корнем семантической таблицы по лемме 1.10.5. Тогда для каждого W такого, что $W(A)=f$ и $W(B)=f$, мы имеем:

$$\begin{aligned} W[(A\to B)\leftrightsquigarrow(A\vee B)] &= W(A\to B)\leftrightarrow W(A\vee B)= \\ &= [W(A)\leftrightsquigarrow W(B)]\leftrightarrow[W(B)\sqcup W(B)]= \\ &= [f\rightsquigarrow f]\leftrightsquigarrow[f\sqcup f]= \\ &= t\leftrightarrow f = f. \end{aligned}$$

1.14.31. Доказать, что если A и $A\to B$ выводимы по Бету, то B также выводимо по Бету.

Решение. Поскольку A доказуемо по Бету, A тождественно истинно. Тогда для любого означивания W мы имеем $W(A)=t$. Высказывание $A\to B$ также тождественно истинно и для любого означивания W мы имеем $W(A\to B)=t$. Тогда

$$t = W(A\to B) = W(A)\rightsquigarrow W(B) = t\rightsquigarrow W(B).$$

Но это означает, что $W(B)=t$ для любого означивания W. Следовательно, B доказуемо по Бету.

1.14.32. Доказать:

а) $\{A\vee B, A\to C, B\to D\}\vdash C\vee D$;

б) $\{A\to(B\to C), \neg D\vee A, B\}\vdash D\to C$.

Решение. а) Рассмотрим $S=\{A\vee B, A\to C, B\to D\}$. По теоремам корректности и полноты мы имеем

$$S\models C\vee D \Leftrightarrow S\vdash C\vee D.$$

Таким образом, нам достаточно доказать $S\models C\vee D$, или, другими словами, что всякое означивание, подтверждающее S, подтверждает также $C\vee D$.

Итак, мы построили семантическую таблицу с $f(C\vee D)$ в корне (см. рис. 1.7), последовательно присоединив вершины $t(A\vee B)$, $t(A\to C))$ и $t(B\to D)$. Так как построенная семантическая таблица противоречива, то $C\vee D$ подтверждается всеми означиваниями, подтверждающими S (см. рис. 1.7).

1.14.33. Проверьте, верно ли, что:

а) $\{(A\to B)\to B,\ A\wedge\neg B\}\vdash A\to B$;

б) $\{A\to B,\ C\vee B,\ D\to(A\vee C),\ D\}\vdash B$.

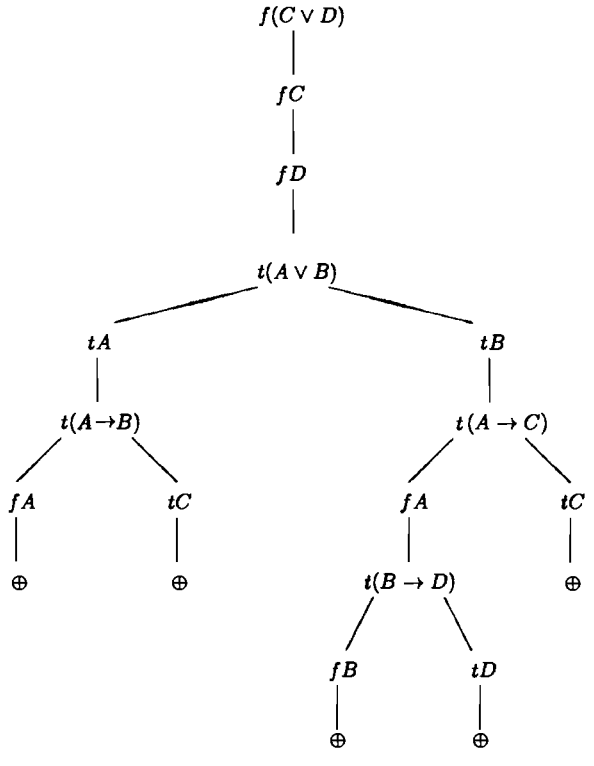


Рис. 1.7

Р е ш е н и е. а) Построим семантическую таблицу с $f(A \to B)$ в корне, последовательно присоединив гипотезы $[(A \to B) \to B]$ и $(A \wedge \neg B)$ (см. рис. 1.8).

Заметим, что $S \not\models A \to B$, поскольку в приведенной семантической таблице ветвь $\varkappa$ непротиворечива. Следовательно, существует такое означивание W, что $W(A) = t$ и $W(B) = f$, которое все высказывания из S подтверждает, в то время как высказывание $A \to B$ опровергает.

1.14.34. Докажите, что если $S \cup \{A\} \vdash B$, то $S \vdash A \to B$.

Р е ш е н и е. По теоремам корректности и полноты

$$S \cup \{A\} \vdash B \Leftrightarrow S \cup \{A\} \models B.$$

Предположим, что $S \cup \{A\} \models B$. Тогда для любого означивания W, подтверждающего все высказывания из S, верно $W(B) = t$, если

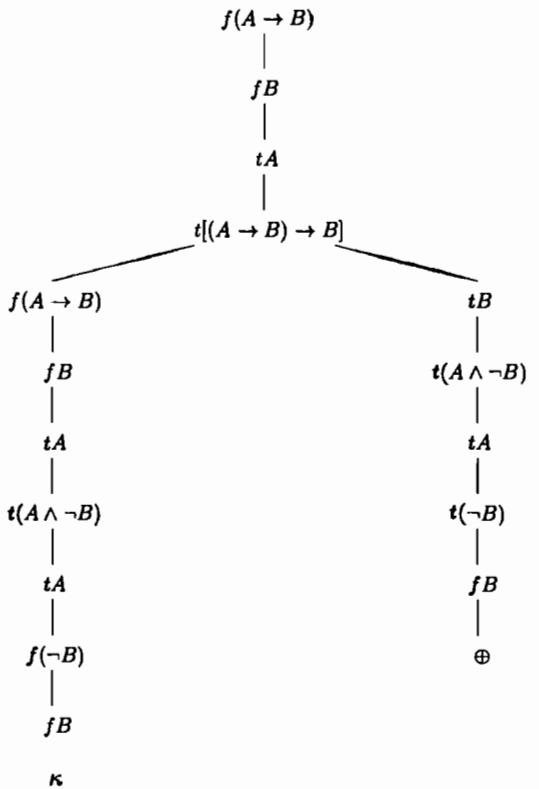


Рис. 1.8

$W(A \to B) = W(A) \rightsquigarrow W(B) = t$. Это следует из того, что $W(A \to \to B)$ может быть ложным только в том случае, если $W(A) = t$ и $W(B) = f$. Поэтому $S \models A \to B$. Значит, $S \cup \{A\} \vdash B \Rightarrow S \cup \{A\} \models B$. По теореме полноты $S \cup \{A\} \models B \Rightarrow S \cup \{A\} \vdash B$.

1.14.35. Пусть $\sigma_1, \sigma_2, \ldots$ — бесконечная последовательность высказываний. Докажите, что если для любого i высказывание $\sigma_{i+1} \to \to \sigma_i$ доказуемо по Бету, а $\sigma_i \to \sigma_{i+1}$ не является доказуемым по Бету, то не существует высказывания τ, для которого справедливы одновременно следующие утверждения:

а) Для каждого i высказывание $\tau \to \sigma_i$ выводимо по Бету;

б) $\{\sigma_1, \sigma_2, \ldots\} \models \tau$.

Р е ш е н и е. а) Докажем, что если для всякого i высказывание $\tau \to \sigma_i$ доказуемо по Бету, то $\{\sigma_1, \sigma_2, \ldots\} \not\models \tau$. То есть множество

высказываний $\{\neg\tau, \sigma_1, \sigma_2, \ldots\}$ непротиворечиво. Другими словами, существует такое означивание W, что:

$$W(\neg\tau) = W(\sigma_1) = W(\sigma_2) = \ldots = t.$$

Поскольку для любого i высказывание $\sigma_i \to \sigma_{i+1}$ не является доказуемым по Бету, для каждого натурального числа n *существует такое означивание* W, что:

$$W(\sigma_n \to \sigma_{n+1}) = f, \text{ или } W(\sigma_n) \rightsquigarrow W(\sigma_{n+1}) = f.$$

Тогда мы можем заключить, что

$$W(\sigma_n) = t, \qquad W(\sigma_{n+1}) = f. \tag{1}$$

Поскольку для каждого $i \in \mathbb{N}$ высказывание $\sigma_{i+1} \to \sigma_i$ доказуемо по Бету, *для любого означивания* W' и для любого $n \in \mathbb{N}$ справедливо $W'(\sigma_{n+1} \to \sigma_n) = t$, что равносильно $W'(\sigma_{n+1}) \rightsquigarrow W'(\sigma_n) = t$. Следовательно,

$$W'(\sigma_{n+1}) = f \quad \text{или} \quad W'(\sigma_n) = t.$$

Подобным **же образом** можно показать, что

$$\begin{gathered} W'(\sigma_n) = f \quad \text{или} \quad W'(\sigma_{n-1}) = t, \\ \ldots\ldots\ldots \\ W'(\sigma_2) = f \quad \text{или} \quad W'(\sigma_1) = t. \end{gathered} \tag{2}$$

Если $W' = W$, то из (1) вытекает, что $W(\sigma_n) = t$ и $W(\sigma_{n+1}) = f$. Тогда на основании (2) мы получим

$$W(\sigma_n) = W(\sigma_{n-1}) = \ldots = t. \tag{3}$$

Для каждого i высказывание $\tau \to \sigma_i$ доказуемо по Бету. Тогда

$$t = W(\tau \to \sigma_{n+1}) = W(\tau) \rightsquigarrow W(\sigma_{n+1}) = W(\tau) \rightsquigarrow f,$$

то есть $W(\tau) = f$. Тогда

$$W(\neg\tau) = \sim W(\tau) = t. \tag{4}$$

Поэтому существует такое означивание W, что (1), (3) и (4) верны. Следовательно, для всех $n \in \mathbb{N}$ множество $\{\neg\tau, \sigma_1, \sigma_2, \ldots, \sigma_n\}$ непротиворечиво. Согласно теореме компактности 1.11.10 это означает, что множество $\{\neg\tau, \sigma_1, \sigma_2, \ldots\}$ также непротиворечиво.

б) Предположим $\{\sigma_1, \sigma_2, \ldots\} \models \tau$. Покажем, что существует означивание W и натуральное число k, такие, что $W(\tau \to \sigma_k) = f$ или, что то же самое, $W(\tau) = t$ и $W(\sigma_k) = f$.

Так как $\{\sigma_1, \sigma_2, \ldots\} \models \tau$, множество высказываний $\{\neg\tau, \sigma_1, \sigma_2, \ldots\}$ противоречиво. По теореме компактности существует такое натуральное число n, что конечное подмножество $\{\neg\tau, \sigma_1, \sigma_2, \ldots\sigma_n\}$ невыполнимо.

При этом, однако, найдется такое означивание W, что

$$W(\sigma_n \to \sigma_{n+1}) = f.$$

Тогда $W(\sigma_n) \rightsquigarrow W(\sigma_{n+1}) = f$, то есть

$$W(\sigma_n) = t, \qquad W(\sigma_{n+1}) = f. \tag{5}$$

Поскольку $\sigma_n \to \sigma_{n-1}$ доказуемо по Бету,

$$t = W(\sigma_n \to \sigma_{n-1}) = W(\sigma_n) \rightsquigarrow W(\sigma_{n-1}),$$

что означает $W(\sigma_n) = f$ или $W(\sigma_{n-1}) = t$. Согласно (5) мы получим

$$W(\sigma_{n-1}) = t. \tag{6}$$

Таким же образом, опираясь на тот факт, что все высказывания $\sigma_i \to \sigma_{i-1}$, $1 < i < n-1$, доказуемы по Бету, мы получим

$$W(\sigma_{n-2}) = \ldots = W(\sigma_1) = t. \tag{7}$$

Аналогично, из (5), (6) и (7) следует, что означивание W обращает в истину все высказывания σ_i, $1 < i < n$. Но множество высказываний $\{\neg\tau, \sigma_1, \sigma_2, \ldots\sigma_n\}$ противоречиво. Значит $W(\neg\tau) = f$. Отсюда следует $W(\tau) = t$. Но в этом случае, согласно (5) получим

$$W(\tau \to \sigma_{n+1}) = W(\tau) \rightsquigarrow W(\sigma_{n+1}) = t \rightsquigarrow f = f.$$

Это означает, что $\tau \to \sigma_{n+1}$ не является доказуемым по Бету.

Глава 2

ЛОГИКА ПРЕДИКАТОВ

συνάψιες ὅλα καὶ οὐχ ὅλα, συμφερόμενον διαφερόμενον, συνᾶιδον διᾶιδον, καὶ ἐκ πάντων ἓν καὶ ἐξ ἑνὸς πάντα.

Связи: целое и не целое, соединяющееся и разнообразящееся, мелодичное и немелодичное, и из всего — единое, и из единого — все.

Гераклит.

§ 2.1. Введение

В главе 1 мы дали аналитическое описание формального языка PL, с помощью которого можно выражать как простые, так и составные утверждения. Далее мы изучали методы вывода заключений из некоторых совокупностей высказываний языка PL.

Хотя язык PL довольно богат, его возможности в формулировке утверждений и отношений ограничены. Возьмем в качестве примера следующее утверждение на русском языке:

S: «Если Джордж — человек, то Джордж смертен.»

Если символом A обозначить утверждение

«Джордж — человек»,

а символом B — утверждение

«Джордж смертен»,

тогда в контексте языка логики высказываний S будет иметь вид

$$S\colon A \to B.$$

Утверждение S выражает некоторые качества отдельного лица, а именно, Джорджа. Возникает следующий вопрос: как выразить подобные свойства других людей, например, Сократа или Питера? Можно было бы решить эту проблему путем введения такого количества пропозициональных символов, сколько существует различных людей. Однако, на практике это невозможно.

В этой главе мы опишем язык логики предикатов, который позволяет решать такие проблемы. Новыми элементами в этом языке будут *переменные* и *кванторы*.

Переменные

Рассмотрим утверждение S и предположим, что x — переменная, принимающая значение на множестве имен всех людей, например,

$$x = \text{Джордж}, \quad \text{или} \quad x = \text{Джон}, \quad \text{или} \quad x = \ldots,$$

а слова «человек» и «смертен» обозначают свойства. Тогда мы можем выразить общую связь между этими свойствами следующим образом:

$$P\text{: человек } (x) \rightarrow \text{смертен } (x).$$

Выражения человек (x) и смертен (x), обозначающие свойства, называются *предикатами*. Выражение P есть пример *формулы*, которая состоит из предикатов, соединенных логическими связками.

Подстановка вместо переменной x константы «Джордж» преобразует P в формулу

$$S'\text{: человек (Джордж)} \rightarrow \text{смертен (Джордж)}.$$

Далее, если переменная x примет значение «Сократ», результат будет представлять собой новую формулу, описывающую связь между Сократом и свойством быть смертным. Слова «Джон», «Джордж» и «Питер» являются *константами* в нашем новом формальном языке.

Вообще говоря, соответствие между переменными и константами с одной стороны и словами русского языка с другой можно представить неформальной схемой:

Русский язык		*Формальный язык*
местоимение	$\longmapsto$	переменная
собственное имя	$\longmapsto$	константа

Специальные слова «Человек» и «Смертен» называются *предикатными символами*. Предикаты могут зависеть более чем от одного аргумента, выражая таким образом не только свойства, но и отношения между несколькими объектами. Например, если переменные x и y принимают значения на множестве целых чисел, и введен предикат I — «больше», мы можем выразить одно из фундаментальных отношений между целыми числами:

$$I(x, y): \quad \text{больше}(x, y),$$

которое *интерпретируется* как «x больше, чем y».

Если в предыдущем выражении мы заменим x на 5 и y на 3, мы, очевидно, получим частный вид предиката I:

$$I(5, 3): \quad \text{больше}(5, 3),$$

который является истинным для данных чисел.

Кванторы

Введение переменных приводит к изменению истинности формулы. Рассмотрим в качестве примера формулу

$$Q(x, y): \quad \text{авиалиния}(x, y),$$

которая *интерпретируется* как «авиалиния соединяет города x и y». *Истинность* этой формулы является *частичной*, поскольку может, например, отсутствовать авиалиния, соединяющая Нью-Йорка и Нью-Дели. Напротив, формула

$$P(x): \text{человек } (x) \rightarrow \text{смертен } (x)$$

имеет *универсальную истинность*, будучи истинной для любого значения переменной x.

В логике предикатов (краткое обозначение PrL) всеобщая или частичная истинность обозначается двумя специальными символами — кванторами. Мы будем использовать *квантор всеобщности* и *квантор существования*, которые изображаются символами $\forall$ и $\exists$ соответственно. Таким образом, из формулы $P(x)$ получаем

$$P: (\forall x)\,(\text{человек } (x) \rightarrow \text{смертен } (x)),$$

а из формулы $Q(x, y)$ получаем

$$Q: (\exists(x, y))\ \text{авиалиния } (x, y).$$

В следующем параграфе мы введем язык логики предикатов формально.

§ 2.2. Язык логики предикатов

Теперь мы дадим формальное описание языка PrL. См. [Chur56, Curr63, Dela87, Hami78, Klee52, Mend64, Meta85, Smul68].

Определение 2.2.1. Язык PrL состоит из следующих основных символов:

I. Логические символы:

i) *переменные* $x, y, z, \ldots, x_0, y_0, z_0, \ldots, x_i, \ldots$;

ii) *логические связки* $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$;

iii) *запятая, скобки,* ( );

iv) *кванторы* $\forall, \exists$.

II. Специальные символы:

i) *предикатные символы* $P, Q, R, \ldots, P_0, Q_0, R_0, \ldots, P_1, \ldots$;

ii) *константы* $a, b, \ldots, a_0, b_0, \ldots, a_1, \ldots, a_2, \ldots$;

iii) *функциональные символы* $f, g, f_0, g_0, f_1, \ldots$ □ 2.2.1

Число аргументов предиката называется *степенью* или *местностью* предиката. Например, $Q(x, y, z)$ — предикат степени 3 или трехместный предикат.

Кванторы всеобщности и существования *двойственны* друг другу: квантор $\forall$ эквивалентен последовательности символов $\neg\exists\neg$, а квантор $\exists$ эквивалентен последовательности $\neg\forall\neg$. Например, для формулы $(\forall x)Q(x)$ мы имеем:

$$(\forall x)Q(x) \leftrightarrow \neg(\exists x)\neg Q(x).$$

Мы будем полагать, что каждый конкретный язык логики предикатов содержит все логические символы. Таким образом, для определения языка достаточно описать его специальные символы.

Пример 2.2.2. Язык арифметики имеет вид $\mathcal{L}_A = (=, \leqslant, +, *, 0, 1)$, где:

$=$ и $\leqslant$ — двухместные предикатные символы: $=(x, y)$ понимается как «$x = y$», и $\leqslant(x, y)$ обозначает «$x \leqslant y$»;

$+$ и $*$ — трехместные предикаты: $+(x, y, z)$ понимается как «$x + y = z$», и $*(x, y, z)$ понимается как «$x * y = z$»;

0 и 1 — символы констант. □ 2.2.2

Определение 2.2.3. *Терм* определяется индуктивно:

i) *константа* есть терм;

ii) *переменная* есть терм;

iii) если f — n-местный *функциональный символ*, и $t_1, \ldots, t_n$ — термы, тогда $f(t_1, \ldots, t_n)$ — терм. □ 2.2.3

Определение 2.2.4. *Элементарной формулой* или *атомом* называется всякое выражение вида $P(t_1, \ldots, t_n)$, где P — n-местный предикатный символ, а t_i — терм для каждого $i = 1, 2, \ldots, n$. □ 2.2.4

Определение 2.2.5. *Формула* определяется индуктивно:

i) каждый *атом* является формулой;

ii) если σ_1, σ_2 — формулы, тогда $(\sigma_1 \wedge \sigma_2)$, $(\sigma_1 \vee \sigma_2)$, $(\sigma_1 \rightarrow \sigma_2)$, $(\neg\sigma_1)$ и $(\sigma_1 \leftrightarrow \sigma_2)$ также являются формулами;

iii) если v — переменная, а φ — формула, тогда $((\exists v)\varphi)$, $((\forall v)\varphi)$ — формулы;

iv) только выражения, полученные согласно пунктам i), ii) и iii), являются формулами. □ 2.2.5

Пример 2.2.6. Следующие выражения являются формулами:

i) φ_1: $(\forall y)(\exists x)[P(x, f(y)) \vee Q(x)]$;

ii) φ_2: $(\forall x)(\exists y)[P(x) \vee Q(x, y) \to \neg(R(x))]$. □ 2.2.6

Замечание 2.2.7. Отметим, что в определении формулы нашего языка разрешается тривиальное использование кванторов, например: $(\exists x)[y = 3]$, что эквивалентно формуле $y = 3$. Хотя такое использование кванторов допустимо, обычно оно бывает нужным только в технических доказательствах. □ 2.2.7

Пример 2.2.8. Приведем несколько формул языка $\mathcal{L}_A$, определенного в примере 2.2.2.

$1)(\forall x)(x = x)$	рефлексивность
$2)(\forall x)(\forall y)(x = y \to y = x)$	симметричность
$3)(\forall x)(\forall y)(\forall v)[(x = y \wedge y = v) \to x = v]$	транзитивность
$4)(\forall x)(x \leqslant x)$	рефлексивность

□ 2.2.8

Приведем еще несколько определений, необходимых для полного описания языка PrL и логического программирования.

Определение 2.2.9. 1) Выражение t_1, входящее в состав терма t, которое в свою очередь является термом, называется *подтермом* t.

2) Выражение φ_1, входящее в состав формулы φ, которое в свою очередь является формулой, называется *подформулой* φ. □ 2.2.9

Пример 2.2.10. 1. Если $f(x, y)$ — терм, то x, y и $f(x, y)$ — его подтермы.

2. $P(x)$, $\neg R(x)$, $P(x) \vee Q(x, y)$ являются подформулами формулы φ_2 из примера 2.2.6. □ 2.2.10

Замечание 2.2.11. В главе, посвященной PL, мы изучали высказывания только в соответствии с их структурой, в основу которой были положены элементарные высказывания и логические связки. Тем самым, мы предполагали, что элементарные высказывания не нуждаются в дальнейшем анализе. Логика предикатов, однако, имеет дело с более общим понятием — элементарной формулой. Здесь мы предполагаем, что элементарные формулы представляют собой предикаты, и что каждый n-местный предикат $P(t_1, \ldots, t_n)$ выражает отношение между термами $t_1, \ldots, t_n$. □ 2.2.11

Определение 2.2.12 (связанное и свободное вхождение переменных). 1. Вхождение переменной v в формулу φ называется *связанным*, если существует подформула ψ формулы φ, которая содержит это вхождение и имеет вид $((\forall v)\psi')$ или $((\exists v)\psi')$.

2. Вхождение переменной v в формулу называется *свободным*, если оно не является связанным. □ 2.2.12

Определение 2.2.13 (свободные и связанные переменные). Переменная v, входящая в формулу φ называется *свободной*, если она имеет по крайней мере одно свободное вхождение в формулу φ. Переменная v называется *связанной*, если она не является свободной. □ 2.2.13

Пример 2.2.14. В примере 2.2.6 переменная x имеет свободное вхождение в подформулу

$$\varphi'\colon (\exists y)\,(P(x) \vee Q(x, y) \to \neg R(x))$$

формулы φ_2, однако, в самой формуле φ_2 это вхождение связанное. Следовательно, переменная x свободна в формуле φ' (так как она имеет по крайней мере одно свободное вхождение), но связана в формуле φ_2. □ 2.2.14

Определение 2.2.15. Терм, в который не входит ни одна переменная, называется *основным термом.* □ 2.2.15

Пример 2.2.16. Если a и b — константы, а f — функциональный символ, тогда a, b, $f(a, b)$, $f(f(a, b), b), \ldots$ — основные термы. □ 2.2.16

Определение 2.2.17. *Предложение* или *замкнутая формула* есть формула, не содержащая ни одной свободной переменной. □ 2.2.17

В соответствии с этим определением, для того, чтобы из заданной формулы получить замкнутую формулу, нужно связать все ее свободные переменные с помощью кванторов.

Пример 2.2.18. Из формулы $\varphi(x, y)$: $(x + y = x * y)$ мы можем получить замкнутую формулу

$$\sigma(x, y)\colon\ (\forall x)\,(\exists y)\,(x + y = x * y). \qquad \square\ 2.2.18$$

Другой способ, позволяющий получать предложения, состоит в подстановке констант на места свободных вхождений переменных. В общем случае, мы имеем следующее определение подстановки [Fitt90].

Определение 2.2.19. *Подстановочное множество* или просто *подстановка* есть множество

$$\theta = \{x_1/t_1,\ x_2/t_2, \ldots,\ x_n/t_n\},$$

где x_i и t_i, $1 \leqslant i \leqslant n$, являются соответетственно переменными и термами, причем равенство $x_i = x_j$ влечет $t_i = t_j$, $1 \leqslant i \leqslant j \leqslant n$.

Если φ есть какое-либо выражение (атом, терм или формула), то $\varphi\theta$ обозначает выражение, полученное путем подстановки на места свободных вхождений переменных $x_1, \ldots, x_n$ соответствующих термов $t_1, \ldots, t_n$.

Пустая подстановка обозначается символом E, другими словами, $E = \{\ \}$. □ 2.2.19

Пример 2.2.20. Если мы применим подстановку $\theta = \{x/2, y/2\}$ к формуле K: $\varphi(x, y)$ из предыдущего примера, то получим формулу $K\theta$: $(2 + 2 = 2 * 2)$. □ 2.2.20

Пример 2.2.21. Рассмотрим формулу

$$\varphi(x, y, z):\ (\exists y)R(x, y) \wedge (\forall z)(\neg Q(x, z))$$

и основной терм $f(a, b)$. Применяя подстановку $\{x/f(a, b)\}$ к формуле $\varphi(x, y, z)$, получим предложение

$$\varphi(f(a, b), y, z):\ (\exists y)R(f(a, b), y) \wedge (\forall z)(\neg Q(f(a, b), z).$$ □ 2.2.21

Основной операцией на множестве подстановок является композиция.

Определение 2.2.22. Пусть

$$\theta = \{u_1/s_1, \ldots, u_m/s_m\} \quad \text{и} \quad \psi = \{v_1/t_1, \ldots, v_n/t_n\}.$$

Композицией θ и ψ называется подстановка

$$\theta\psi = \{u_1/s_1\psi, \ldots, u_n/s_n\psi, v_1/t_1, \ldots, v_n/t_n\} - (\{u_i/s_i\psi \mid u_i = s_i\psi\} \cup \{v_i/t_i \mid v_i \in \{u_1, \ldots, u_m\}\}).$$

□ 2.2.22

Другими словами, мы сначала применяем подстановку ψ к термам $s_1, \ldots, s_m$ подстановки θ, заменяя в этих термах переменные v_i на термы t_i (определение 2.2.19), а затем дополняем полученную подстановку элементами множества ψ. При этом мы отбрасываем элементы вида $u_i/s_i\psi$, если терм $s_i\psi$ совпадает с u_i, и элементы вида v_i/t_i, если v_i содержится среди переменных $u_1, \ldots, u_m$ подстановки θ.

Пример 2.2.23. 1. Рассмотрим две подстановки: $\theta = \{x/f(y), y/z\}$ (в качестве $\{u_i/s_i\}$) и $\psi = \{x/a, y/b, z/y\}$ (в качестве $\{v_i/t_i\}$). Согласно определению 2.2.23 композиция θ и ψ имеет вид

$$\begin{aligned}\theta\psi &= \{x/f(y)\psi, y/z\psi, x/a, y/b, z/y\} \\ &\quad - (\{u_i/s_i \mid u_i = s_i\psi\} \cup \{v_i/t_i \mid v_i \in \{u_j\}\}) \\ &= \{x/f(b), y/y, x/a, y/b, z/y\} - (\{y/y\} \cup \{x/a, y/b\}) \\ &= \{x/f(b), z/y\}.\end{aligned}$$

2. Рассмотрим терм t: $w(f(v_1), h(x), f(v_2), v_3)$ и подстановки

$$\theta = \{v_1/f(g(x)), v_2/h(v_1), v_3/h(v_3)\}, \quad \psi = \{x/z, v_1/v_2, v_3/v_1\}.$$

Имеем

$$\begin{aligned}\theta\psi &= \{v_1/f(g(x))\psi, v_2/h(v_1)\psi, v_3/h(v_3)\psi, x/z, v_1/v_2, v_3/v_1\} \\ &\quad - (\varnothing \cup \{v_1/v_2, v_3/v_1\}) \\ &= \{v_1/f(g(z)), v_2/h(v_2), v_3/h(v_1), x/z, v_1/v_2, v_3/v_1\} \\ &\quad - \{v_1/v_2, v_3/v_1\} \\ &= \{v_1/f(g(z)), v_2/h(v_2), v_3/h(v_1), x/z\}.\end{aligned}$$

Следовательно,

$$t(\theta\psi): \ w(f(f(g(z))), h(z), f(h(v_2)), h(v_1)).$$

Отметим также, что

$$\begin{aligned}(t\theta)\psi &= w(f(f(g(x))), h(x), f(h(v_1)), h(v_3))\psi \\ &= w(f(f(g(z))), h(z), f(h(v_2)), h(v_1)) \\ &= t(\theta\psi)\end{aligned}$$ □ 2.2.23

Таким образом, мы можем заключить, что для композиции подстановок из примера 2.2.23 выполняется свойство ассоциативности. В общем случае, верна следующая теорема.

Теорема 2.2.24. *Для произвольных подстановок θ, ψ, γ и для произвольного терма t выполняются равенства:*

1) $\theta E = E\theta = \theta$; 2) $(t\theta)\psi = t(\theta\psi)$; 3) $(\theta\psi)\gamma = \theta(\psi\gamma)$. □ 2.2.24

Определение 2.2.25. Пусть φ — некоторая формула без кванторов, и θ — подстановка. Запись $\varphi\theta$ обозначает результат замены каждого терма t в формуле φ на $t\theta$.

Соответственно, если $S = \{C_1, \ldots, C_k\}$ — множество формул PrL без кванторов, то множество $S\theta = \{C_1\theta, \ldots, C_k\theta\}$ есть результат замены формул C_i, $1 \leqslant i \leqslant k$, на формулы $C_i\theta$. □ 2.2.25

Определение 2.2.26. Два множества формул без кванторов S_1 и S_2 называются *вариантами*, если можно подобрать такие две подстановки θ и ψ, что

$$S_1 = S_2\theta, \quad \text{а} \quad S_2 = S_1\psi.$$ □ 2.2.26

Пример 2.2.27. Множества $S_1 = \{P(f(x, y)), Q(h(z), b)\}$ и $S_2 = \{P(f(y, x), Q(h(u), b)\}$, где b — константа, являются вариантами. Действительно, если $\theta = \{x/y, y/x, z/u\}$ и $\psi = \{x/y, y/x, u/z\}$, то

$$S_1\theta = \{P(f(x, y))\theta, Q(h(z), b)\theta\} = \{P(f(y, x)), Q(h(u), b)\} = S_2,$$
$$S_2\theta = \{P(f(y, x))\theta, Q(h(u), b)\theta\} = \{P(f(x, y)), Q(h(z), b)\} = S_1.$$
□ 2.2.27

Определение 2.2.28. *Переименованием* называется подстановка вида

$$\{v_1/u_1, \ldots, v_n/u_n\},$$

где все u_i, $1 \leqslant i \leqslant n$, являются переменными. □ 2.2.28

§ 2.3. Аксиоматическое основание логики предикатов

В § 1.7 гл. 1 мы аксиоматизировали логику высказываний посредством системы, состоящей из трех аксиом и одного правила вывода. Подобным образом можно аксиоматизировать и логику предикатов [Dela87, Hami78, Klee52, Mend64, Rasi74, RaSi70]. Сначала введем вспомогательное определение.

Определение 2.3.1. *Переменная x свободна для терма t в формуле σ* (формальная запись $\mathit{free}(x, t, \sigma)$), если никакая из переменных терма t не становится связанной после подстановки терма t на все места свободных вхождений переменной x в формуле σ. □ 2.3.1

Пример 2.3.2. Предположим, что $\sigma = (\forall y)P(x, y)$. Тогда переменная x не свободна для терма y в формуле σ, поскольку после подстановки терма y на места свободных вхождений переменной x переменная y становится связанной.

С другой стороны, переменная x свободна для терма z в формуле σ, если z есть переменная отличная от y, так как после замены переменной x на терм z в формуле σ переменная z, единственная переменная терма z, не становится связанной. Более того, y свободна для y в формуле σ (так как σ не содержит свободных вхождений переменной y). □ 2.3.2

Определение 2.3.3. Для произвольных формул PrL φ, τ, σ следующие формулы являются аксиомами:

1) $\varphi \to (\tau \to \varphi)$;

2) $(\varphi \to (\tau \to \sigma)) \to ((\varphi \to \tau) \to (\varphi \to \sigma))$;

3) $(\neg\varphi \to \neg\tau) \to (\tau \to \varphi)$;

4) если $\mathit{free}(x, t, \varphi)$, то аксиомой является формула

$$(\forall x)\varphi \to \varphi(x/t);$$

5) если переменная x не свободна в формуле φ, аксиомой является формула

$$(\forall x)(\varphi \to \tau) \to (\varphi \to (\forall x)\tau)).$$

Точно так же, как и в PL, символ $\vdash$ обозначает выводимость формул в аксиоматической системе. Данная система аксиом содержит два правила:

1) Modus Ponens: $\varphi,\ \varphi \to \tau \vdash \tau$;

2) обобщение: $\varphi \vdash (\forall x)\varphi$. □ 2.3.3

Замечание 2.3.4. 1. Согласно правилу обобщения, если формула φ выводима из аксиом с помощью правил PrL, то формула $(\forall x)\varphi$ также выводима в этой аксиоматической системе.

В качестве примера предположим, что выводима следующая формула:

$$\text{человек}\ (x) \rightarrow\ \text{смертен}\ (x).$$

Тогда формула

$$(\forall x)(\ \text{Человек}\ (x) \rightarrow\ \text{Смертен}\ (x))$$

также выводима.

Другими словами, мы всегда можем обосновать обобщенную формулу $(\forall x)\varphi$, опираясь на истинность формулы φ. Некорректное применение правила обобщения в обычных спорах часто вызывает множество ошибок. Например, мы часто утверждаем, что

«все политики — обманщики»,

потому что мы знаем, что политики a и b — обманщики. Однако, это утверждение логически неверно: чтобы обобщить утверждение на всех политиков, а не только на a и b, мы должны быть уверены, что следующая формула выводима в нашей системе аксиом:

$$\text{политик}\ (x) \rightarrow\ \text{обманщик}\ (x),$$

что (хочется надеяться) неверно.

2. Сравнивая аксиоматические системы для PL и PrL, можно заметить, что аксиомы и правило для PL содержатся среди аксиом и правил для PrL. Однако, логика высказываний имеет дело с высказываниями, тогда как логика предикатов рассматривает более сложные объекты — формулы логики предикатов.

3. Квантор $\exists$ не включен в систему аксиом, так как он выражается посредством $\neg\forall\neg$, как было сказано в разделе 2.1.

4. Опираясь на пункт 2) следствия 1.12.2, мы можем заключить, что все тавтологии PL выводимы в логике предикатов, если вместо высказываний в них подставить формулы PrL. Например, высказыванию $A \leftrightarrow \neg\neg A$, выводимому в PL, соответствует формула PrL $\varphi \leftrightarrow \neg\neg\varphi$, которая выводима в аксиоматической системе, описанной в определении 2.3.3. Таким образом, мы можем применить все тавтологии PL к формулам PrL (в соответствии с теоремой о полноте 1.12.1) и получить формулы PrL, выводимые в аксиоматической системе для PrL. □ 2.3.4

В PrL справедлива теорема об эквивалентной замене, подобная теореме для PL. Ее доказательство аналогично доказательству соответствующей теоремы в PL.

Т е о р е м а 2.3.5 (теорема об эквивалентной замене для PrL). *Если формула A_1 получена из формулы A путем замены некоторых вхождений подформулы B на формулу B_1, и выполняется*

$$\vdash (\forall x_1) \dots (\forall x_n)(B \leftrightarrow B_1),$$

где $x_1, \dots, x_n$ — свободные переменные формул B и B_1, являющиеся в то же время связанными в A, тогда

$$\vdash A \leftrightarrow A_1. \qquad \square\ 2.3.5$$

В контексте PrL мы можем работать только с теми «правильными» формулами, которые выводимы в аксиоматической системе PrL. Следующая теорема дает нам список наиболее часто используемых формул. Эти формулы выражают коммутативность кванторов и дистрибутивность кванторов относительно логических связок. Как показано в теореме, эти свойства выполняются не всегда.

Т е о р е м а 2.3.6. *Пусть φ и σ — произвольные формулы PrL. Тогда следующие формулы выводимы в PrL:*

$$(\forall x)(\varphi \to \sigma) \to ((\forall x)\varphi \to (\forall x)\sigma);$$
$$((\forall x)\varphi \to (\forall x)\sigma) \to (\exists x)(\varphi \to \sigma);$$
$$((\exists x)\varphi \to (\exists x)\sigma) \to (\exists x)(\varphi \to \sigma);$$
$$(\exists x)(\varphi \to \sigma) \leftrightarrow ((\forall x)\varphi \to (\exists x)\sigma);$$
$$((\forall x)\varphi \vee (\forall x)\sigma) \to (\forall x)(\varphi \vee \sigma);$$
$$(\forall x)(\varphi \vee \sigma) \to ((\exists x)\varphi \vee (\forall x)\sigma);$$
$$(\exists x)(\varphi \vee \sigma) \leftrightarrow ((\exists x)\varphi \vee (\exists x)\sigma);$$
$$(\exists x)(\varphi \wedge \sigma) \to ((\exists x)\varphi \wedge (\exists x)\sigma);$$
$$(\forall x)(\varphi \wedge \sigma) \to ((\forall x)\varphi \wedge (\exists x)\sigma);$$
$$(\forall x)(\varphi \wedge \sigma) \leftrightarrow ((\forall x)\varphi \wedge (\forall x)\sigma);$$
$$(\exists y)(\forall x)\varphi \to (\forall x)(\exists y)\varphi;$$
$$(\forall x)(\forall y)\varphi \leftrightarrow (\forall y)(\forall x)\varphi;$$
$$(\exists x)(\exists y)\varphi \leftrightarrow (\exists y)(\exists x)\varphi;$$
$$(\forall x)\varphi \leftrightarrow \varphi, \quad \textit{если } x \textit{ не входит свободно в } \varphi;$$
$$(\exists x)\varphi \leftrightarrow \varphi, \quad \textit{если } x \textit{ не входит свободно в } \varphi. \qquad \square\ 2.3.6$$

Одной из ошибок, часто встречающихся в формальных научных доказательствах, является неправильное использование дистрибутивности кванторов относительно логических связок. Например,

формулы

$$((\forall x)\varphi \vee (\forall x)\sigma) \to (\forall x)(\varphi \vee \sigma)$$

и

$$(\exists x)(\varphi \wedge \sigma) \to ((\exists x)\varphi \wedge (\exists x)\sigma),$$

взятые из списка предыдущей теоремы, выводимы в **PrL**. Но формулы

$$((\forall x)\varphi \vee (\forall x)\sigma) \leftrightarrow (\forall x)(\varphi \vee \sigma)$$

и

$$(\exists x)(\varphi \wedge \sigma) \leftrightarrow ((\exists x)\varphi \wedge (\exists x)\sigma),$$

выражающие полную дистрибутивность кванторов $\forall$ и $\exists$ относительно связок $\vee$ и $\wedge$ соответственно, не являются истинными. Действительно, формулы

$$(\forall x)(\varphi \vee \sigma) \to ((\forall x)\varphi \vee (\forall x)\sigma)$$

и

$$((\exists x)\varphi \wedge (\exists x)\sigma) \to (\exists x)(\varphi \wedge \sigma)$$

НЕ ВЫВОДИМЫ в аксиоматической системе PrL и НЕВЕРНЫ.

Например, формула

$$(\forall x)[(x = 2x) \vee (x \neq 2x)]$$

истинна (см. определение 2.4.2).

Однако, формула

$$[(\forall x)(x = 2x)] \vee [(\forall x)(x \neq 2x)]$$

НЕВЕРНА. Если бы она была истинной, по крайней мере одна из формул

$$[(\forall x)(x = 2x)], \quad [(\forall x)(x \neq 2x)]$$

была бы истинной. Но это не так: при $x = 1$ неверно $x = 2x$, а при $x = 0$ неверно $x \neq 2x$. Таким образом, нужно проявлять осторожность при использовании коммутативности и дистрибутивности кванторов, иначе это может легко привести к неверным заключениям.

Замечание 2.3.7. Если бы мы захотели расширить язык PrL особым двухместным логическим символом равенства «=», то некоторые свойства равенства мы бы не смогли выразить посредством системы аксиом из определения 2.3.3. Эта аксиоматизация выражает общие свойства предикатов и не может описать свойства отдельных предикатов, таких как «=». Например, отношение равенства должно быть рефлексивным, симметричным и транзитивным. Для того, чтобы аксиоматически описать эти свойства, нужно расширить систему аксиом из определения 2.3.3 двумя аксиомами [Dela87, Hami78, Mend64, Schw71].

6) Для произвольного терма x формула

$$x = x$$

является аксиомой.

7) Если формула A_1 получена из формулы A заменой одного или нескольких вхождений терма x на терм y, тогда формула

$$(x = y) \to (A \leftrightarrow A_1)$$

является аксиомой.

Таким образом, мы аксиоматизировали рефлексивность равенства и правило замены равных термов. Свойства симметричности и транзитивности равенства выводимы из аксиом 1)–7) с помощью правил обобщения и Modus Ponens. □ 2.3.7

Доказательство теоремы 2.3.6, а также теорем корректности и полноты аксиоматической системы из определения 2.3.3, с равенством или без равенства, выходит за рамки этой книги. Читатель может их найти, например, в [Klee52, Mend64, RaSi70].

Определение 2.3.8. i) Для некоторого, возможно пустого, множества формул S и произвольной формулы A *выводом* A *из* S называется конечная последовательность формул логики предикатов $B_1, \ldots, B_k$, где каждая формула B_i, $1 \leqslant i \leqslant k$, либо является аксиомой, либо принадлежит S, либо получена из некоторых формул B_j, B_l, $1 \leqslant j,\ l \leqslant i$ по одному из правил — Modus Ponens или обобщения.

ii) Формула A *выводима из множества* S (обозначение $S \vdash A$), если существует вывод A из S.

iii) Формула A *выводима*, если существует вывод A из пустого множества. □ 2.3.8

В логике предикатов большой интерес вызывает теорема дедукции. Применение соответствующей теоремы для логики высказываний, т. е. теоремы 1.8.7, к формулам PrL может привести к неожиданным результатам.

Пусть мы знаем, что в мире есть богатые люди:

$$\text{богатый } (x).$$

Тогда по правилу обобщения мы имеем:

$$\text{богатый } (x) \vdash (\forall x) \text{ богатый } (x),$$

и по «соответствующей» теореме дедукции для PrL заключаем:

$$\text{богатый } (x) \to (\forall x) \text{ богатый } (x).$$

Иначе говоря, если есть богатые люди, то все люди богаты, что, конечно, не соответствует действительности. Поэтому для того, чтобы теорема дедукции порождала правильные заключения, она должна содержать ограничения на использование правила обобщения.

Теорема 2.3.9 (теорема дедукции). *Пусть S — множество формул PrL; A, B — такие формулы PrL, что $S \cup \{A\} \vdash B$, и правило обобщения не применялось в процессе вывода B из $S \cup \{A\}$ к свободным переменным формулы A. Тогда $S \vdash A \to B$.*

□ 2.3.9

Доказательство теоремы дедукции для PrL аналогично доказательству соответствующей теоремы для PL.

Так как наша аксиоматическая система содержит правило Modus Ponens, верно утверждение, обратное теореме дедукции. Отсюда

$$S \vdash A \to B \Leftrightarrow S \cup \{A\} \vdash B.$$

§ 2.4. Система обозначений логического программирования

В параграфе 1.9 мы определили некоторые основные понятия логического программирования в рамках логики высказываний. Теперь мы обобщим эти определения на случай логики предикатов [ChLe73, Dela87].

Определение 2.4.1. i) *Литералом* называется произвольный атом (определение 2.2.4) или его отрицание.

ii) Выражение вида

$$(\forall x_1)(\forall x_2)\ldots(\forall x_k)(C_1 \vee C_2 \vee \ldots \vee C_n),$$

где C_i, $i = 1, \ldots, n$ — литералы, и $x_1, \ldots, x_k$ — все переменные, встречающиеся в C_i, $1 \leqslant i \leqslant n$, называется *дизъюнктом*. При $n = 0$ мы имеем пустой дизъюнкт, который будем обозначать символом □.

□ 2.4.1

Мы будем опускать скобки, если расположение в формуле переменных и кванторов не вызывает сомнения.

Дизъюнкт может быть представлен в одном из следующих видов:

а) $\forall x_1 \ldots \forall x_k (A_1 \vee \ldots A_m \vee \neg B_1 \vee \ldots \vee \neg B_l)$;

б) $\forall x_1 \ldots \forall x_k (A_1 \vee \ldots A_m \leftarrow B_1 \wedge \ldots \wedge B_l)$;

в) $\forall x_1 \ldots \forall x_k (A_1, \ldots, A_m \leftarrow B_1, \ldots, B_l)$;

г) $\{C_1, C_2, \ldots, C_n\}$ — теоретико-множественное представление, где для всех i, $1 \leqslant i \leqslant n$, $C_i = A_j$, $1 \leqslant j \leqslant m$, или $C_j = \neg B_j$, $1 \leqslant j \leqslant n$;

д) $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_l$.

По определению 2.2.17 каждый дизъюнкт является предложением. А вот обратное неверно из-за возможного вхождения в предложение квантора $\exists$. Однако, логическое программирование имеет дело со всеми возможными предложениями PrL. Решение этой проблемы будет описано в параграфе 2.7, где мы будем изучать сколемовские формы.

Пример 2.4.2. Следующие выражения являются дизъюнктами:

i) $\forall x \forall y \forall z\,(P(x) \vee \neg Q(x, y) \vee R(x, y, z))$;

ii) $\forall x \forall y\,(\neg P(f(x, y), a) \vee Q(x, y))$. □ 2.4.2

Определение 2.4.3. Предложение, полученное из дизъюнкта φ путем удаления кванторов и подстановки вместо всех переменных констант, называется *основным примером* φ. □ 2.4.3

Например, предложение

$$P(a) \vee Q(b) \vee \neg R(a, b)$$

является основным примером дизъюнкта

$$\forall x \forall y\,(P(x) \vee Q(y) \vee \neg R(x, y)).$$

Определение 2.4.4. *Хорновским дизъюнктом* называется дизъюнкт вида:

$$\forall x_1 \ldots \forall x_k\,(A \leftarrow B_1, \ldots, B_l),$$

где $A, B_1, \ldots, B_l$ — атомы и $l \geqslant 1$. Атомы B_i, $i = 1, \ldots, l$, называются предпосылками хорновского дизъюнкта, а A — заключением. □ 2.4.4

Определение 2.4.5. *Целевым утверждением* или *целью* называется хорновский дизъюнкт, не содержащий заключения, т. е., дизъюнкт вида:

$$\forall x_1 \ldots \forall x_k\,(\leftarrow B_1, \ldots, B_l),$$

Атомы B_i называются *подцелями* данной цели. □ 2.4.5

Интуитивный смысл понятия цели станет ясным, если мы запишем это утверждение в виде формулы PrL и используем двойственность кванторов $\forall$ и $\exists$, а также правило де Моргана (см. замечание 2.3.4 (3)):

$$\forall x_1 \ldots \forall x_k\,(\neg B_1 \vee \ldots \vee \neg B_l) \leftrightarrow \forall x_1 \ldots \forall x_k \neg (B_1 \wedge \ldots \wedge B_l)$$
$$\leftrightarrow \neg (\exists x_1 \ldots \exists x_k)\,(B_1 \wedge \ldots \wedge B_l)$$

Другими словами, здесь утверждается, что нельзя подобрать такие значения $x_1, \ldots, x_k$, чтобы все предпосылки $B_1, \ldots, B_l$ стали истинными.

Определение 2.4.6. *Фактом* называется хорновский дизъюнкт, не содержащий ни одной предпосылки, т. е. дизъюнкт вида:

$$\forall x_1 \ldots \forall x_k\,(A \leftarrow). \qquad \square\ 2.4.6$$

Замечание 2.4.7. В следующем разделе мы будем изучать предложения логики предикатов, т. е. формулы без свободных переменных (определения 2.2.13 и 2.2.17).

Наш интерес к предложениям вызван двумя причинами.

1. Дизъюнкты (определения 2.4.1, 2.4.4, 2.4.4 и 2.4.6), используемые в логическом программировании, являются предложениями PrL, в которых все переменные связаны кванторами всеобщности.

2. Для каждой формулы φ логики предикатов с множеством свободных переменных $\{x_1, \ldots, x_k\}$ мы имеем:

$$\vdash \varphi \Leftrightarrow \vdash (\forall x_1) \ldots (\forall x_k)\varphi$$

(где $(\Rightarrow)$ выполняется в соответствии с правилом обобщения, а $(\Leftarrow)$ получаем, применяя аксиому 4) и правило Modus Ponens).

Таким образом, мы можем изучать предложение $(\forall x_1) \ldots (\forall x_k)\,\varphi$ вместо формулы φ. $\square$ 2.4.7

Определение 2.4.8. *Программой* называется конечное множество хорновских дизъюнктов. $\square$ 2.4.8

Проиллюстрируем это определение на примере.

Пример 2.4.9. Пусть даны следующие утверждения на русском языке:

S_1: Питер — вор.

S_2: Мэри любит шоколад.

S_3: Мэри любит вино.

S_4: Питер любит деньги.

S_5: Питер любит x, если x любит вино.

S_6: x может украсть y, если x — вор и x любит y.

Введя константы «Питер», «Мэри», «вино», «шоколад» и «деньги», переменные x, y и предикаты «вор» (одноместный), «любит» (двуместный) и «может_украсть» (двуместный), мы можем составить следующую программу:

C_1: вор(Питер) $\leftarrow$;

C_2: любит(Мэри, шоколад) $\leftarrow$;

C_3: любит(Мэри, вино) $\leftarrow$;

C_4: любит(Питер, деньги) $\leftarrow$;

C_5: любит(Питер, x) $\leftarrow$ любит(x, вино);

C_6: может_украсть(x, y) $\leftarrow$ вор(x), любит(x, y).

Хорновские дизъюнкты C_1, C_2, C_3 и C_4 являются фактами. Хорновские дизъюнкты C_5 и C_6 образуют процедурную часть программы.

Пусть мы хотим узнать, что именно Питер может украсть (если он вообще может что-то украсть). Мы должны записать следующую цель:

$$G: \leftarrow \text{может\_украсть (Питер, } y)$$

Ответ мы найдем в примере 2.10.12. □ 2.4.9

Позже мы подробно обсудим способы выведения заключений из множеств дизъюнктов, подобных множеству из предыдущего примера.

В параграфе 2.5 мы разовьем теорию интерпретаций и опишем, как с помощью интерпретаций можно приписывать истинностные значения предложениям логики предикатов, то есть формулам без свободных переменных.

§ 2.5. Интерпретация в логике предикатов

В рамках логики высказываний при нахождении истинностных значений составных высказываний мы использовали понятие *функции означивания*. Мы приписывали некоторые значения атомарным формулам данного высказывания A и по индукции определяли истинностное значение всего высказывания A.

В этом параграфе мы введем понятие интерпретации, на котором основаны методы нахождения истинностных значений предложений PrL.

Интерпретация: неформальное описание

Мы хотим проинтерпретировать предложения PrL, т. е. приписать им истинностные значения. Основными элементами этих предложений являются термы, состоящие из переменных и констант. Поэтому для интерпретации термов мы должны сначала выбрать множество объектов, на котором термы будут интерпретироваться. Затем мы сможем определить истинностное значение предикатов и предложений языка.

Итак, мы видим, что семантика PrL сложнее, чем семантика PL. Для лучшего понимания последующих определений, приведем несколько примеров.

Пример 2.5.1. Рассмотрим язык $\mathcal{L} = \{Q, \alpha\}$, где Q — предикат, α — константа, и предложение этого языка

$$S: (\exists x)(\forall y)Q(x, y).$$

Для каждой интерпретации этого языка нам нужно определить множество объектов, элементы которого должны соответствовать

символам $\mathcal{L}$, и приписать истинностные значения предложениям языка. Возьмем множество натуральных чисел $\mathbb{N}$ в качестве множества объектов. Тогда мы можем определить интерпретацию как структуру вида:

$$\mathcal{A} = (\mathbb{N},\ \leqslant,\ 1),$$

где

$\mathbb{N}$: множество натуральных чисел,
$\leqslant$: отношение «меньше или равно» на $\mathbb{N}$,
1: натуральное число 1.

Установим следующее соответствие между элементами $\mathcal{A}$ и символами языка $\mathcal{L}$:

символ Q соответствует отношению $\leqslant$ на множестве $\mathbb{N}$,
символ α соответствует натуральному числу 1.

Пусть переменные x и y формулы S принимают значения на множестве $\mathbb{N}$. Тогда формула S в интерпретации $\mathcal{A}$ имеет следующий смысл:

S: существует такое натуральное число x,
что для каждого натурального числа y
выполняется отношение $x \leqslant y$.

Таким образом, S утверждает, что в $\mathcal{A}$ существует минимальный элемент. Это, конечно, верно для $\mathcal{A}$, так как 1 является минимальным элементом $\mathcal{A}$.

Теперь определим другую интерпретацию того же самого языка $\mathcal{L}$:

$$\mathcal{A}' = (\mathbb{N},\ >,\ 1),$$

где предикат Q соответствует отношению «$>$» — «больше» на множестве натуральных чисел, а константа α соответствует числу 1.

В этой интерпретации формула S имеет следующий смысл:

S: существует такое натуральное число x, что
для каждого натурального числа y
выполняется отношение $x > y$.

В данном случае S утверждает, что в $\mathcal{A}'$ существует максимальный элемент, а это, очевидно, неверно, так как $\mathcal{A}'$ не имеет максимального элемента.

В итоге, мы видим, что можно определять различные интерпретации для одного и того же языка, приписывая при этом различные истинностные значения предложениям. □ 2.5.

Интерпретация и истинность: формальное описание

Теперь мы перейдем к формальному описанию интерпретации и определим понятие истинности предложения в контексте PrL [ChLe73, Dela87, Hami78, Meta85, RaSi70].

О п р е д е л е н и е 2.5.2. Пусть мы имеем язык PrL

$$\mathcal{L} = \{R_0, R_1, \ldots, f_0, f_1, \ldots, c_0, c_1, \ldots\},$$

где R_i — предикатные символы, f_i — функциональные символы и c_i — константы, $i = 0, 1, \ldots$ *Интерпретация* языка $\mathcal{L}$

$$\mathcal{A} = (A, \varepsilon(R_0), \varepsilon(R_1), \ldots, \varepsilon(f_0), \varepsilon(f_1), \ldots, \varepsilon(c_0), \varepsilon(c_1), \ldots)$$

включает в себя:

i) множество $A \neq \varnothing$ — область интерпретации;

ii) n-местное отношение $\varepsilon(R_i) \subseteq A^n$ для каждого n-местного предикатного символа R_i;

iii) функцию $\varepsilon(f_i)$: $A^m \mapsto A$ для каждого функционального символа f_i;

iv) элемент $\varepsilon(c_i) \in A$ для каждой константы c_i.

Элементы $\varepsilon(R_i)$, $\varepsilon(f_i)$ и $\varepsilon(c_i)$ называются *интерпретациями* символов R_i, f_i и c_i в $\mathcal{A}$. □ 2.5.2

П р и м е р 2.5.3. Рассмотрим язык

$$\mathcal{L} = \{=, \leqslant, +, *, 0, 1\}$$

из примера 2.2.2. Интерпретация этого языка имеет вид:

$$\mathcal{A} = (\mathbb{N}, =, \leqslant, +, *, 0, 1),$$

где $\mathbb{N}$ — множество натуральных чисел, «$=$» — отношение равенства на множестве $\mathbb{N}$, «$\leqslant$» — отношение «меньше или равно», «$+$», «$*$» — сложение и умножение на множестве $\mathbb{N}$.

Отметим здесь, что символ «$+$» языка $\mathcal{L}$ является трехместным предикатным символом, например, $+(2, 3, 5)$ означает $2 + 3 = 5$. В интерпретации $\mathcal{A}$ этот символ *интерпретируется* как $\varepsilon(+)$ — символ функции сложения натуральных чисел. Мы, как правило, будем обозначать выражением $\varepsilon(@)$ объект, являющийся интерпретацией символа «$@$» языка $\mathcal{L}$ в интерпретации $\mathcal{A}$.

Таким образом, мы имеем следующие интерпретации символов $\mathcal{L}$:

$$\varepsilon(=) \subseteq \mathbb{N} \times \mathbb{N};$$
$$\varepsilon(\leqslant) \subseteq \mathbb{N} \times \mathbb{N};$$
$$\varepsilon(+) \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N};$$
$$\varepsilon(*) \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{N};$$
$$\varepsilon(0) \in \mathbb{N};$$
$$\varepsilon(1) \in \mathbb{N}.$$

□ 2.5.3

Как мы видели, язык PrL и его интерпретация — разные понятия. Однако, в дальнейшем для простоты изложения мы часто будем использовать символы языка для обозначения интерпретаций этих символов. Например, символ $\leqslant$ будет использоваться и в качестве предикатного символа языка, и в качестве его интерпретации вместо $\varepsilon(\leqslant)$ — подмножества $\mathbb{N} \times \mathbb{N}$.

Как мы видели в примере 2.5.1, один и тот же язык может иметь несколько интерпретаций.

Пример 2.5.4. Рассмотрим интерпретацию языка $\mathcal{L}$, отличную от интерпретации, изучаемой в примере 2.5.3:

$$\mathcal{A}' = (\{2n \mid n \in \mathbb{N}_0\}, =', *', +', 0') \qquad \text{(четные натуральные числа)},$$

где

$\varepsilon(=)$ есть $='$ — равенство на множестве $\{2n \mid n \in \mathbb{N}_0\}$,

$\varepsilon(*)$ есть $*'$ — умножение на множестве $\{2n \mid n \in \mathbb{N}_0\}$,

$\varepsilon(+)$ есть $+'$ — сложение на множестве $\{2n \mid n \in \mathbb{N}_0\}$,

$\varepsilon(0)$ есть $0'$ — нулевой элемент операции $+'$.

Как отмечалось ранее, предложение языка может быть «истинным» в одной интерпретации и «ложно» в другой. Например, предложение

$$(\exists y)(\forall x)(x * y = x),$$

утверждающее о существовании единичного элемента операции $*'$, истинно в $\mathcal{A}$ и ложно в $\mathcal{A}'$. □ 2.5.4

Теперь мы дадим индуктивное определение истинности предложения в интерпретации $\mathcal{A}$.

Определение 2.5.5 (индуктивное определение истинности предложения в данной интерпретации). Пусть

$$\mathcal{L} = \{R_0, R_1, \ldots, f_0, f_1, \ldots, c_0, c_1, \ldots\}$$

— некоторый язык, и $\mathcal{A}$ — некоторая его интерпретация.

а) Интерпретацией основного терма t называется такой элемент $\varepsilon(t) \in A$, что

если t есть константа c, то $\varepsilon(t) = \varepsilon(c)$;

если t имеет вид $f(t_1, \ldots, t_n)$, где $t_1, \ldots, t_n$ — основные термы, тогда $\varepsilon(t) = \varepsilon(f)(\varepsilon(t_1), \ldots, \varepsilon(t_n))$.

б) Атомарное предложение $R_i(t_1, \ldots, t_n)$ *истинно* в интерпретации $\mathcal{A}$ тогда и только тогда, когда

$$(\varepsilon(t_1), \ldots, \varepsilon(t_n)) \in \varepsilon(R_i).$$

В этом случае мы записываем формально

$$\mathcal{A} \models R_i(t_1, \ldots, t_n)$$

Пусть φ, φ_1 и φ_2 — некоторые предложения языка $\mathcal{L}$. Тогда

в) $\mathcal{A} \models \neg\varphi \Leftrightarrow \mathcal{A} \not\models \varphi$;

г) $\mathcal{A} \models \varphi_1 \vee \varphi_2 \Leftrightarrow \mathcal{A} \models \varphi_1$ или $\mathcal{A} \models \varphi_2$;

д) $\mathcal{A} \models \varphi_1 \wedge \varphi_2 \Leftrightarrow \mathcal{A} \models \varphi_1$ и $\mathcal{A} \models \varphi_2$;

е) $\mathcal{A} \models \varphi_1 \to \varphi_2 \Leftrightarrow \mathcal{A} \not\models \varphi_1$ или $\mathcal{A} \models \varphi_2$;

ж) $\mathcal{A} \models \varphi_1 \leftrightarrow \varphi_2 \Leftrightarrow (\mathcal{A} \models \varphi_1$ и $\mathcal{A} \models \varphi_2)$ или $(\mathcal{A} \not\models \varphi_1$ и $\mathcal{A} \not\models \varphi_2)$;

з) $\mathcal{A} \models (\forall u)\varphi(u) \Leftrightarrow$ существует элемент $a \in A$ такой, что $\mathcal{A}' \models \varphi(c)$, где $c \notin \mathcal{L}$, и $\mathcal{A}'$ получена из $\mathcal{A}$ путем добавления элемента $\varepsilon(c) = a$;

и) $\mathcal{A} \models (\forall u)\varphi(u) \Leftrightarrow$ для любого элемента $a \in A$ выполняется $\mathcal{A}' \models \varphi(c)$, где $c \notin \mathcal{L}$, и $\mathcal{A}'$ получена из $\mathcal{A}$ путем добавления $\square$ 2.5.5

Замечание 2.5.6. 1. Согласно определению 2.5.5 и замечанию 2.3.7 мы имеем

$$\mathcal{A} \models c_1 = c_2 \Leftrightarrow \varepsilon(c_1) \text{ совпадает с } \varepsilon(c_2).$$

2. Если атомарное предложение $R_i(c_{i_1}, \ldots, c_{i_n})$ является истинным в $\mathcal{A}$, то R_i принимает значение t, если предложение не является истинным, то R_i принимает значение f (упражнение 12). $\square$ 2.5.6

В определении истинности формулы в данной интерпретации индукция проводится по сложности предложения. Например, согласно пункту в), дизъюнкция $(\varphi_1 \vee \varphi_2)$ истинна в $\mathcal{A}$ тогда и только тогда, когда по крайней мере одно из предложений φ_1, φ_2 истинно в $\mathcal{A}$.

Все аксиомы из определения 2.3.3 и замечания 2.3.7 являются формулами, истинными во всякой интерпретации $\mathcal{A}$. Более того, по правилам обобщения и Modus Ponens из истинных формул получаются также истинные формулы в любой интерпретации PrL (что означает корректность аксиоматической системы PrL).

Пример 2.5.7. Пусть $\mathcal{L} = \{\mathbb{Q}, g\}$ — язык арифметики, и

$$\mathcal{A} = (\mathbb{Q}_+, =, g), \qquad \mathcal{A}' = (\mathbb{R}_+, =, g)$$

— две интерпретации языка $\mathcal{L}$, где

$\mathbb{Q}_+$: множество положительных рациональных чисел,

$\mathbb{R}_+$: множество положительных вещественных чисел,

g: $g(x) = x^2$,

$\varepsilon'(Q)$: $=$ — равенство на множестве $\mathbb{R}_+$,

$\varepsilon(Q)$: $=$ — равенство на множестве $\mathbb{Q}_+$,

$\varepsilon'(g)$: функция g, определенная на множестве $\mathbb{R}_+$,

$\varepsilon(g)$: функция g, определенная на множестве $\mathbb{Q}_+$.

Тогда предложение S: $(\forall x)(\exists y)Q(x, g(y))$, т. е. $(\forall x)(\exists y)(x = y^2)$, истинно в интерпретации $\mathcal{A}'$, но ложно в интерпретации $\mathcal{A}$, так как уравнение $x = y^2$ для заданного положительного x всегда имеет решение на множестве $\mathbb{R}_+$, но не всегда на множестве $\mathbb{Q}_+$. $\square$ 2.5.7

Пример 2.5.8. Для языка $\mathcal{L}=\{Q, f, a, b\}$ мы можем определить следующую интерпретацию:

$$\mathcal{A}=(A,\ \text{ребенок},\ \text{мать},\ \text{Джон},\ \text{Мэри},\ \text{Наполеон}),$$

где

A: множество всей людей,

$\varepsilon(Q)$: отношение «ребенок», т. е. $\varepsilon(Q)(x_1, x_2)=\text{ребенок}(x_1, x_2)=$ $=«x_1$ является ребенком $x_2»$,

$\varepsilon(f)$: функция «мать», т. е. $\varepsilon(f)(x)=\text{мать}(x)=«\text{мать } x»$,

$\varepsilon(a)$: Мэри,

$\varepsilon(b)$: Джон,

$\varepsilon(c)$: Наполеон.

Отметим, что в интерпретации $\mathcal{A}$ один символ, а именно c, не является элементом языка, тем не менее, он тоже проинтерпретирован. Ниже (см. теорему 2.5.14) мы увидим, что такая интерпретация символа, не принадлежащего языку, не вызывает трудностей.

Рассмотрим предложение S: $Q(b, f(b))\vee(\exists x)(Q(a, x))$. В интерпретации $\mathcal{A}$ оно имеет вид:

S: «Джон является ребенком матери Джона

или

существует некто x, чьим ребенком является Мэри»

Очевидно, Джон является ребенком матери Джона, поэтому предложение S истинно в интерпретации $\mathcal{A}$ (пункт в) определения 2.5.5). □ 2.5.8

Определение 2.5.9. Интерпретация $\mathcal{A}$ языка $\mathcal{L}$ называется *моделью* предложения σ тогда и только тогда, когда $\mathcal{A}\models\sigma$. □ 2.5.9

Определение 2.5.10. Для интерпретации $\mathcal{A}$ языка $\mathcal{L}$ множество

$$\theta(\mathcal{A})=\{\sigma \mid \mathcal{A}\models\sigma\}$$

называется *теорией интерпретации.* □ 2.5.10

Другими словами, теория интерпретации есть множество всех предложений, истинных в этой интерпретации.

В примере 2.5.8 мы видели, что возможно проинтерпретировать символы, не являющиеся элементами языка. Дадим определение, которое будет полезным в таких случаях.

Определение 2.5.11. Пусть мы имеем такую интерпретацию $\mathcal{A}$ языка $\mathcal{L}$, что элементы $a_1, a_2, \ldots$ из области интерпретации не являются интерпретациями констант языка $\mathcal{L}$.

1. Сформируем *новый язык*:

$$\mathcal{L}^*=\mathcal{L}\cup\{c_1, c_2, \ldots\},$$

где символы $c_1, c_2, \ldots$ являются *новыми* символами констант, не принадлежащими $\mathcal{L}$. Язык $\mathcal{L}^*$ называется *элементарным расширением* языка $\mathcal{L}$.

2. Для произвольной интерпретации

$$\mathcal{A} = (A, R_1, R_2, \ldots, f_1, f_2, \ldots, d_1, d_2, \ldots)$$

мы построим *новую интерпретацию*:

$$\mathcal{A}^* = (A, R_1, R_2, \ldots, f_1, f_2, \ldots, d_1, d_2, \ldots, a_1, a_2, \ldots)$$

языка $\mathcal{L}^* = \mathcal{L} \cup \{c_1, c_2, \ldots\}$, где $c_1, c_2, \ldots \notin \mathcal{L}$, фиксируя при этом следующую интерпретацию символов $c_1, c_2, \ldots$: $\varepsilon(c_1) = a_1$, $\varepsilon(c_2) = a_2, \ldots$ Интерпретация $\mathcal{A}^*$ называется *элементарным расширением* интерпретации $\mathcal{A}$. □ 2.5.11

Пример 2.5.12. Предположим, что

$$\mathcal{L} = \{=, \leqslant, +, *, 0, 1\}, \quad \mathcal{A} = (\mathbb{N}, =, \leqslant, +, *, 0, 1).$$

Тогда мы можем построить язык

$$\mathcal{L}^* = \{=, \leqslant, +, *, 0, 1, c_1, c_2, \ldots\}$$

и его интерпретацию

$$\mathcal{A}^* = (\mathbb{N}, =, \leqslant, +, *, 0, 1, 2, 3, \ldots). \qquad \square\ 2.5.12$$

Замечание 2.5.13. Кроме констант в язык $\mathcal{L}$ могут быть добавлены также новые символы функций и предикатов. Расширение языка $\mathcal{L}$ новыми символами функций и предикатов $\mathcal{L}^*$ называется неэлементарным расширением $\mathcal{L}$. Соответствующее расширение $\mathcal{A}^*$ интерпретации $\mathcal{A}$ называется неэлементарным расширением $\mathcal{A}$. □ 2.5.13

Сформулируем теорему об истинности предложения в контексте PrL

Теорема 2.5.14. *Пусть $\mathcal{A}$ есть интерпретация языка $\mathcal{L}$. Предположим, что $\mathcal{A}^*$ и $\mathcal{L}^*$ — такие элементарные расширения $\mathcal{A}$ и $\mathcal{L}$, что каждый элемент области интерпретации $\mathcal{A}^*$ является интерпретацией некоторого символа языка $\mathcal{L}^*$. Тогда предложение σ языка $\mathcal{L}$ истинно в интерпретации $\mathcal{A}$ тогда и только тогда, когда оно истинно в интерпретации $\mathcal{A}^*$. Формально,*

$$\mathcal{A} \models \sigma \Leftrightarrow \mathcal{A}^* \models \sigma$$

Доказательство. Индукция по сложности предложения σ. Вначале следует отметить, что при элементарном расширении значения всех основных термов языка $\mathcal{L}$ остаются в интерпретации

$\mathcal{A}^*$ такими же, как и в исходной интерпретации $\mathcal{A}$. В этом легко убедиться, воспользовавшись математической индукцией по структуре основного терма и принимая во внимание то обстоятельство, что все элементы языка $\mathcal{L}$ имеют в интерпретации $\mathcal{A}^*$ то же самое значение, что и в интерпретации $\mathcal{A}$.

Если σ имеет вид $P(t_1, \ldots, t_k)$, где $t_1, \ldots, t_k$ — основные термы, то

$$\begin{aligned}\mathcal{A}^* \models P(t_1, \ldots, t_k) &\Leftrightarrow (\varepsilon(t_1), \ldots, \varepsilon(t_k)) \in \varepsilon(P) \\ &\quad \text{и} \quad \varepsilon(t_1), \ldots, \varepsilon(t_k) \in \mathcal{A}^*, \quad \varepsilon(P) \subseteq A^k \\ &\Leftrightarrow (\varepsilon(t_1), \ldots, \varepsilon(t_k)) \in \varepsilon(P) \\ &\quad \text{и} \quad \varepsilon(t_1), \ldots, \varepsilon(t_k) \in \mathcal{A}, \quad \varepsilon(P) \subseteq A^k, \\ &\quad \text{так как} \quad \sigma \text{ — предложение языка } \mathcal{L} \\ &\Leftrightarrow \mathcal{A} \models P(t_1, \ldots, t_k).\end{aligned}$$

Случаи σ: $\neg\varphi$, σ: $\varphi \vee \varphi'$, σ: $\varphi \wedge \varphi'$ и σ: $\varphi \to \varphi'$ рассматриваются аналогично.

Пусть предложение σ имеет вид $(\exists x)\varphi(x)$, где x — единственная свободная переменная формулы φ. Тогда

$$\mathcal{A}^* \models (\exists)\varphi(x) \quad \Leftrightarrow$$

для некоторого элемента $\alpha \in \mathcal{A}^*$ и константы $c \notin \mathcal{L}^*$, $\varepsilon(c) = \alpha$

$$\mathcal{A}^* \models \varphi(c) \quad \Leftrightarrow$$

согласно предположению индукции, равенству областей интерпретаций $\mathcal{A}^*$ и $\mathcal{A}$, а также тому факту, что φ — предложение языка $\mathcal{L}$

$$\mathcal{A} \models \varphi(c).$$

Случай σ: $(\forall x)\varphi(x)$ рассматривается аналогично. □ 2.5.14

Согласно теореме 2.5.14, истинность предложения в интерпретации $\mathcal{A}^*$ не зависит от выбора новых символов констант и их интерпретации.

Определение 2.5.15. *Предложение* σ языка $\mathcal{L}$ *выполнимо* или *непротиворечиво* тогда и только тогда, когда существует интерпретация $\mathcal{A}$ языка $\mathcal{L}$ в которой это предложение истинно, т. е. $\mathcal{A} \models \sigma$. □ 2.5.15

Определение 2.5.16. *Множество предложений* S *выполнимо* или *непротиворечиво*, если существует интерпретация в которой все предложения S истинны. В противном случае, множество S *невыполнимо* или *противоречиво*. □ 2.5.16

Пример 2.5.17. Пусть

$$\mathcal{A} = (\mathbb{N}, =, \leqslant, +, *, 0, 1).$$

Тогда выполняется $\mathcal{A} \models \sigma$, где σ — одно из предложений (1)–(2) примера 2.2.8.

Докажем, что $\mathcal{A} \models (\forall x)(\forall y)(x = y \to y = x)$.

Предположим, $\mathcal{A} \models (\forall x)(\forall y)(x = y \to y = x)$ неверно. Тогда по определению 2.5.5 з) и б) существуют c_1, c_2 такие, что

$$\mathcal{A} \models \neg[(c_1 = c_2) \to (c_2 = c_1)].$$

Следовательно,

$$\mathcal{A} \models \neg[\neg(c_1 = c_2) \vee (c_2 = c_1)],$$

или

$$\mathcal{A} \models [(c_1 = c_2) \wedge \neg(c_2 = c_1)],$$

и по определению 2.5.5 г)

$$\mathcal{A} \models (c_1 = c_2) \tag{1}$$

и

$$\mathcal{A} \models \neg(c_2 = c_1). \tag{2}$$

По седьмой аксиоме замечания 2.3.7, т. е. аксиоме подстановки равных термов, мы, используя (1), подставляем c_2 вместо c_1 в (2). Тогда

$$\mathcal{A} \models \neg(c_2 = c_2)$$

или, по определению 2.5.5 ж),

$$\mathcal{A} \models (\exists x)\neg(x = x).$$

Используя двойственность $\forall$ и $\exists$, получаем

$$\mathcal{A} \models \neg(\forall x)(x = x),$$

что противоречит рефлексивности отношения равенства. □ 2.5.17

Пример 2.5.18. Стандартная интерпретация языка $\mathcal{L} = \{=, \leqslant, +, *, 0, 1\}$ имеет вид:

$$\mathcal{A} = (\mathbb{N}, =, \leqslant, +, *, 0, 1).$$

Рассмотрим другую интерпретацию языка $\mathcal{L}$

$$\mathcal{B} = (\mathbb{Q}, =, \leqslant, +, *, 0, 1).$$

где $\mathbb{Q}$ — множество рациональных чисел.

Пусть предложение σ имеет следующий вид (формальное определение плотного порядка):

$$(\forall x)(\forall y)[\neg(x = y) \to (\exists z)[\neg(z = x) \wedge \neg(z = y) \wedge (x \leqslant z) \wedge (z \leqslant y)]].$$

Тогда $\mathcal{A} \not\models \sigma$, но $\mathcal{B} \models \sigma$. □ 2.5.18

Точно также, как в PL, в каждом языке PrL существуют предложения, истинные во всех интерпретациях этого языка.

Определение 2.5.19 (Общезначимая формула). Если формула σ языка $\mathcal{L}$ истинна во всех интерпретациях $\mathcal{L}$, то она называется *общезначимой* или *логически истинной*. □ 2.5.19

Пример 2.5.20. Пусть φ — произвольное предложение языка $\mathcal{L}$, и ψ — произвольная формула $\mathcal{L}$.

Тогда предложение

$$S\colon ((\forall v)(\varphi \to \psi) \to (\varphi \to (\forall v)\psi))$$

истинно во всех интерпретациях $\mathcal{L}$.

Доказательство. Допустим, что предложение S не истинно во всех интерпретациях языка $\mathcal{L}$. Тогда существует интерпретация $\mathcal{A}$ языка $\mathcal{L}$ такая, что

$$\mathcal{A} \not\models ((\forall v)(\varphi \to \psi) \to (\varphi \to (\forall v)\psi)).$$

Отсюда, по определению 2.5.5 мы имеем

$$\text{неверно } [\mathcal{A} \not\models (\forall v)(\varphi \to \psi) \quad \text{или} \quad \mathcal{A} \models \varphi \to (\forall v)\psi],$$

по правилу де Моргана

$$\text{неверно } [\mathcal{A} \not\models (\forall v)(\varphi \to \psi)] \quad \text{и} \quad \text{неверно } [\mathcal{A} \models \varphi \to (\forall v)\psi].$$

По закону двойного отрицания получаем

$$\mathcal{A} \models (\forall v)(\varphi \to \psi) \qquad (1) \quad \text{и} \quad \mathcal{A} \not\models \varphi \to (\forall v)\psi \qquad (2).$$

Из (2) следует $\mathcal{A} \not\models \neg\varphi \vee (\forall v)\psi$, и по определению 2.5.5 б) мы имеем:

$$\mathcal{A} \models \varphi \quad \text{и} \quad \mathcal{A} \not\models (\forall v)\psi. \tag{3}$$

Из (3) и определения 2.5.5 з) следует, что существует константа $c \in \mathcal{L}$ такая, что

$$\mathcal{A} \models \neg\psi(v/c),$$

и из $\mathcal{A} \models \varphi$ и определения 2.5.5 г) получаем

$$\mathcal{A} \models \varphi \wedge \neg\psi(v/c),$$

что эквивалентно

$$\mathcal{A} \models \neg(\neg\varphi \vee \psi(v/c)),$$

что, в свою очередь, эквивалентно

$$\mathcal{A} \models \neg(\varphi \to \psi(v/c)). \tag{4}$$

Из (1) и определения 2.5.5 з) для всякой константы $c \in \mathcal{L}$ выполняется

$$\mathcal{A} \models (\varphi \to \psi)(v/c). \tag{5}$$

Однако, φ является предложением и не имеет свободных переменных. Следовательно, в формуле φ невозможно сделать ни одну подстановку. Отсюда, (5) имеет вид $\mathcal{A} \models \varphi \to \psi(v/c)$ для всех $c \in \mathcal{L}$, что вступает в противоречие с (4). Следовательно, предложение S истинно во всех интерпретациях языка $\mathcal{L}$. □ 2.5.20

Определение 2.5.21. Пусть $\mathcal{L}$ — язык PrL, и S — множество предложений PrL. Предложение σ называется *следствием* S (краткая запись $S \models \sigma$) тогда и только тогда, когда во всякой интерпретации $\mathcal{A}$ языка $\mathcal{L}$, в которой истинны все предложения из S, истинно также и предложение σ. Это можно записать следующим образом:

$$\sigma \in Con(S) \;\Leftrightarrow\; (\forall \mathcal{A})[\mathcal{A} \models S \Rightarrow \mathcal{A} \models \sigma] \qquad \square\ 2.5.21$$

§ 2.6. Нормальные формы в логике предикатов

Изучая логику высказываний, мы рассматривали две эквивалентные нормальные формы высказываний: КНФ (конъюктивная нормальная форма) и ДНФ (дизъюнктивная нормальная форма). Аналогичные формы можно найти также и в логике предикатов. В контексте PrL, однако, существуют две дополнительные нормальные формы: предваренная нормальная форма (ПНФ) и сколемовская нормальная форма (СНФ), которые различаются типом кванторов, входящих в предложение [Chur56, ChLe73, Hami78, Klee52, Thay88]. Преобразуя каждое из двух заданных предложений в одну из этих форм, мы можем легко их сравнить и определить, эквивалентны ли они, является ли одно из них отрицанием другого, или просто, обладают ли они какими-либо особенностями. Сколемовская нормальная форма играет важную роль в логическом программировании.

Мы рассмотрим эти формы в следующих пунктах.

Предваренная нормальная форма

Определение 2.6.1. Формула φ находится в *предваренной нормальной форме* (сокращенно ПНФ), если она имеет вид:

$$\varphi\colon (Q_1 x_1)(Q_2 x_2)\ldots(Q_n x_n)\sigma,$$

где Q_i, $i = 1, \ldots, n$ обозначает один из кванторов $\forall, \exists$, и σ — формула без кванторов. Выражение $(Q_1 x_1)\ldots(Q_n x_n)$ называется *префиксом* формулы φ, а σ называется *матрицей* φ. □ 2.6.1

Пример 2.6.2. Следующие предложения находятся в ПНФ:

а) $(\forall x)(\forall y)[P(x, y) \to Q(x)]$;

б) $(\forall x)(\exists y)[Q(x, y) \vee P(x, y)]$ □ 2.6.2

Чтобы привести формулу или предложение к ПНФ, кроме эквивалентностей логики высказываний мы также будем использовать следующие тождества:

1) $(Qx)P(x) \vee G \leftrightarrow (Qx)[P(x) \vee G]$, где x не входит свободно в G;

2) $(Qx)P(x) \wedge G \leftrightarrow (Qx)[P(x) \wedge G]$, где x не входит свободно в G;

3) $\neg(\forall x)P(x) \leftrightarrow (\exists x)(\neg P(x))$;

4) $\neg(\exists x)P(x) \leftrightarrow (\forall x)(\neg P(x))$;

5) $(\forall x)\,P(x) \wedge (\forall x)\,G(x) \leftrightarrow (\forall x)\,(P(x) \wedge G(x))$;

6) $(\exists x)\,P(x) \vee (\exists x)\,G(x) \leftrightarrow (\exists x)\,(P(x) \vee G(x))$;

где (Qx) обозначает либо $(\forall x)$, либо $(\exists x)$.

Представленные выше формулы выводимы в аксиоматической системе PrL (теорема 2.3.6). Формулы 1) и 2) утверждают, что зону действия квантора можно распространить на конъюнкцию или дизъюнкцию, если переменная под квантором не входит свободно в соответствующую подформулу. Формулы 3) и 4) очевидны ввиду двойственности кванторов $\forall$ и $\exists$. Формулы 5) и 6) утверждают о дистрибутивности квантора всеобщности относительно конъюнкции и квантора существования относительно дизъюнкции. Обратное, однако, неверно:

7) $(\exists x)P(x) \wedge (\exists x)G(x) \not\to (\exists x)\,(P(x) \wedge G(x))$;

8) $(\forall x)P(x) \vee (\forall x)G(x) \not\to (\forall x)\,(P(x) \vee G(x))$.

Чтобы получить эквивалентности в случаях, подобных 7) и 8), мы сначала переименуем все вхождения переменной x в формулах $(\forall x)G(x)$ и $(\exists x)G(x)$. Это делать разрешено, поскольку x — связанная переменная. Затем по теореме 2.3.6, используя 5) и 6), получаем (как?) следующие эквивалентности:

9) $(Q_1 x)P(x) \wedge (Q_2 x)G(x) \leftrightarrow (Q_1 x)\,(Q_2 z)[P(x) \wedge G(z)]$,

10) $(Q_1 x)P(x) \vee (Q_2 x)G(x) \leftrightarrow (Q_1 x)\,(Q_2 z)[P(x) \vee G(z)]$,

где $Q_1, Q_2 \in \{\forall, \exists\}$, и переменная z не входит в $G(x)$ вообще и не входит свободно в $P(x)$.

Теперь опишем формальную процедуру для приведения предложения к ПНФ.

Построение 2.6.3.

Шаг 1. Избавляемся от символов $\leftrightarrow$ **и** $\to$ **с помощью формул:**

1а) $(A \leftrightarrow B) \leftrightarrow ((A \to B) \wedge (B \to A))$;

1б) $(A \to B) \leftrightarrow (\neg A \vee B)$.

Шаг 2. Проносим отрицания вглубь формулы до атомов с помощью формул:

2а) $\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$;

2б) $\neg(A \wedge B) \leftrightarrow \neg A \vee \neg B$;

2в) $\neg(\neg A) \leftrightarrow A$;

2г) $\neg(\forall x)A \leftrightarrow (\exists x)\neg A$;

2д) $\neg(\exists x)A \leftrightarrow (\forall x)\neg A$.

Шаг 3. Выносим кванторы наружу с помощью формул:

3а) $(\forall x)A(x) \wedge (\forall x)B(x) \leftrightarrow (\forall x)\,(A(x) \wedge B(x))$;

$(\exists x)A(x) \vee (\exists x)B(x) \leftrightarrow (\exists x)\,(A(x) \vee B(x))$;

3б) $(Qx)A(x) \wedge B \leftrightarrow (Qx)[A(x) \wedge B]$;

$(Qx)A(x) \vee B \leftrightarrow (Qx)[A(x) \vee B]$;

в формулах (3б) B не содержит свободных вхождений x;

3в) $(Q_1x)A(x) \vee (Q_2x)B(x) \leftrightarrow (Q_1x)\,(Q_2z)[A(x) \vee B(z)]$;

$(Q_1x)A(x) \wedge (Q_2x)B(x) \leftrightarrow (Q_1x)\,(Q_2z)[A(x) \wedge B(z)]$;

в формулах 3в) B не содержит свободных вхождений x, переменная z не входит в B и не входит свободно в A, и $B(z)$ есть результат замены свободных вхождений x на z. □ 2.6.3

Пример 2.6.4.

$$\begin{aligned}\varphi\colon\ (\forall x)\,P(x) \to \mathbf{(\exists x)}\,\boldsymbol{R}(x) &\leftrightarrow \neg(\forall x)P(x) \vee (\exists x)R(x) && (1б)\\ &\leftrightarrow (\exists x)\neg P(x) \vee (\exists x)R(x) && (2г)\\ &\leftrightarrow (\exists x)[\neg P(x) \vee R(x)] && (3а)\end{aligned}$$

□ 2.6.4

Пример 2.6.5.

$$\begin{aligned}&\varphi\colon (\forall x)\,(\forall y)[(\exists z)\,(P(x,z) \wedge P(y,z)) \to (\exists u)R(x,y,u)]\\ &\leftrightarrow(\forall x)\,(\forall y)[\neg(\exists z)\,(P(x,z) \wedge P(y,z)) \vee (\exists u)R(x,y,u)] && (1б)\\ &\leftrightarrow(\forall x)\,(\forall y)[(\forall z)\,(\neg P(x,z) \vee \neg P(y,z)) \vee (\exists u)R(x,y,u)] && (2б)\\ &\leftrightarrow(\forall x)\,(\forall y)\,(\forall z)[\neg P(x,z) \vee \neg P(y,z) \vee (\exists u)R(x,y,u)] && (3б)\\ &\leftrightarrow(\forall x)\,(\forall y)\,(\forall z)\,(\exists u)[\neg P(x,z) \vee \neg P(y,z) \vee R(x,y,u)] && (3б)\end{aligned}$$

□ 2.6.5

Теперь мы можем перейти к преобразованию предложения φ в *универсальное* предложение φ^*, которое содержит только кванторы всеобщности, находится в предваренной форме и выполнимо тогда и только тогда, когда выполнимо φ.

Сколемовская нормальная форма

Теорема 2.6.6 (Лёвенгейм, Сколем). *Для каждого предложения φ логики предикатов мы можем построить универсальное предложение φ^* такое, что*

$$\varphi \text{ выполнимо} \Leftrightarrow \varphi^* \text{ выполнимо.} \qquad \square\ 2.6.6$$

Опишем, как построить φ^* для произвольного предложения φ.

Построение 2.6.7. Пусть φ — предложение языка PrL.

Шаг 1. Строим ПНФ предложения φ.

Шаг 2. Последовательно (слева направо) вычеркиваем каждый квантор существования $(\exists y)$, заменяя все вхождения переменной y на новый, еще не использованный функциональный символ f, в качестве аргументов f берем все переменные, связанные предшествующими $(\exists y)$ кванторами всеобщности. Функциональный символ f называется *сколемовской функцией*. Предложение φ^*, полученное после выполнения шагов 1 и 2 называется *сколемовской нормальной формой*, коротко СНФ. $\square$ 2.6.7

Пример 2.6.8. Рассмотрим предложение

$$\varphi\colon\ (\forall x)(\exists y)(\forall z)(\exists v)P(x, y, z, v),$$

которое находится в ПНФ.

1. Вычеркиваем $(\exists y)$ и заменяем y на сколемовскую функцию $f(x)$. Таким образом, получаем предложение

$$\varphi_1\colon\ (\forall x)(\forall z)(\exists v)P(x, f(x), z, v).$$

2. Вычеркиваем $(\exists v)$ в φ_1 и заменяем v на сколемовскую функцию $g(x, z)$, так как кванторы $(\forall x)$, $(\forall z)$ предшествуют квантору $(\exists v)$. В итоге, получаем

$$\varphi^*\colon\ (\forall x)(\forall z)P(x, f(x), z, g(x, z)). \qquad \square\ 2.6.8$$

Говоря неформально, в предыдущем примере $f(x)$ указывает на существование переменной y, которая зависит от x (по определению 2.5.5 з) и ж) для каждого x существует константа, которую можно подставить вместо y). Значит, переменная y должна быть связана квантором существования. Следовательно, из φ_1 и φ^* следует φ, если принять во внимание $f(x)$ и $g(x, z)$.

Пример 2.6.9. СНФ предложения

$$\varphi\colon\ (\exists y)(\forall x)(\forall z)\psi(x, y, z)$$

имеет вид

$$\varphi^*: \ (\forall x)(\forall z)\psi(x, c, z),$$

где c — константа, поскольку соответствующая сколемовская функция f не имеет аргументов ($(\exists y)$ — первый квантор предложения φ, следовательно, y не зависит ни от одной переменной, и функция f превращается в константу c). □ 2.6.9

Приведение предложения к СНФ имеет большое значение в логическом программировании, это станет еще более очевидным, когда мы представим метод резолюций для вывода формул в PrL.

В определении 1.9.4 для логики высказываний мы видели, что предложение вида

$$(A_{i_1} \vee \ldots \vee A_{i_n}) \wedge \ldots \wedge (A_{k_1} \vee \ldots \vee A_{k_n}),$$

то есть, конъюнкция дизъюнкций, может быть представлена в теоретико-множественном виде:

$$\{\{A_{i_1}, \ldots, A_{i_n}\}, \ldots, \{A_{k_1}, \ldots, A_{k_n}\}.$$

Вспоминая определение 2.4.1 и учитывая кванторы, мы вводим определение теоретико-множественного представления предложений PrL.

Определение 2.6.10. Пусть предложение φ PrL находится в СНФ:

$$\varphi: \ (\forall x_1) \ldots (\forall x_l) A(x_1, \ldots, x_l),$$

где $A(x_1, \ldots, x_l)$ есть конъюнкция

$$C_1(x_1, \ldots, x_l) \wedge \ldots \wedge C_k(x_1, \ldots, x_l),$$

и каждое C_i, $1 \leqslant i \leqslant k$ есть дизъюнкция атомов PrL или их отрицаний $P_{i_1}, \ldots, P_{i_n}$. Тогда *теоретико-множественным* представлением φ называется множество

$$S = \{\{P_{i_1}, \ldots, P_{i_n}\}, \ldots, \{P_{k_1}, \ldots, P_{k_n}\}\} \text{ или } S = \{C_1, \ldots, C_k\}.$$

Каждое C_i является *дизъюнктом*, а S — множеством дизъюнктов. □ 2.6.10

Пример 2.6.11. Предложение

$$(\forall x)(\forall z)[P_1(x, y) \wedge (P_2(x) \vee P_3(z)) \wedge P_4(z, x)]$$

имеет следующее теоретико-множественное представление:

$$\{\{P_1(x, y)\}, \{(P_2(x), P_3(z)\}, \{P_4(z, x)\}\} \qquad \square \ 2.6.11$$

В следующих разделах мы рассмотрим *эрбрановские интерпретации*, а также методы доказательства выполнимости предложения или множества дизъюнктов PrL.

§ 2.7. Эрбрановские интерпретации

В этом разделе мы опишем особый вид интерпретаций — эрбрановские интерпретации [ChLe73, Dela87, Klee52, Thay88], которые играют важную роль в теоретическом основании логического программирования.

Эрбрановские интерпретации были темой докторской диссертации Эрбрана в 1930 г.. Пожалуй, не будет преувеличением сказать, что без вклада Эрбрана логическое программирование было бы и до сих пор несбыточной мечтой. Основная проблема автоматического доказательства теорем состоит в построении универсальной процедуры, с помощью которой мы могли бы проверить, общезначимо предложение логики предикатов или нет [ChLe73]. В 1936 г. Тьюринг и Чёрч независимо друг от друга доказали, что такой универсальной процедуры не существует. Однако, Эрбран к тому времени уже решил косвенно эту проблему, представив алгоритм для построения интерпретации, опровергающей данную формулу φ. Если φ общезначима, опровергающей ее интерпретации не существует, и алгоритм останавливается за конечное число шагов.

Первые попытки применить идеи Эрбрана в логическом программировании предприняли в 1960 г. Гилмор, а также Дэвис и Путнэм; эти попытки, однако, нельзя назвать удачными. Успех пришел в 1965 г., когда Робинсон, применяя метод Эрбрана, ввел и использовал понятие резолюции.

Описание эрбрановского универсума

Для данного предложения PrL φ мы хотим определить, выполнимо φ или нет. Мы будем решать вопрос о выполнимости предложения, должным образом исследуя соответствующее φ множество дизъюнктов S.

Однако, проверка выполнимости или невыполнимости всех основных термов, входящих в дизъюнкт, практически невозможна. Поэтому, мы построим множество интерпретации, на котором будут принимать значения термы предложения φ. Это множество называется эрбрановским универсумом. *Эрбрановский универсум* строится индуктивно следующим образом.

Построение 2.7.1. (Построение эрбрановского универсума.)

Пусть S — множество дизъюнктов, соответствующее предложению φ.

Шаг 1.

$H_0 = \{c \mid$ константа c встречается в $S\}$;

$\{c_0\}$, если S не содержит ни одной **константы**,

где c_0 — новая константа («новая» означает, что она не встречается в S), которую мы выбираем произвольным образом.

Шаг $i+1$.

$$H_{i+1} = H_i \cup \{f(a_1, \ldots, a_n) \mid a_j,\ 1 \leqslant j \leqslant n, \text{ — термы из } H_i,\ f \text{ — функция, входящая в } S\}$$

И наконец, полагаем

$$H = \cup_{i \in N} H_i.$$

Множество H всех термов, построенных из констант H_0 и функциональных символов, встречающихся в S, называется *эрбрановским универсумом* для S. Множества H_i, $i = 0, 1, 2, \ldots$ называются *эрбрановскими множествами* S. □ 2.7.1

Пример 2.7.2. Пусть задано множество дизъюнктов

$$S\colon \{\{P(a)\}, \{\neg P(a), P(f(x))\}\},$$

где a — константа. Тогда

$$H_0 = \{a\},$$
$$H_1 = \{a, f(a)\},$$
$$H_2 = \{a, f(a), f(f(a))\},$$
$$\ldots$$

и наконец,

$$H = \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}.$$

□ 2.7.2

Пример 2.7.3. Рассмотрим программу

$$P(x) \leftarrow Q(f(x), g(x))$$
$$R(x) \leftarrow$$

Вводим константу a и получаем

$$H = \{a, f(a), g(a), f(f(a)), f(g(a)), g(f(a)), g(g(a)), \ldots\}.$$

□ 2.7.3

Пример 2.7.4.

$$S = \{\{P(x), Q(x)\}, \{T(x), \neg R(x)\}\}$$

Вводим новую константу c_0. Получаем $H_0 = \{c_0\}$. Так как в S не встречается ни одного функционального символа, имеем $H = H_0 = \{c_0\}$. □ 2.7.4

Теоретически, для того, чтобы проверить предложение на выполнимость, мы должны узнать, существует ли какая-нибудь интерпретация (в бесконечном множестве возможных интерпретаций), в которой истинно данное предложение. Задача решается намного проще, если мы ограничимся рассмотрением универсальных предложений, т. е. предложений, содержащих только кванторы всеобщности. В этом случае достаточно работать только с небольшим классом интерпретаций, называемых *эрбрановскими интерпретациями*.

Дадим формальное описание эрбрановской интерпретации.

Определение 2.7.5. Эрбрановская интерпретация.

Пусть S — множество дизъюнктов, и H — эрбрановский универсум, соответствующий S. *Эрбрановская интерпретация* $\mathcal{A}_H$ для S определяется следующим образом.

1. Областью интерпретации является H.

2. Интерпретация символа константы есть сама константа.

3. Интерпретация терма $f(t_1, \ldots, t_n)$, где f — функциональный символ, есть терм $f(t_1', \ldots, t_n')$, где $t_1', \ldots, t_n'$ — интерпретации термов $t_1, \ldots, t_n$ соответственно.

4. Интерпретация n-местного предикатного символа $P(t_1, \ldots, t_n)$ есть n-местное отношение $P(t_1', \ldots, t_n')$ на множестве H, где $t_1', \ldots, t_n'$ — интерпретации термов $t_1, \ldots, t_n$ соответственно.

□ 2.7.5

Пример 2.7.6. Для языка арифметики $\mathcal{L} = \{\leqslant, +, *, s, 0\}$, где s — функция следования, имеем

$$H_0 = \{0\},$$
$$H_1 = \{0, s(0)\},$$
$$H_2 = \{0, s(0), s(s(0))\},$$
$$\ldots$$

и наконец,

$$H = \{0, s(0), s(s(0)), s(s(s(0))), \ldots\}$$

Интерпретациями элементов языка $\mathcal{L}$, отличных от 0, являются соответствующие отношения на множестве H. Например, для s и $\leqslant$ согласно определению 2.5.2 мы имеем:

$$\varepsilon(s)\colon H \longmapsto H = a \longmapsto s(a) \in H \qquad \square\ 2.7.6$$

$$\varepsilon(\leqslant) \subseteq H^2\colon \varepsilon(\leqslant) = \{(0, s(0)), (s(0), s^2(0)), \ldots, (s^n(0), s^{n+1}(0)), \ldots\}$$

Далее мы сформулируем основную теорему об эрбрановских интерпретациях.

Теорема 2.7.7. *Пусть φ — универсальное предложение, и S — его множество дизъюнктов. Тогда φ выполнимо (в какой-либо интерпретации) тогда и только тогда, когда оно выполнимо в эрбрановской интерпретации.*

Доказательство. ($\Leftarrow$): Эрбрановская интерпретация является частным случаем интерпретации (очевидный факт).

($\Rightarrow$): Мы хотим доказать, что если предложение φ выполнимо в какой-либо интерпретации $\mathcal{A}$, то можно определить такие отношения на эрбрановском универсуме, которые будут обращать дизъюнкты из множества S в истину. Предположим, что φ выполнимо в интерпретации $\mathcal{A}$, которая имеет область интерпретации A. Для обозначения интерпретаций различных символов языка $\mathcal{L}$, к которому принадлежит φ, мы используем:

$\varepsilon(c) = \widehat{c} \in A$, где c — константа; $\varepsilon(f) = \widehat{f}$: $A^n \longmapsto A$, где f — n-местный функциональный символ;

$\varepsilon(t) = \varepsilon(f(t_1, \ldots, t_n)) = \widehat{f}(\widehat{t_1}, \ldots, \widehat{t_n})$, где $t = f(t_1, \ldots, t_n)$ — терм[1]);

$\varepsilon(R) = \widehat{R} \subseteq A^n$, где R — n-местный предикатный символ.

Для каждого n-местного предикатного символа R мы определяем отношение R_H на эрбрановском универсуме H (определения 2.7.5, 2.5.2 ii), 2.5.5) следующим образом:

$$R_H(t_1, \ldots, t_n) \Leftrightarrow (\widehat{t_1}, \ldots, \widehat{t_n}) \in \widehat{R}$$

Таким образом, мы имеем эрбрановскую интерпретацию $\mathcal{A}_H$.

Положим $A' = \{\widehat{t} \mid t \in H\}$. Строим интерпретацию $\mathcal{A}'$, беря в качестве области интерпретации A', а в качестве интерпретаций функциональных и предикатных символов — сужение на множестве A' соответствующих интерпретаций из $\mathcal{A}$. Из того, что $\mathcal{A} \models \varphi$, и интерпретации всех термов, встречающихся в φ, содержатся в A', следует, что $\mathcal{A}' \models \varphi$. Отсюда, по определению $\mathcal{A}_H$ следует $\mathcal{A}_H \models \varphi$.

Следовательно, предложение φ действительно выполнимо в эрбрановской интерпретации. □ 2.7.7

Согласно теореме 2.7.7, если предложение φ не выполнимо в эрбрановской интерпретации, то φ не выполнимо, т. е. не существует интерпретации, в которой оно истинно.

Другими словами:

С помощью эрбрановских интерпретаций мы сводим проблему невыполнимости множества дизъюнктов к проблеме невыполнимости множества основных примеров этих дизъюнктов на эрбрановском универсуме. Так как в основные примеры дизъюнктов не входят переменные, выполнимость может быть доказана с помощью методов логики высказываний, таких как метод семантических таблиц и метод резолюций.

Метод семантических таблиц Бета и метод резолюций являются алгоритмическими доказательствами (в отличие от обычных методов проверки выполнимости в PrL, пример 2.5.17). Этот результат в

[1]) Элемент множества A, являющийся интерпретацией терма t, обозначается $\widehat{t}$. — *Прим. перев.*

классической и современной литературе известен как теорема Эрбрана, независимо от того, в какой формулировке он приводится. Он будет анализироваться в следующем разделе при работе с выводами, построенными с помощью семантических деревьев.

§ 2.8. Метод систематических таблиц

В логике предикатов, как и в логике высказываний, мы можем определить, является ли предложение или множество предложений выполнимым. Методы, изученные нами ранее, можно применять и в контексте PrL.

Начнем с семантических таблиц[1]) [Fitt69, Fitt90, Meta85, Smul68]. Эти таблицы используются для нахождения истинностного значения сложного предложения языка PrL $\mathcal{L}$.

Определение 2.8.1 (исчисление замкнутых систематических таблиц). Предположим, мы имеем язык $\mathcal{L}$ и константы $c_0, c_1, \ldots$ этого языка собраны в один бесконечный список (значение этого списка станет понятным в построении 2.8.5).

Пусть $\sigma, \sigma_1, \sigma_2$ — предложения языка $\mathcal{L}$. Перечень семантических таблиц приведен на рис. 2.1. □ 2.8.1

Семантические таблицы для PrL получаются из соответствующих таблиц для PL путем добавления правил для кванторов. С помощью семантической таблицы

$$t(\forall x)\varphi(x)$$
$$\Big|$$
$$t\varphi(c)$$
для всех c

мы представляем факт «для истинности формулы $(\forall x)\varphi(x)$ необходимо, чтобы $\varphi(c)$ было истинным для всех констант c».

Соответственно, семантическая таблица

$$t(\exists x)\varphi(x)$$
$$\Big|$$
$$t\varphi(c)$$
для новой c

[1]) В этом разделе метод семантических таблиц излагается для собственного подмножества PrL, не содержащего функциональных символов — так называемой узкой логики предикатов — *Прим. перев*

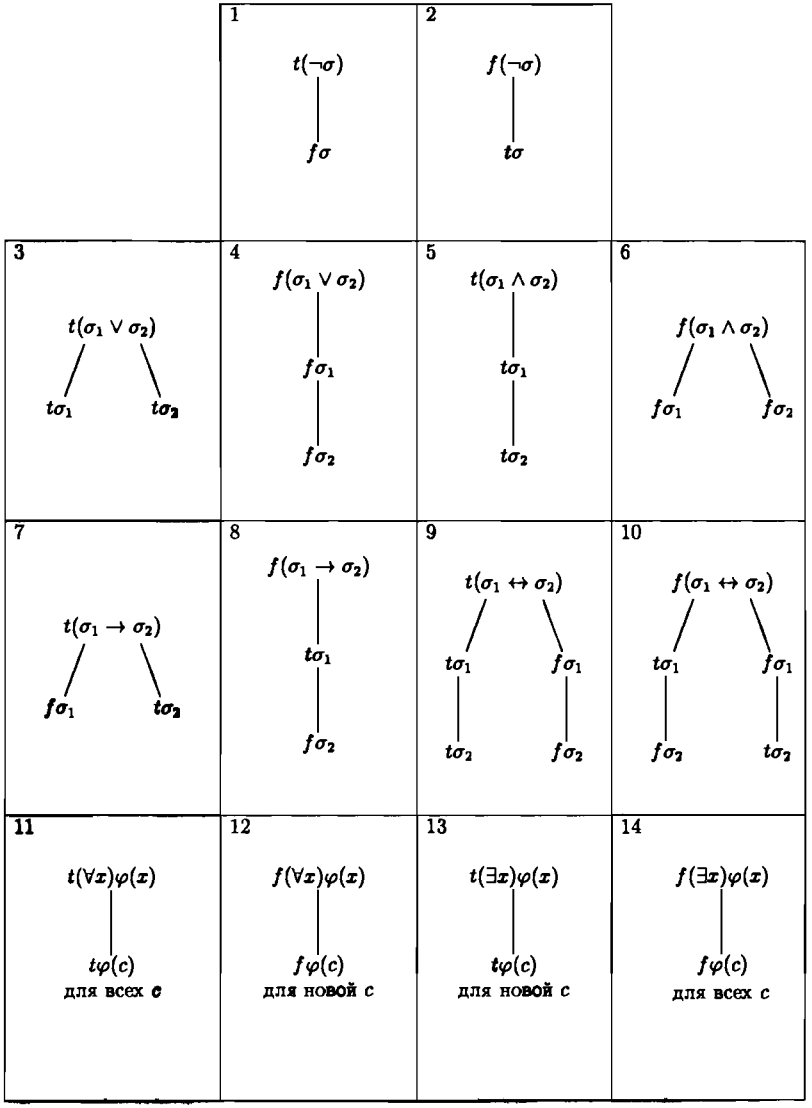

1
$t(\neg\sigma)$
$f\sigma$
2
$f(\neg\sigma)$
$t\sigma$
3
$t(\sigma_1 \vee \sigma_2)$
$t\sigma_1$ $t\sigma_2$
4
$f(\sigma_1 \vee \sigma_2)$
$f\sigma_1$
$f\sigma_2$
5
$t(\sigma_1 \wedge \sigma_2)$
$t\sigma_1$
$t\sigma_2$
6
$f(\sigma_1 \wedge \sigma_2)$
$f\sigma_1$ $f\sigma_2$
7
$t(\sigma_1 \to \sigma_2)$
$f\sigma_1$ $t\sigma_2$
8
$f(\sigma_1 \to \sigma_2)$
$t\sigma_1$
$f\sigma_2$
9
$t(\sigma_1 \leftrightarrow \sigma_2)$
$t\sigma_1$ $f\sigma_1$
$t\sigma_2$ $f\sigma_2$
10
$f(\sigma_1 \leftrightarrow \sigma_2)$
$t\sigma_1$ $f\sigma_1$
$f\sigma_2$ $t\sigma_2$
11
$t(\forall x)\varphi(x)$
$t\varphi(c)$
для всех c
12
$f(\forall x)\varphi(x)$
$f\varphi(c)$
для новой c
13
$t(\exists x)\varphi(x)$
$t\varphi(c)$
для новой c
14
$f(\exists x)\varphi(x)$
$f\varphi(c)$
для всех c

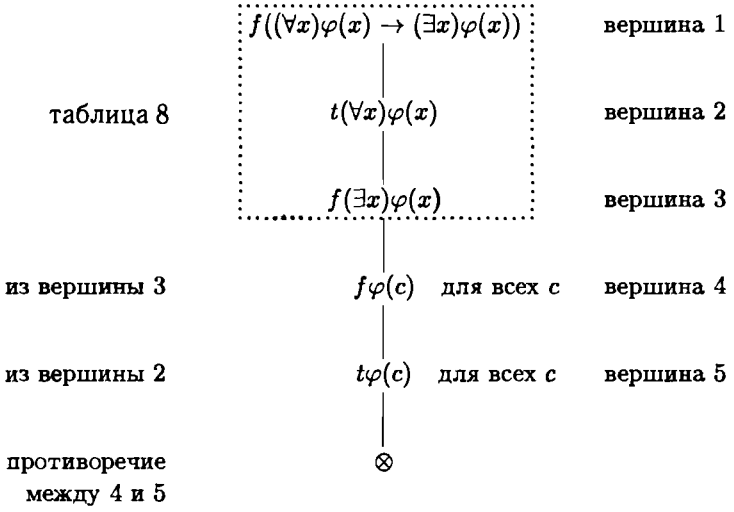
Рис. 2.1

представляет факт «для истинности формулы $(\exists x)\varphi(x)$ необходимо, чтобы существовала константа c, которая еще не появлялась в таблице, такая что $\varphi(c)$ принимает истинное значение».

Семантические таблицы для PrL называются *замкнутыми систематическими таблицами*, коротко ЗСТ. Строение замкнутых систематических таблиц для предложений PrL аналогично строению соответствующих таблиц для PL. Сначала рассмотрим несколько примеров.

Пример 2.8.2. Допустим, что нам нужно доказать общезначимость формулы σ: $(\forall x)\varphi(x) \to (\exists x)\varphi(x)$, где φ — предложение PrL.

Начнем с таблицы, в корне которой находится $f\sigma$.



В последней вершине семантической таблицы мы использовали ту же самую константу c для того, чтобы получить противоречие. Так разрешается делать, поскольку таблица для $(\forall x)\varphi(x)$ позволяет использовать любую константу.

Неформально, приведенный выше вывод по Бету означает, что предложение $(\forall x)\varphi(x) \to (\exists x)\varphi(x)$ общезначимо, так как все попытки опровергнуть его привели к противоречию. □ 2.8.2

Пример 2.8.3. ЗСТ для общезначимой формулы, фигурирующей в теореме 2.3.6 имеет вид, изображенный на рис. 2.2. □ 2.8.3

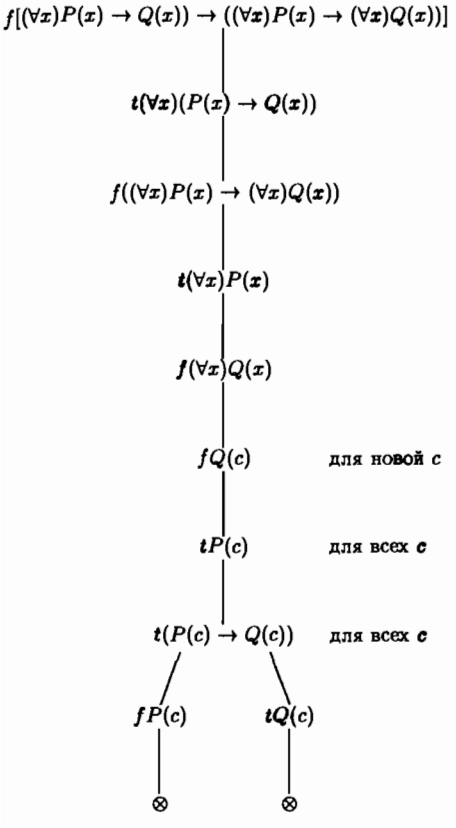


Рис. 2.2

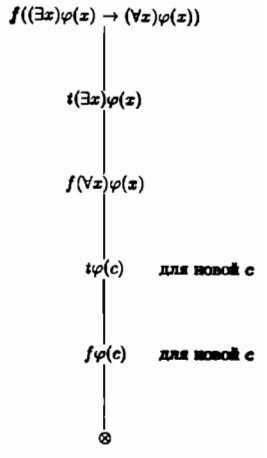


Рис. 2.3

Семантическая таблица для $t(\forall x)\varphi(x)$ (или двойственная таблица для $f(\exists x)\varphi(x)$) позволяет нам объявлять формулу $\varphi(c)$ истинной (соответственно ложной) для всех констант c. Семантическая таблица для $t(\exists x)\varphi(x)$ позволяет нам объявлять $\varphi(c)$ истинной только для тех констант, которые еще не встречались в систематической таблице.

Следующий пример демонстрирует, что могло бы произойти без этого ограничения.

Пример 2.8.4. Рассмотрим предложение $(\exists x)\varphi(x) \to$ $\to (\forall x)\varphi(x)$. Это предложение не общезначимо, так как существование x, такого что формула $\varphi(x)$ истинна, не влечет истинность $\varphi(x)$ для всех x (например, из существования x такого, что $x > 3$ не следует, что $x > 3$ верно для всех x). Однако, см. рис. 2.3.

В вершине 5 мы не имели права использовать ту же константу, что и в предыдущей вершине 4. Поэтому нам удалось «доказать», что предложение $(\exists x)\varphi(x) \to (\forall x)\varphi(x)$ общезначимо, тогда как это, очевидно, неверно. □ 2.8.4

Из-за таблиц 11 и 14 (см. рис. 2.1) систематическая таблица может быть бесконечной, если в одной из ее ветвей не удается по-

лучить противоречие (в примерах 2.8.2 и 2.8.3 нет необходимости выписывать все константы таблиц 11 и 14). Этот факт станет понятнее из последующего формального описания замкнутой систематической таблицы предложения φ.

Построение 2.8.5. Построение замкнутых систематических таблиц.

Построение начинается с того, что помеченная формула $f\varphi$ или $t\varphi$ помещается в корень таблицы. Далее выполняется индуктивная процедура.

Шаг n. Мы уже построили таблицу T_n.

Таблица T_n далее расширяется до новой таблицы T_{n+1}, с использованием некоторых вершин T_n.

Шаг $n+1$. Пусть X — неиспользованная неэлементарная вершина, самая левая среди наиболее удаленных от корня вершин.[1]) Если такой вершины не существует, то систематическая таблица является замкнутой. Если такая вершина есть, то мы строим таблицу T_{n+1}, продолжая каждую непротиворечивую ветвь, проходящую через вершину X, присоединением (в конец ветви) таблицы, соответствующей X. Рассмотрим новые случаи для PrL:

Случай 1. X имеет вид $t((\forall x)\varphi(x))$.

Пусть c_n — первая константа в списке всех констант языка, которая не встречалась в вершинах вида $t\varphi(c_n)$ ни в одной ветви, проходящей через X, Тогда мы добавляем $t\varphi(c_n)$ в конец каждой непротиворечивой ветви, проходящей через X, в соответствии с рис. 2.4[2]).

Рис. 2.4

Рис. 2.5

Случай 2: X имеет вид $f((\forall x)\varphi(x))$.

Пусть c_k — первая константа в списке, которая не встречалась ни в одной вершине каждой из ветвей, проходящей через X. Тогда мы добавляем $f\varphi(c_k)$ в конец каждой ветви, проходящей через X, в соответствии с рис. 2.5.

Случаи 3, 4: X имеет вид $f((\exists x)\varphi(x))$ и $t((\exists x)\varphi(x))$ соответственно.

---

[1]) Рассматриваемая здесь детерминированная стратегия построения замкнутых деревьев неполна, поскольку для некоторых общезначимых формул она приводит к построению бесконечных деревьев (см. теорему 2.10.2). В данном случае целесообразно отказаться от однозначного выбора неиспользованной вершины на $n+1$-ом шаге и полагать, что ее выбор производится недетерминированно. В этом случае мы получим корректную и полную, хотя и неоднозначную процедуру построения замкнутой семантической таблицы. Однако существуют полные корректные детерминированные стратегии построения семантических таблиц, с которыми можно ознакомиться в [Klee52]. — *Прим. перев.*

[2]) Вершина X остается неиспользованной. — *Прим. перев.*

Эти случаи двойственны случаям 1 и 2.

Говоря неформально, в случаях 1 и 3 (двойственно 2 и 4) мы хотим избавиться от повторений и объявляем $\varphi(c)$ истинной каждый раз для новой константы, выбирая ее из списка констант (что, вообще говоря, может продолжаться бесконечно). □ 2.8.5

Определение 2.8.6. *Замкнутая систематическая таблица*, коротко ЗСТ, есть объединение всех таблиц T_n из предыдущего построения, т. е.:

$$T = \bigcup_{n \in N} T_n \qquad \square\ 2.8.6$$

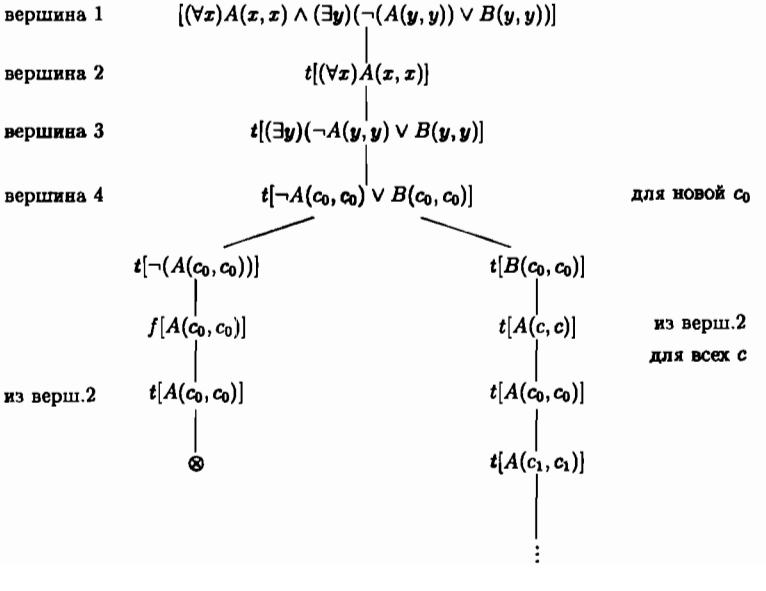
ЗСТ может иметь бесконечное число вершин, тогда как семантические таблицы для PL всегда конечны.

Определение 2.8.7. 1. ЗСТ называется *противоречивой*, если все ее ветви противоречивы.

2. Предложение σ *выводимо по Бету* (*опровержимо по Бету*), если существует противоречивая ЗСТ с корнем $f\sigma$ ($t\sigma$). Тот факт, что предложение σ выводимо по Бету обозначается $\vdash_B \sigma$.

3. Предложение σ *выводимо по Бету* из множества предложений PrL S, если существует противоречивая ЗСТ с корнем $f\sigma$ и следующей вершиной tP, где P представляет собой конъюнкцию предложений множества S. Этот факт обозначается $S \vdash_B \sigma$. □ 2.8.7

Пример 2.8.8.



В этом примере левая ветвь противоречива, тогда как правая ветвь продолжается бесконечно. □ 2.8.8

Исчисление семантических деревьев очень похоже на исчисление Бета. Мы рассмотрим неформальное описание задачи и метода ее решения.

Семантические деревья: неформальное описание

Пусть φ — предложение и S — соответствующее множество дизъюнктов. Если φ выполнимо, то оно истинно в некоторой эрбрановской интерпретации (теорема 2.7.7). Соответственно, все основные примеры дизъюнктов из S истинны в этой интерпретации. Следовательно, если предложение φ невыполнимо, любая попытка подтвердить все основные примеры дизъюнктов из S с помощью означивания основных атомов $R(t_1, t_2, \ldots, t_n)$, где термы $t_1, t_2, \ldots, t_n$ принадлежат эрбрановскому универсуму, обречена на неудачу. Этот факт может быть обнаружен за конечное число шагов путем построения конечного множества невыполнимых в эрбрановской интерпретации основных примеров. По теореме 2.7.7 эти основные примеры невыполнимы во всех интерпретациях.

Следовательно, вопрос состоит в том, как построить процедуру, которая, имея на входе предложение φ и соответствующее множества дизъюнктов S,

1) если φ невыполнимо, останавливается после конечного числа шагов, выдавая конечное множество основных примеров;

2) если φ выполнимо, процедура в общем случае не дает никакого ответа за конечный период времени, при этом она осуществляет построение эрбрановской интерпретации, на которой истинно φ.

Другими словами, с помощью этой процедуры мы хотим получить доказательство невыполнимости предложения или контрпример. При построении такой процедуры используются семантические деревья [ChLe73, Dela87].

Определение 2.8.9. *Деревом* называется структура $T = \{X, r\}$, где X есть множество вершин дерева T и r есть бинарное отношение на множестве X удовлетворяющее следующим условиям.

1. Если $x, y \in X$ и xry, то x называется *предшественником* y, а y называется *последователем* x.

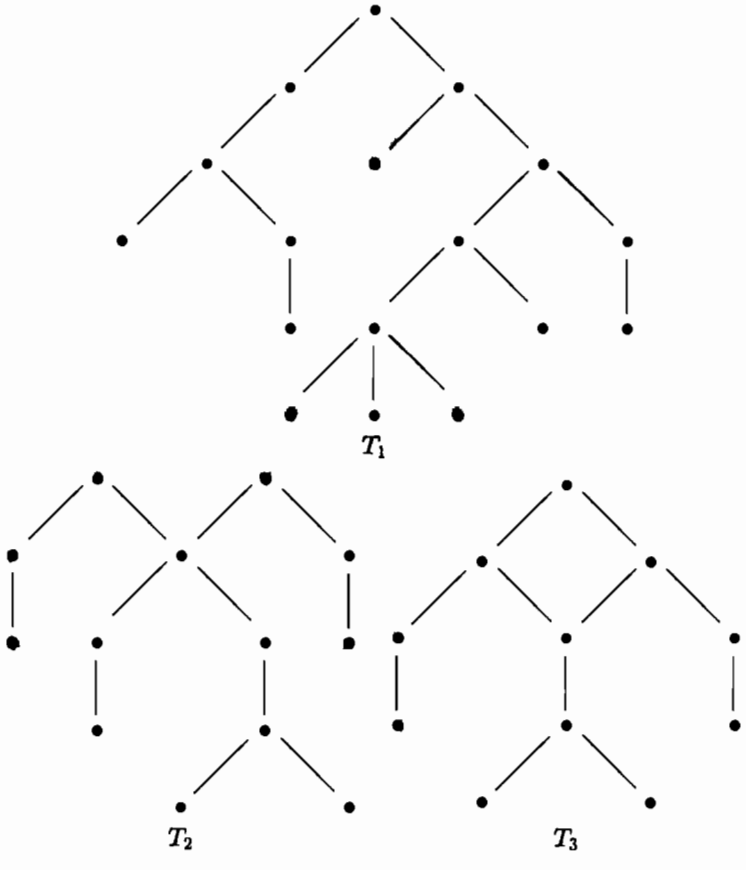
2. Дерево T содержит в точности одну вершину, не имеющую предшественников. Эта вершина называется *корнем* дерева T.

3. Каждая вершина дерева T, отличная от корня, имеет в точности одного предшественника.

Линия, соединяющая вершину с ее последователем, называется *дугой*. *Заключительной* называется вершина, для которой не существует последователей. Последовательность дуг, соединяющих за-

ключительную вершину дерева T с его корнем, называется *ветвью* дерева T. □ 2.8.9

Пример 2.8.10.



Граф T_1 является деревом. Обратите внимание на ветвление в нижней части, которое напоминает листву дерева, перевернутого корнем вверх. Граф T_2 *не является* деревом, он имеет два корня и, кроме того, вершину, у которой есть две предшествующие вершины. Граф T_3 тоже не является деревом, он содержит вершину, имеющую две предшествующие. □ 2.8.10

Замечание 2.8.11. Семантические таблицы для высказываний PL и замкнутые систематические таблицы для предложений PrL являются деревьями. □ 2.8.11

Определение 2.8.12. (семантические деревья). Пусть $S=\{C_1,\ldots,C_n\}$ — множество дизъюнктов, $P_1,\ldots,P_\lambda$ — атомы, входящие в дизъюнкты множества S, и $\{a_1,\ldots,a_n,\ldots\}$ — эрбрановский универсум множества S. *Семантическим деревом* для S называется дерево T, удовлетворяющее следующим условиям.

1. Корнем дерева T является произвольная точка. Вершины, отличные от корня представляют собой основные примеры атомов $P_1,\ldots,P_\lambda$ на универсуме $\{a_1,\ldots,a_n,\ldots\}$. Каждая вершина имеет в точности две последующих вершины, а именно $P_i(a_{i1},\ldots,a_{ik})$ и $\neg P_i(a_{i1},\ldots,a_{ik})$.

2. Каждая ветвь таблицы T, содержащая в точности основные примеры

$$P_{i_1}(a_{11},\ldots,a_{1k_1}),\ldots,P_{i_p}(a_{p1},\ldots,a_{pk_p})$$

представляет конъюнкцию

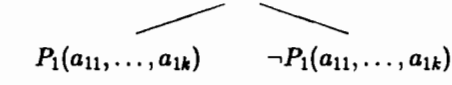
$$P_{i_1}(a_{11},\ldots,a_{1k_1})\wedge\ldots\wedge P_{i_p}(a_{p1},\ldots,a_{pk_p}).$$

3. Дизъюнкция всех конъюнкций ветвей дерева T является общезначимой формулой.

4. Если вершина дерева T имеет вид $P(a_{i1},\ldots,a_{ik})$, то ни на какой ветви дерева, проходящей через эту вершину, не может встречаться $\neg P(a_{i1},\ldots,a_{ik})$.

5. Если в процессе построения дерева T мы получили вершину k, которая вступает в противоречие с каким-либо основным примером одного из дизъюнктов $C_1,\ldots,C_n$ множества S, тогда k считается заключительной вершиной, и о соответствующей ветви говорят, что она *противоречива*. □ 2.8.12

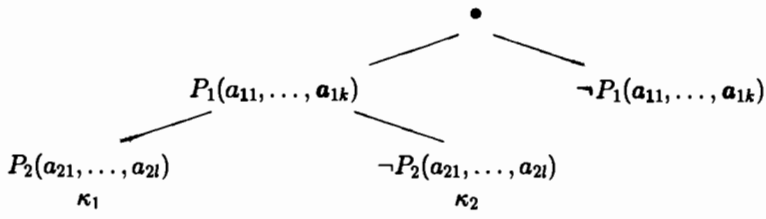
На практике, при построении семантического дерева множества S мы начинаем с $P_1(a_{11},\ldots,a_{1k})$:



Если один из дизъюнктов множества S содержит $P_1(a_{11},\ldots,a_{1k})$, то правая ветвь противоречива, $\neg P_1(a_{11},\ldots,a_{1k})$ является заключительной вершиной, и построение продолжается по левой ветви. Если один из дизъюнктов S содержит $\neg P_1(a_{11},\ldots,a_{1k})$, левая ветвь ветвь противоречива, $P_1(a_{11},\ldots,a_{1k})$ — заключительная вершина, и построение продолжается по правой ветви.

Предположим, что $P_1(a_{11},\ldots,a_{1k})$ — не заключительная вершина. Тогда мы строим вершину $P_2(a_{2l},\ldots,a_{2l})$. (Выбор атома для продолжения построения произволен.)

Ветвь $\varkappa_1$ представляет конъюнкцию $P_1(a_{11}, \ldots, a_{1k}) \wedge P_2(a_{21}, \ldots, a_{2l})$. Ветвь $\varkappa_2$ представляет конъюнкцию $P_1(a_{11}, \ldots, a_{1k}) \wedge \neg P_2(a_{21}, \ldots, a_{2l})$. Мы проверяем, содержит ли какой-либо дизъюнкт множества S один из атомов $\neg P_1(a_{11}, \ldots, a_{1k})$ или $\neg P_2(a_{21}, \ldots, a_{2l})$.



Если содержит, $\varkappa_1$ — противоречивая ветвь, и мы далее рассматриваем ветвь $\varkappa_2$. Таким образом, мы продолжаем проверку и построение. Наша цель — достичь заключительной вершины, исчерпав при этом все атомы из S.

Определение 2.8.13. 1. Множество дизъюнктов S называется *опровергаемым семантическим деревом*, если существует семантическое дерево для S, ветви которого противоречивы.

2. Ветвь $\varkappa$ семантического дерева множества дизъюнктов S называется *полной*, если для всех основных примеров $P(a_1, \ldots, a_n)$ каждого атома P в этой ветви содержатся или $P(a_1, \ldots, a_n)$, или $\neg P(a_1, \ldots, a_n)$. □ 2.8.13

Пример 2.8.14. Пусть дано φ: $(\forall x)[P(x) \wedge (\neg P(x) \vee Q(f(x))) \wedge \neg Q(f(x))]$. Тогда

$$S = \{\underbrace{\{P(x)\}}_{1}, \underbrace{\{\neg P(x), Q(f(x))\}}_{2}, \underbrace{\{\neg Q(f(x))\}}_{3}\},$$

и H = эрбрановский универсум $= \{a, f(a), f(f(a)), f(f(f(a))), \ldots\}$,

основные примеры $= \{P(a), Q(a), P(f(a)), Q(f(a)), \ldots\}$.

Семантическое дерево T_1 для S изображено на рис. 2.6

Посмотрим, почему ветвь $\varkappa_2$ противоречива: $\varkappa_2$ представляет конъюнкцию

$$\neg Q(f(a)) \wedge P(f(a)) \wedge Q(a) \wedge P(a).$$

Второй дизъюнкт множества S имеет вид $\{\neg P(x), Q(f(x))\}$. Поэтому, согласно S, мы требуем, чтобы формула

$$\neg P(x) \vee Q(f(x))$$

была истинной для каждого x. Однако, $\varkappa_2$ утверждает, что в эрбрановском универсуме существует значение переменной x равное a, для которого выполняется

$$\neg Q(f(a)) \wedge P(f(a)) \wedge Q(a) \wedge P(a).$$

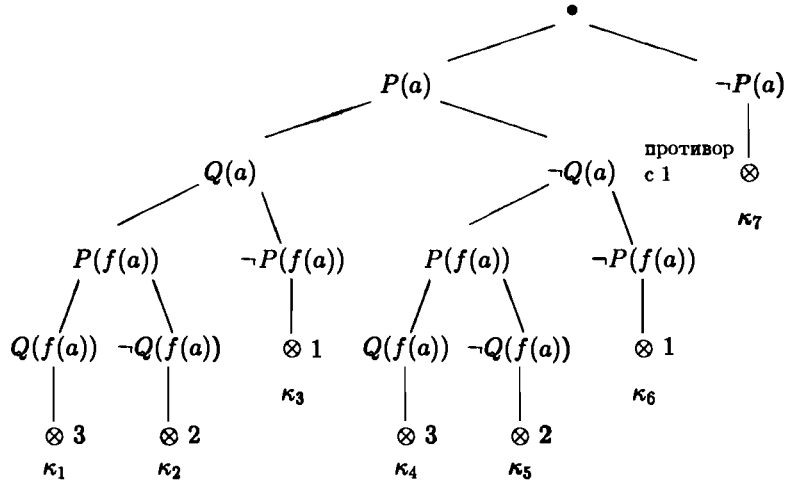


Рис. 2.6

Отсюда мы имеем также

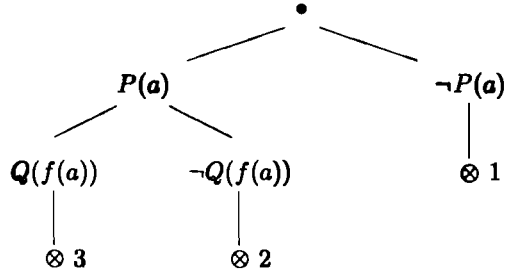
$$\neg Q(f(a)) \wedge P(a) \qquad \text{(по формуле } A \wedge B \to A\text{)},$$

и по правилу де Моргана

$$\neg(Q(f(a)) \vee \neg P(a)).$$

Итак, ветвь $\varkappa_2$ дает противоречие со вторым дизъюнктом множества S.

Как было отмечено ранее, мы сами выбираем порядок выбора очередного элемента из H для подстановки в атомы из множества S. Например, если мы выберем $Q(f(a))$ сразу вслед за $P(a)$, мы получим очень простое семантическое дерево T_2 для множества S, в котором все ветви противоречивы.



□ 2.8.1

Пример 2.8.15.

$$\varphi\colon (\forall x)(\forall z)[(\neg P(x) \vee Q(f(x), x)) \wedge P(g(b)) \wedge \neg Q(x, z)].$$

Тогда

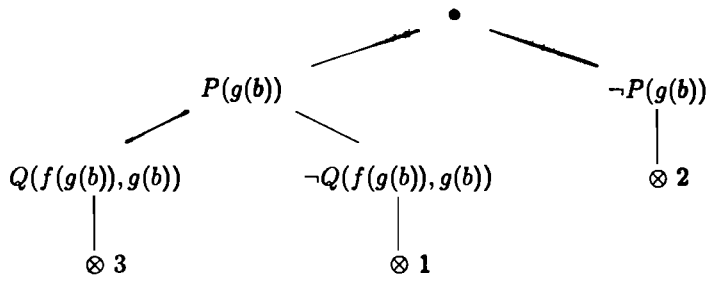
$$S = \{\underbrace{\{\neg P(x), Q(f(x), x)\}}_{1}, \underbrace{\{P(g(b))\}}_{2}, \underbrace{\{\neg Q(x, z)\}}_{3}\},$$

$$H = \{b, f(b), g(b), f(f(b)), \ldots\},$$

а основные примеры образуют множество

$$\{P(b), Q(f(b), b), P(g(b)), \ldots\}.$$

Семантическое дерево для S:



□ 2.8.15

Как говорилось ранее, число шагов, необходимых для достижения заключительной вершины противоречивой ветви, существенно зависит от порядка использования атомов при построении дерева.

В замечании 2.8.11 мы упоминали о том, что семантические таблицы и замкнутые систематические таблицы являются деревьями. По лемме Кёнига (лемма 1.11.9) каждое конечно ветвящееся дерево с бесконечным числом вершин имеет по крайней мере одну бесконечную ветвь. Следовательно, если S невыполнимо, то построение его семантического дерева приведет через конечное число шагов к обнаружению эрбрановской интерпретации $\mathcal{A}_H$ такой, что $\mathcal{A}_H \not\models S$. Если S выполнимо, то результатом соответствующего построения будет бесконечное дерево,[1]) каждая ветвь которого будет определять эрбрановскую интерпретацию, на которой выполняется S. Это заключение составляет содержание теоремы Эрбрана, которую мы докажем с помощью метода построения семантического дерева для данного множества дизъюнктов S.

[1]) Если бесконечен эрбрановский универсум H. — *Прим. перев.*

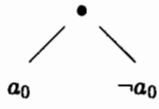
Теорема 2.8.16 (теорема Эрбрана). *Если множество дизъюнктов S невыполнимо, то S опровергается семантическим деревом.*

Доказательство. Опишем алгоритм построения семантического дерева для S. Мы формируем эрбрановский универсум для S, а также множество

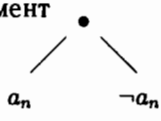
$$\{a_0, a_1, \ldots\}$$

основных примеров атомов из множества S.

Шаг 0: Строим дерево:



Шаг n. Добавляем к заключительной вершине каждой непротиворечивой ветви α фрагмент



Теперь предположим, что S не опровергается семантическим деревом. Тогда построение, описанное в нашем алгоритме никогда не закончится. Однако, в этом случае по лемме Кёнига дерево содержит бесконечную ветвь α. Для каждого основного примера a_n ветвь α в качестве некоторой своей вершины содержит либо a_n, либо его отрицание $\neg a_n$.

Определяем эрбрановскую интерпретацию следующим образом.

Для каждого n-местного предикатного символа P и термов $t_1, t_2, \ldots, t_n$, интерпретации которых принадлежат эрбрановскому универсуму для S, интерпретация P есть отношение

$$\varepsilon(P)(\varepsilon(t_1), \ldots, \varepsilon(t_n)),$$

где $P(t_1, \ldots, t_n)$ является какой-либо вершиной бесконечной ветви. Очевидно, что на этой интерпретации выполнимы все дизъюнкты множества S. Следовательно, множество S выполнимо. □ 2.8.16

Фактически, теорема Эрбрана предоставляет возможность при проверке выполнимости предложения или множества дизъюнктов S использовать методы логики высказываний, например, метод семантических таблиц или метод резолюций. Если S невыполнимо, то существует множество основных примеров дизъюнктов множества S, которое также будет невыполнимым. Это конечное множество состоит из высказываний PL, и его невыполнимость может быть обнаружена известными нам методами. Таким образом, для каждого множества дизъюнктов S мы начинаем перечислять все основные примеры дизъюнктов из S. Во время выполнения этой процедуры мы систематически проверяем выполнимость каждого конечного

множества основных примеров, пользуясь методами логики высказываний. Если S невыполнимо, тогда такая проверка покажет, что одно из конечных подмножеств невыполнимо. Если S выполнимо, то эта процедура будет продолжаться бесконечно.

Пример 2.8.17. Пусть дано предложение из примера 2.8.15.

$$\varphi\colon (\forall x)(\forall z)[(\neg P(x) \vee Q(f(x), x)) \wedge P(g(b)) \wedge \neg Q(x, z)].$$

Нужно определить, выполнимо φ или нет. Соответствующее множество дизъюнктов имеет вид:

$$S = \{\underbrace{\{\neg P(x), Q(f(x), x)\}}_{1}, \underbrace{\{P(g(b))\}}_{2}, \underbrace{\{\neg Q(x, z)\}}_{3}\}.$$

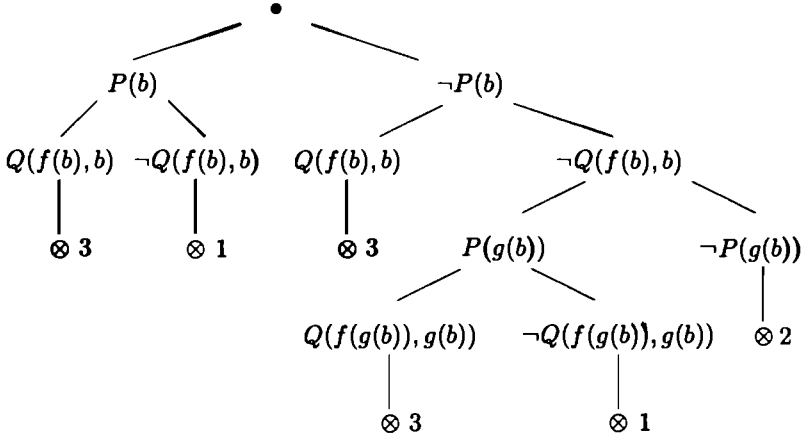
Эрбрановский универсум для S есть множество:

$$H = \{b, g(b), f(b), f(g(b)), g(f(b)), \ldots\}.$$

Множество основных примеров атомов из S:

$$\{P(b), Q(f(b), b), P(g(b)), Q(f(g(b)), g(b)), P(f(b)), \ldots\}.$$

Строим семантическое дерево для S в соответствии с методом из теоремы 2.8.16.



Это семантическое дерево служит доказательством того факта, что множество S невыполнимо, и предъявляет некоторое невыполнимое подмножество основных примеров дизъюнктов из S. Это подмножество тех и только тех основных примеров, которые были использованы для объявления некоторой ветви противоречивой. А именно:

$$\{\{\neg Q(f(b), b), \{\neg Q(f(g(b)), g(b))\}, \{P(g(b))$$
$$\{\neg P(b), Q(f(b), b)\}, \{\neg P(g(b)), Q(f(g(b)), g(b))\}\}$$

Первые два основных примера получены из дизъюнкта 3 множества S, третий — из дизъюнкта 2, и два последних — из дизъюнкта 1. Невыполнимость этого конечного множества действительно можно доказать методом резолюций в рамках логики высказываний. Введем обозначения:

$$A\colon Q(f(b), b)$$
$$B\colon Q(f(g(b)), g(b))$$
$$C\colon P(g(b))$$
$$D\colon P(b)$$

Теперь мы можем переписать наше конечное подмножество основных примеров дизъюнктов множества S следующим образом:

$$S' = \{\underbrace{\{\neg A\}}_{1}, \underbrace{\{\neg B\}}_{2}, \underbrace{\{C\}}_{3}, \underbrace{\{\neg D, A\}}_{4}, \underbrace{\{\neg C, B\}}_{5}\}$$

Используя метод резолюций, получаем

(1) $\neg A$
(2) $\neg B$
(3) C
(4) $\neg D, A$
(5) $\neg C, B$
(6) B резолюция (3) и (5)
(7) $\square$ резолюция (2) и (6)

В этом месте мы должны отметить, что невыполнимое множество основных примеров, определяемое нашим алгоритмом построения семантических таблиц, не всегда минимально.

Так в предыдущем примере подмножество

$$S_1 = \{\underbrace{\{\neg B\}}_{1}, \underbrace{\{C\}}_{2}, \underbrace{\{\neg C, B\}}_{3}\}$$

уже является невыполнимым:

(1) $\neg B$
(2) C
(3) $\neg C, B$
(4) B резолюция (2) и (3)
(5) $\square$ резолюция (1) и (4)

$\square$ **2.8.17**

Пример 2.8.18. Приведем пример выполнимого множества дизъюнктов. Пусть дано предложение

$$\varphi\colon (\forall x)[(\exists y)P(x, y) \to (\exists y)P(a, y)]$$

СНФ предложения φ имеет вид

$$\varphi\colon (\forall x)(\forall y)[\neg P(x, y) \vee P(a, f(x, y))].$$

(Почему?) Соответствующее предложению φ множество дизъюнктов имеет вид

$$S = \{\{\neg P(x, y), P[a, f(x, y)]\}\},$$

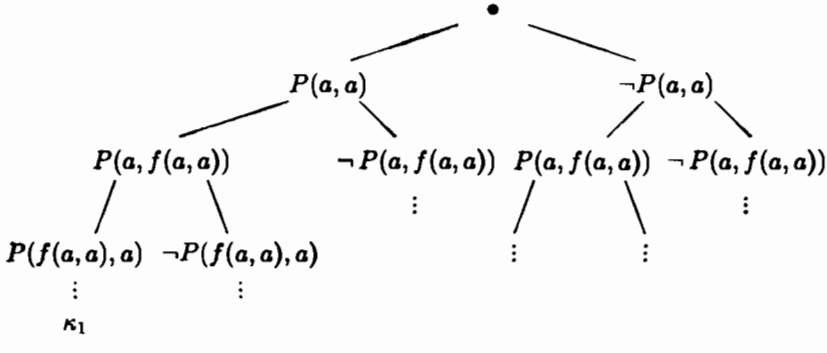
и соответствующий эрбрановский универсум —

$$H = \{a, f(a, a), f(a, f(a, a)), f(f(a, a), a), f(f(a, a))\, f(a, a)), \ldots\}.$$

Основными примерами атомов из S являются

$$\{P(a, a), P(a, f(a, a)), P(f(a, a), a), P(f(a, a), f(a, a)), \ldots\}.$$

Семантическое дерево для множества S имеет вид:



Это бесконечное семантическое дерево содержит ветви, которые по своей конструкции не являются противоречивыми, так как конъюнкция их атомов не противоречит ни одному дизъюнкту множества S. Например, ветвь $\varkappa_1$, которая не содержит отрицаний P, непротиворечива. Ветвь $\varkappa_1$ определяет эрбрановскую интерпретацию, в которой выполняется S. □ 2.8.18

§ 2.9. Унификация и резолюция в логике предикатов

Мы уже упоминали о предваренной и сколемовской нормальной форме предложения. Пусть φ — некоторое предложение языка PrL. Тогда последовательностью преобразований:

A: предваренная форма;

B: КНФ подформул, не содержащих кванторов;

C: сколемовская нормальная форма;

D: множество дизъюнктов;

мы можем привести φ к множеству дизъюнктов и определить его истинностное значение с помощью обычных методов PrL.

Пример 2.9.1. Пусть предложение φ уже находится в предваренной форме (A).

$$\varphi\colon (\forall x)(\exists y)(\exists z)((\neg P(x, y) \wedge Q(x, z)) \vee R(x, y, z))$$
$$\leftrightarrow (\forall x)(\exists y)(\exists z)((\neg P(x, y) \vee R(x, y, z)) \wedge (Q(x, z) \vee R(x, y, z))).$$

Теперь вводим сколемовские функциональные символы f, g и обозначаем $y = f(x)$, $z = g(x)$. Получаем

$$\varphi \leftrightarrow (\forall x)[(\neg P(x, f(x)) \vee R(x, f(x), g(x))) \wedge (Q(x, g(x)) \vee R(x, f(x), g(x)))].$$

И наконец, определяем множество дизъюнктов (D):

$$S = \{\{\neg P(x, f(x)), R(x, f(x), g(x))\}, \{Q(x, g(x)), R(x, f(x), g(x))\}\}.$$

□ 2.9.1

Когда предложение φ представлено в виде множества дизъюнктов, нам приходится работать с переменными и сколемовскими функциями. Справиться с этой проблемой можно с помощью семантических деревьев, определенных в предыдущем разделе. Используя семантические деревья, мы можем выбрать термы для подстановки вместо переменных из соответствующего эрбрановского универсума и применить метод резолюции для того, чтобы найти противоречие между основными примерами дизъюнктов. Такая процедура, однако, требует больших затрат времени. Более того, ее нельзя представить в виде хорошо структурированного алгоритма, который можно было бы реализовать на компьютере. Таким образом, нам нужен более четкий с алгоритмической точки зрения метод, который можно использовать для компьютерного программирования. Процедура унификации, к изучению которой мы переходим [ChLe73, Dela87, Fitt90, Lloy87, Thay88], как раз и предлагает необходимый метод для определения противоречивости множества дизъюнктов.

Унификация: неформальное описание

В определении 2.2.19 мы ввели понятие подстановки. Пусть нам даны следующие дизъюнкты:

$$C_1\colon\ \{P(f(x), y), Q(a, b, x)\} \qquad \text{и} \qquad C_2\colon\ \{P(f(g(c)), g(d))\}.$$

Мы хотим применить резолюцию к C_1 и C_2, подставив вместо x терм g(c) и вместо y — терм $g(d)$. Для этого нам необходимо

1) проверить, можно ли применить резолюцию к C_1 и C_2, и

2) найти подходящую подстановку, допускающую резолюцию.

Такую проверку и вычисление подстановки осуществляет алгоритм унификации. Посмотрим, как этот алгоритм унифицирует C_1 и C_2.

Шаг 1. Мы начинаем сравнивать соответствующие термы дизъюнктов слева направо, пока не встретим первые термы с одинаковыми функциональными символами, которые различаются переменными или константами. Мы создаем множество, состоящее из этих термов, называемое *множеством рассогласования*. Для C_1 и C_2 первым множеством рассогласования будет $\{x, g(c)\}$.

Шаг 2. Для каждой переменной множества рассогласования мы проверяем, встречается ли она в каком-нибудь другом терме этого же множества.

Шаг 3. Если на предыдущем шаге получен положительный ответ, то дизъюнкты не унифицируемы, и применение алгоритма оканчивается неудачей. Если ответ отрицательный, мы осуществляем подстановку вместо переменной из множества рассогласования терма из этого же множества. Для C_1 и C_2 мы применяем подстановку $\theta_1 = \{x/g(c)\}$. В результате C_1 и C_2 принимают вид

$$C_1^1 = \{P(f(g(c)), y), Q(a, b, x)\}$$
$$C_2^1 = C_2.$$

Шаг 4: Мы продолжаем двигаться направо, снова выполняя пункты 1–3. Новым множеством рассогласования будет $\{y, g(d)\}$. Применяем подстановку $\{y/g(d)\}$ и получаем

$$C_1^2 = \{P(f(g(c)), g(d)), Q(a, b, x)\}$$
$$C_2^2 = C_2.$$

Очевидно, мы можем применить резолюцию к дизъюнктам C_1^2 и C_2^2.

В конце концов, алгоритм заканчивает свою работу, выдавая в качестве ответа множество подстановок, позволяющих унифицировать C_1 и C_2 и затем применить к ним правило резолюции для

PL. Это множество подстановок называется *общим унификатором* (ОУ) дизъюнктов. Для C_1 и C_2 общим унификатором является множество $\theta = \{x/g(c), y/g(d)\}$. Если дизъюнкты не могут быть унифицируемы, алгоритм останавливается на шаге 2.

Унификация: формальное описание

Далее мы представим формальное описание алгоритма унификации и необходимые для этого определения.

Определение 2.9.2 (множество рассогласования). Пусть $S = \{C_1, C_2, \ldots, C_n\}$ — множество дизъюнктов. Множество $DS(S) = \{t_1, t_2, \ldots, t_n \mid t_1, \ldots, t_n$ — первые слева термы, стоящие на соответствующих местах в дизъюнктах $C_1, \ldots, C_n$, среди которых есть по крайней мере два различных $\}$ называется *множеством рассогласования* S. □ 2.9.2

Множество рассогласования не определяется однозначно, оно зависит от порядка следования атомов в дизъюнктах.

В примере 2.9.1 мы имеем

$$S = \{C_1, C_2\} = \{\{P(f(x), y), Q(a, b, x)\}, \{\neg P(f(g(c)), g(d))\}\}.$$

Первыми слева различными термами в атомах $P(f(x), y)$ и $P(f(g(c)), g(d))$ являются x и $g(c)$, таким образом, первым множеством рассогласования будет $DS_1 = \{x, g(c)\}$. После применения подстановки $\theta_1 = \{x/g(c)\}$ возникает несоответствие между термами y и $g(d)$, и следующее множество рассогласования, определяемое рассматриваемым алгоритмом будет иметь вид $DS_2 = \{y, g(d)\}$.

Определение 2.9.3. Пусть $X = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ — множество атомов или термов. Подстановка θ называется *унификатором* для X, если $\sigma_1\theta = \sigma_2\theta = \ldots = \sigma_n\theta$. □ 2.9.3

Определение 2.9.4. Унификатор θ называется *наиболее общим унификатором* (коротко НОУ), если для любого другого унификатора ψ существует подстановка γ такая, что $\psi = \theta\gamma$. □ 2.9.4

Для множества термов или атомов S наиболее общий унификатор определяется единственным образом [Lloy87][1]).

Пример 2.9.5. Рассмотрим множество атомов $S = \{Q(g(x), w), Q(y, b)\}$, где b — константа. Унификатором для S является подстановка

$$\psi = \{y/g(b), x/b, w/b\}$$

Если подстановку ψ применить к множеству S, получатся следующие основные примеры:

$$Q(g(x), w)\psi = Q(g(b), b) \qquad Q(y, b)\psi = Q(g(b), b).$$

[1]) Это утверждение вообще говоря неверно. Например, атомы $P(x)$ и $P(y)$ имеют, по крайней мере, два различных НОУ $\theta_1 = \{x/y\}$ и $\theta_2 = \{y/x\}$. — *Прим. перев.*

Если мы теперь положим $\gamma = \{x/b\}$ и $\theta = \{y/g(x), w/b\}$, мы можем легко показать, что $\psi = \theta\gamma$.

Кроме того, подстановка θ унифицирует рассматриваемые атомы. Таким образом, θ является наиболее общим унификатором. □ 2.9.5

Пример 2.9.6. Рассмотрим множество термов

$$C = \{f(x, g(x)), f(h(y), g(h(y)))\}.$$

Подстановка $\psi = \{x/h(g(c)), y/g(c)\}$ является унификатором C. Подстановка $\theta = \{x/h(y)\}$ также является унификатором C. Если мы положим $\gamma = \{y/c\}$, легко доказать, что $\psi = \theta\gamma$, а также что θ — наиболее общий унификатор. □ 2.9.6

Теперь мы можем перейти к формальному описанию алгоритма унификации.

Алгоритм 2.9.7 (алгоритм унификации).

Пусть $T = \{P_1, P_2, \ldots, P_n\}$ — множество атомарных формул.

Шаг 0. $\theta_0 = E$ (тождественная подстановка)

Шаг k. Мы уже построили подстановки $\theta_0, \theta_1, \ldots, \theta_k$.

Шаг $k+1$ (рекурсия).

1) Если $P_1\theta_1\theta_2\ldots\theta_k = P_2\theta_1\theta_2\ldots\theta_k = \ldots = P_n\theta_1\theta_2\ldots\theta_k$, алгоритм заканчивает работу, выдавая $\theta = \theta_1\theta_2\ldots\theta_k$ в качестве наиболее общего унификатора.

2) Если $P_i\theta_1\theta_2\ldots\theta_k \neq P_j\theta_1\theta_2\ldots\theta_k$ для некоторых i, j, тогда

а) строим множество рассогласования

$$DS(P_1\theta_1\ldots\theta_k, \ldots, P_n\theta_1\ldots\theta_k) = DS(T\theta_1\ldots\theta_k) = DS_k;$$

б) осуществляем *проверку вхождений* переменных, т. е. проверяем для каждой переменной $v \in DS_k$, содержится ли она в каком-либо другом элементе DS_k.

Если ПВ дает положительный ответ, мы останавливаемся и делаем заключение, что множество T не унифицируемо. Если множество DS_k вообще не содержит переменных, ПВ также дает положительный ответ. Если DS_k содержит переменную v и терм t, и ПВ дает отрицательный ответ, мы полагаем $\theta_{k+1} = \{v/t\}$, избавляясь таким образом от несоответствия между $P_1, \ldots, P_n$ в термах v, t.

Шаг $k+2$. Мы повторяем шаги $k+1, k+2$ для $k = k+1$.

Теорема 2.9.8 показывает, что если T унифицируемо, то алгоритм останавливается, выдавая в качестве ответа наиболее общий унификатор:

$$\text{НОУ} = \theta = \theta_1\theta_2\ldots\theta_m \qquad \square\ 2.9.7$$

Доказательство указанной теоремы не рассматривается в этой книге, но его можно найти в [Robi65, ChLe73].

Теорема 2.9.8 (Дж. А. Робинсон). *Применение алгоритма унификации к множеству атомов* $T = \{P_1, P_2, \ldots, P_n\}$ *дает следующий результат:*

если T *унифицируемо, то алгоритм останавливается, выдавая в качестве ответа НОУ множества* T*;*

если T *не унифицируемо, то алгоритм останавливается, объявляя, что унификатора не существует.* □ 2.9.8

Теорема утверждает, что если T унифицируемо, алгоритм унификации останавливается, определяя именно НОУ (а не произвольный унификатор) множества T.

Пример 2.9.9. Дано множество формул:

$$S = \{Q(a, x, f(g(z))), Q(z, f(y), f(y))\}$$

Определить, унифицируемо ли S. Если да, найти НОУ.

Шаг 0: Полагаем $\theta_0 = E$

Шаг 1: $S\theta_0 = S$
$DS(S\theta_0) = DS_1 = \{a, z\}$
ПВ негативна
Полагаем $\theta_1 = \{z/a\}$

Шаг 2: $S\theta_0\theta_1 = \{Q(a, x, f(g(a))), Q(a, f(y), f(y))\}$
$DS(S\theta_0\theta_1) = DS_2 = \{x, f(y)\}$
ПВ негативна
Полагаем $\theta_2 = \{x/f(y)\}$

Шаг 3: $S\theta_0\theta_1\theta_2 = \{Q(a, f(y), f(g(a))), Q(a, f(y), f(y))\}$
$DS(S\theta_0\theta_1\theta_2) = DS_3 = \{g(a), y\}$
ПВ негативна
Полагаем $\theta_3 = \{y/g(a)\}$

Шаг 4: $S\theta_0\theta_1\theta_2\theta_3 = \{Q(a, f(g(a)), f(g(a))), Q(a, f(g(a)), f(g(a)))\}$
$DS(S\theta_0\theta_1\theta_2\theta_3) = \varnothing$

Следовательно, S унифицируемо и его НОУ имеет вид

$$\text{НОУ} = \theta_0\theta_1\theta_2\theta_3 = \{z/a, x/f(g(a)), y/g(a)\} \quad \square\ 2.9.9$$

Пример 2.9.10. Дано множество формул:

$$S = \{Q(y, y), Q(z, f(z))\}.$$

Определить, унифицируемо ли S. Если да, найти НОУ.

Шаг 0: $\theta_0 = E$

Шаг 1: $S\theta_0 = S$
$DS(S\theta_0) = DS_1 = \{y, z\}$
ПВ негативна
Полагаем $\theta_1 = \{y/z\}$

Шаг **2:** $S\theta_0\theta_1 = \{Q(z, z), Q(z, f(z))\}$
$DS(S\theta_0\theta_1) = DS_2 = \{z, f(z)\}$
ПВ позитивна, z встречается в $f(z)$.

Следовательно, S не унифицируемо. □ 2.9.10

Здесь мы должны заметить, что во многих приложениях в целях большей эффективности процедура поиска вывода ПРОЛОГа, основанная на алгоритме унификации, не осуществляет проверку вхождений. Другими словами, она подставляет первый терм из текущего DS вместо первой переменной из этого множества. Это, конечно, может привести к неверным результатам, поэтому программисту приходится создавать в программе соответствующие средства защиты для избежания таких ошибок.

Теперь мы можем весьма просто описать метод резолюций для PrL.

Метод резолюций для PrL

Метод резолюций для PrL фактически представляет собой комбинацию алгоритма унификации для PrL и метода резолюций для PL. Точно так же, как в PL (определение 1.9.17), для множества дизъюнктов PrL S ***резолютивный вывод из*** S есть конечная последовательность дизъюнктов $C_1, \ldots, C_n$ такая, что для каждого C_i $(1 \leqslant i \leqslant n)$ либо $C_i \in S$, либо $C_i \in R(\{C_j, C_k\})$ $(1 \leqslant j, k \leqslant i)$, где $R(\{C_j, C_k\})$ — множество резольвент C_j и C_k.

Отметим здесь, что поскольку переменные во всех дизъюнктах считаются связанными кванторами всеобщности, мы можем переименовать эти переменные, чтобы избежать конфликтов во время процедуры унификации. Такое переименование называется ***нормализацией переменных.***

Рассмотрим пример.

Пример 2.9.11. Пусть даны следующие дизъюнкты:

$$C_1 = \{\neg P(x, y), \neg P(y, z), P(x, z)\};$$
$$C_2 = \{\neg P(u, v), P(v, u)\}.$$

Мы хотим сделать заключение

$$C_3 = \{\neg P(x, y), \neg P(z, y), P(x, z)\}.$$

Соответствующее обозначение для C_1, C_2 и C_3 в контексте PrL имеет вид

$$(\forall x)(\forall y)(\forall z)[P(x, y) \wedge P(y, z) \to P(x, z)] \quad \text{для } C_1$$
$$(\forall u)(\forall v)[P(u, v) \to P(v, u)] \quad \text{для } C_2$$
$$(\forall x)(\forall y)(\forall z)[P(x, y) \wedge P(z, y) \to P(x, z)] \quad \text{для } C_3$$

Метод I. Мы работаем непосредственно в контексте PrL. Дизъюнкты C_1, C_2 и C_3 эквивалентны предложениям

$$(\forall x)(\forall y)(\forall z)[\neg P(x, y) \vee \neg P(y, z) \vee P(x, z)] \tag{1}$$

$$(\forall u)(\forall v)[\neg P(u, v) \vee P(v, u)] \tag{2}$$

$$(\forall x)(\forall y)(\forall z)[\neg P(x, y) \vee \neg P(z, y) \vee P(x, z)] \tag{3}$$

Проводим переименование переменных предложения (2):

$$(\forall x)(\forall z)[\neg P(x, z) \vee P(z, x)] \tag{4}$$

По теореме 2.3.6 из (4) получаем

$$(\forall x)(\forall y)(\forall z)[\neg P(x, z) \vee P(z, x)] \tag{5}$$

Конъюнкция (1) и (5) по теореме 2.3.6 дает

$$(\forall x)(\forall y)(\forall z)[(\neg P(x, z) \vee P(z, x)) \wedge \wedge (\neg P(x, y) \vee \neg P(y, z) \vee P(x, z))] \quad ((6))$$

и по формуле

$$(\neg A \vee B) \wedge (A \vee C) \to (B \vee C) \tag{*}$$

выводимой в PrL (почему?), мы преобразуем (6) к виду

$$(\forall x)(\forall y)(\forall z)[(P(z, x) \vee \neg P(x, y) \vee \neg P(y, z)], \tag{7}$$

Проводим переименование переменные предложения (2) другим способом:

$$(\forall x)(\forall z)[\neg P(z, x) \vee P(x, z)], \tag{8}$$

и по теореме 2.3.6 из (8) получаем

$$(\forall x)(\forall y)(\forall z)[\neg P(z, x) \vee P(x, z)]. \tag{9}$$

Конъюнкция (9) и (7) согласно (*) дает

$$(\forall x)(\forall y)(\forall z)[\neg P(x, y) \vee \neg P(y, z) \vee P(x, z)], \tag{10}$$

Проводим переименование переменных в предложении (2):

$$(\forall y)\,(\forall z)[\neg P(z, y) \vee P(y, z)] \tag{11}$$

По теореме 2.3.6 из (11) получаем

$$(\forall x)\,(\forall y)\,(\forall z)[\neg P(z, y) \vee P(y, z)] \tag{12}$$

Конъюнкция (10) и (12) согласно (*) дает

$$(\forall x)\,(\forall y)\,(\forall z)[\neg P(x, y) \vee \neg P(z, y) \vee P(x, z)],$$

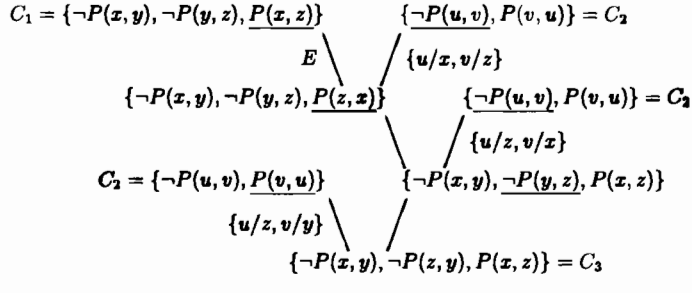
что и является искомой формулой.

Метод II. Построим вывод дизъюнкта C_3 методом резолюций:

(1) C_1	
(2) C_2	
(3) $\{\neg P(x, z), P(z, x)\}$	из (2), подстановка $\{u/x, v/z\}$
(4) $\{\neg P(x, y), \neg P(y, z), P(z, x)\}$	из (1) и (3), резолюция
(5) $\{\neg P(z, x), P(x, z)\}$	из (2), подстановка $\{u/z, v/x\}$
(6) $\{\neg P(x, y), \neg P(y, z), P(x, z)\}$	из (4) и (5), резолюция
(7) $\{\neg P(z, y), P(y, z)\}$	из (2), подстановка $\{u/z, v/y\}$
(8) C_3	из (6) и (7), резолюция

Метод II представляет собой более механистическую процедуру, которая, несомненно, быстрее и эффективнее, чем метод I.

Метод III. Вывод также может быть представлен в виде перевернутого дерева, корнем которого является C_3. Дизъюнкты, выбранные для резолюции, расположены в одном ряду, их переменные нормализованы. Применяемая унификация обозначена около ветви дерева, и атомы, к которым применяется резолюция, подчеркнуты.



Эта процедура, представленная деревом и также позволяющая вывести дизъюнкт C_3, используется в языке ПРОЛОГ. □ 2.9.11

Динамизм ПРОЛОГа и эффективность логического программирования в целом в области символических вычислений, не вызывает сомнений. В третьей главе дается более подробное изложение основ ПРОЛОГа.

Замечание 2.9.12. Метод резолюций фактически является развитием и упрощением соответствующих методов работы с логикой предикатов. Следовательно, если предложение σ логики PrL выводимо методом резолюций из множества S предложений PrL (сокращенно $S \vdash_R \sigma$), то σ выводима из S по определению 2.3.8. Формально:

$$S \vdash_R \sigma \Rightarrow S \vdash \sigma.$$

Обратное, однако, также выполняется для любого множества дизъюнктов S:

$$S \vdash \sigma \Rightarrow S \vdash_R \sigma \qquad \square\ 2.9.12$$

Теперь обратимся к результатам о корректности и полноте рассмотренных ранее методов.

§ 2.10. Корректность и полнота исчислений логики предикатов

Мы приведем здесь результаты, касающиеся полноты и корректности систем вывода для PrL так, как мы это делали для PL. Доказательства этих фактов очень похожи на соответствующие доказательства для PL, их можно найти в [ChLe73, Meta85, Smul68].

Корректность и полнота метода семантических таблиц

Запись $\vdash_B \sigma$ будет означать, что предложение σ выводима по Бету, а запись $\models \sigma$ — то, что предложение σ общезначимо. Сформулируем вспомогательную лемму, а также теоремы корректности и полноты исчисления Бета.

Лемма 2.10.1. *Пусть R — n-местный предикатный символ языка $\mathcal{L}$, и σ — предложение $\mathcal{L}$. Предположим, что существует непротиворечивая ветвь $\mathfrak{x}$ в систематической таблице с корнем $\psi\sigma$. Построим интерпретацию $\mathcal{A}$ следующим образом. В качестве области интерпретации возьмем произвольное множество $\{a_1, a_2, \ldots\} = A$, элементы которого находятся во взаимнооднозначном соответствии с символами констант языка $\mathcal{L}$. Интерпретация каждой константы c_i есть элемент $a_i = \varepsilon(c_i)$.*

Определяем отношение $\varepsilon(R) \subseteq A^h$:

$$\varepsilon(R)(a_{i_1}, a_{i_2}, \ldots, a_{i_n}) \Leftrightarrow tR(c_{i_1}, c_{i_2}, \ldots, c_{i_n}) \text{ — вершина ветви } \mathfrak{x}.$$

Тогда

а) *если $f\sigma$ является вершиной ветви $\mathfrak{x}$, то предложение σ ложно в интерпретации $\mathcal{A}$;*

б) *если $t\sigma$ является вершиной ветви $\varkappa$, то предложение σ истинно в интерпретации $\mathcal{A}$.* □ 2.10.1

Теорема 2.10.2 (полнота исчисления Бета). *Если предложение σ является следствием множества предложений S, то оно выводимо по Бету из S*[1]*):*

$$S \models \sigma \Rightarrow S \vdash_B \sigma \qquad \square\ 2.10.2$$

Следствие 2.10.3. *Если предложение σ общезначимо, то оно выводимо по Бету:*

$$\models \sigma \Rightarrow \vdash_B \sigma \qquad \square\ 2.10.3$$

Определение 2.10.4. Интерпретация $\mathcal{A}$ языка $\mathcal{L}$ *согласуется* с ветвью $\varkappa$ семантической таблицы, если

а) $t\sigma$ является вершиной $\varkappa \Rightarrow \mathcal{A} \models \sigma$,

б) $f\sigma$ является вершиной $\varkappa \Rightarrow \mathcal{A} \not\models \sigma$. □ 2.10.4

Лемма 2.10.5. *Пусть T — замкнутая систематическая таблица с корнем $f\sigma$, и $\mathcal{A}$ — сужение интерпретации языка $\mathcal{L}$ на множество символов констант, входящих в предложение σ, такое что $\mathcal{A} \models \neg\sigma$. Тогда существует по крайней мере одна ветвь таблицы T, которая согласуется с некоторым расширением $\mathcal{A}$.* □ 2.10.5

Теорема 2.10.6 (корректности исчисления Бета). *Если предложение σ выводимо по Бету из множества предложений S логики предикатов, то σ является следствием S:*

$$S \vdash_B \sigma \Rightarrow S \models \sigma \qquad \square\ 2.10.6$$

Следствие 2.10.7. *Если предложение σ выводимо по Бету, то оно общезначимо:*

$$\vdash_B \sigma \Rightarrow \models \sigma \qquad \square\ 2.10.7$$

Теорема 2.10.8 (компактности). *Множество предложений S выполнимо тогда и только тогда, когда каждое конечное подмножество S выполнимо.* □ 2.10.8

Корректность и полнота исчисления резолюций

Теорема 2.10.9 (корректности исчисления резолюций). *Пусть S — множество дизъюнктов, и $R^*(S)$ — множество резольвент S. Если множество $R^*(S)$ содержит пустой дизъюнкт, то S невыполнимо:*

$$\square \in R^*(S) \Rightarrow S \text{ невыполнимо} \qquad \square\ 2.10.9$$

[1]) То есть существует такой порядок выбора неиспользованных вершин в недетерминированной процедуре построения семантического дерева, при котором будет построена замкнутая таблица. — *Прим. перев.*

Лемма 2.10.10. *Если C_1' и C_2' являются основными примерами дизъюнктов C_1 и C_2, а C' есть резольвента C_1' и C_2', тогда существует резольвента C дизъюнктов C_1 и C_2 такая, что C' является основным примером C.* □ 2.10.10

Теорема 2.10.11 (полнота исчисления резолюций). *Пусть S — множество дизъюнктов, и $R^*(S)$ — множество резольвент S. Если S невыполнимо, то множество $R^*(S)$ содержит пустой дизъюнкт:*

$$S \text{ невыполнимо} \Rightarrow \square \in R^*(S) \qquad \square\ 2.10.11$$

Содержательное истолкование этой теоремы имеет следующий вид.

Чтобы доказать выполнимость или невыполнимость предложения методом резолюций, достаточно вывести пустой дизъюнкт из соответствующего множества дизъюнктов. В частности, если дано непротиворечивое множество дизъюнктов S, и мы хотим вывести предложение φ, то мы применяем метод резолюции к множеству дизъюнктов $S \cup S'$, где S' соответствует $\neg\varphi$. Если мы получим пустой дизъюнкт, противоречивость будет следствием предположения $\neg\varphi$, следовательно, мы докажем, что предложение φ общезначимо.

Пример 2.10.12. Пусть задано множество хорновских дизъюнктов из примера 2.4.7. Нужно с помощью метода резолюций найти ответ на следующие запросы:

а) «Что может украсть Питер?»;

б) «Может ли Питер украсть Мэри?».

Решение: Мы записываем данные хорновские дизъюнкты и запросы в теоретико-множественной форме. Таким образом, получаем следующее множество хорновских дизъюнктов:

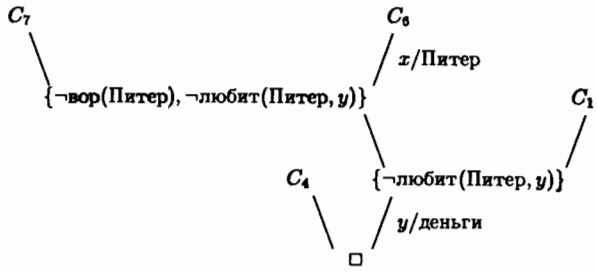
C_1: {вор(Питер)}
C_2: {любит(Мэри, шоколад)}
C_3: {любит(Мэри, вино)}
C_4: {любит(Питер, деньги)}
C_5: {любит(Питер, x), ¬любит(x, вино)}
C_6: {может_украсть(x, y), ¬вор(x), ¬любит(x, y)}

Запросы принимают следующий вид:

C_7: {¬может_украсть(Питер, x)}
C_8: {¬может_украсть(Питер, Мэри)}

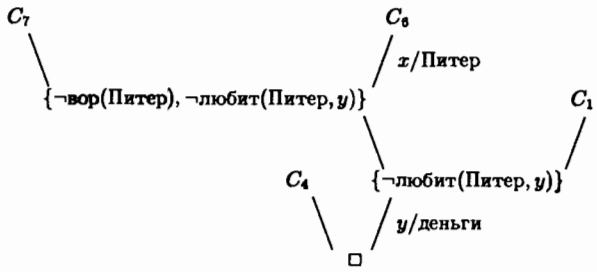
Тогда:

а)



Другими словами, имея в начале отрицание запроса, мы получили пустой дизъюнкт. Следовательно, C_7 не является справедливым требованием. Более того, в предыдущем выводе определяется значение переменной y, тем самым, мы получаем ответ $y =$ деньги на запрос (а), означающий, что Питер может украсть деньги.

б)



Таким образом, из C_8 можно также вывести пустой дизъюнкт. Следовательно, Питер может украсть Мэри! □ 2.10.12

Замечание 2.10.13. Если во время выполнения процедуры резолюции алгоритм унификации не может найти наиболее общий унификатор, другими словами, мы не можем вывести □ из наших данных, то цель называется *неуспешной*. В противном случае, цель называется *успешной* (замечание 1.9.9). В частности, цель «может_украсть(Питер, Мэри)» из примера 2.10.12 является успешной. □ 2.10.13

Полнота аксиоматического исчисления

Доказательство следующей теоремы о выводе в аксиоматической системе выходит за рамки этой книги. Его можно найти, например, в [Klee52, Mend64, Rasi74, RaSi70].

Теорема 2.10.14 (полноты и корректности аксиоматического исчисления, Гёдель, 1930 г.). *Формула φ логики предикатов выводима из множества S предложений PrL тогда и только тогда, когда φ является следствием S. Формально:*

$$S \vdash \varphi \Leftrightarrow S \models \varphi \qquad \square\ 2.10.14$$

Следствие 2.10.15. *Формула φ выводима из аксиом PrL тогда и только тогда, когда φ общезначима. Формально:*

$$\vdash \varphi \Leftrightarrow \models \varphi \qquad \square\ 2.10.15$$

§ 2.11. Проблема разрешимости логики предикатов

Многие математические задачи являются частным случаем некоторой общей схемы и поэтому могут быть решены посредством применения обобщенной процедуры к конкретной задаче. Например, мы можем ответить на вопрос

«делится ли полином $f(x)$ на полином $g(x)$?»

с помощью алгоритма деления, пытаясь разделить конкретный полином $f(x)$ на конкретный полином $g(x)$. Если остаток от деления равен нулю, ответом на вопрос будет «да». Если остаток отличен от нуля, ответом будет «нет».

Определение 2.11.1. Метод, позволяющий ответить «да» или «нет» на произвольный частный случай общего запроса, называется *разрешающей процедурой*. Задача нахождения такого метода для некоторого общего запроса называется *проблемой разрешимости* для этого запроса. $\square$ 2.11.1

Многие проблемы разрешимости в математике или не могут быть решены в общем виде, или имеют только частные решения, то есть могут быть решены при выполнении некоторых условий. Одной из таких проблем является проблема разрешимости для произвольной логики L. Логика L определяется своим языком, который состоит из логических и специальных символов, а также аксиомами и правилами, с помощью которых мы можем выводить и анализировать правильно построенные предложения логики L.

Определение 2.11.2. Проблема разрешимости для логики L состоит в нахождении алгоритмической процедуры, с помощью которой мы можем решить для каждого правильно построенного предложения логики L, выводимо оно в L или нет, другими словами можно ли его доказать в L или нет. $\square$ 2.11.2

Проблема разрешимости в логике высказываний решается с помощью таблиц истинности: если нам дано высказывание A, мы строим таблицу истинности A и проверяем, является ли A тавтологией. Если высказывание A есть тавтология, то по следствию 1.12.2 A выводимо в PL; если же A не является тавтологией, то оно не выводимо в PL. Следовательно, справедлива следующая теорема:

Теорема 2.11.3. *Проблема разрешимости для PL имеет положительное решение.* □ 2.11.3

Не так проста ситуация в PrL. По следствию 2.10.15 мы знаем, что формула φ языка $\mathcal{L}$ логики предикатов выводима тогда и только тогда, когда φ общезначима, т. е. истинна во всех интерпретациях языка $\mathcal{L}$. Однако, число интерпретаций языка $\mathcal{L}$ бесконечно, и мы, конечно, не в состоянии все их проверить. Следующая теорема известна с 1936 года [Chur36, Turi37].

Теорема 2.11.4. *Проблема разрешимости для PrL не имеет решения. Другими словами, не существует алгоритмической процедуры, позволяющей определить, выводима данная формула PrL или нет.* □ 2.11.4

Хотя проблема разрешимости для PrL не может быть решена в общем виде, для некоторых частных случаев существуют решения [Klee52, Chur56].

Теорема 2.11.5. *Проблема разрешимости имеет решение для формул в предваренной нормальной форме, в которых все кванторы существования следуют за кванторами всеобщности. Иначе говоря, существует алгоритмическая процедура, позволяющая определить, выводима формула вида*

$$\underbrace{(\forall x_1)\ldots(\forall x_k)}_{\text{только } \forall}\underbrace{(\exists y_1)\ldots(\exists y_l)}_{\text{только } \exists}\varphi,$$

или нет. □ 2.11.5

Теорема 2.11.6. *Проблема разрешимости имеет решение для всех формул, содержащих только предикаты, степень которых не превосходит единицы, т. е. ноль-местные и одноместные предикаты.* □ 2.11.6

В [Chur56] приведена аналитическая таблица, представляющая все известные частные решения проблемы разрешимости для PrL.

Наряду с попытками найти частные решения проблемы разрешимости для PrL, определенные усилия были затрачены на решение проблемы выполнимости формулы PrL, то есть на поиск алгоритмической процедуры, позволяющей определить, выполнима ли данная формула PrL или нет. Теоремы 2.6.6 (Лёвенгейм, Сколем), 2.7.7 и 2.10.6 утверждают, что проблема выполнимости для PrL имеет решение в частных случаях.

§ 2.12. Упражнения

2.12.1. Определите, чем являются следующие выражения: термами, формулами или ни тем, ни другим:

а) Майкл;
б) математик (x);
в) число (6);
г) является_планетой (x);
д) $(3+1)+10$;
е) $(\forall x)$ [число$(x) \wedge x = x + x$];
ж) $=[+(x+y), z]$;
з) $(x+y)+j^2$;
и) лучшая книга;
к) ненавидит$(x, y) \wedge$ любит (x, y).

Решение.

а) терм;
б) формула;
в) формула;
г) формула;
д) терм;
е) формула;
ж) формула;
з) терм;
и) ни то, ни другое;
к) формула.

2.12.2. Найдите свободные вхождения переменных в следующих формулах:

а) $(\forall x)P(x, y) \rightarrow (\forall z)Q(z, x)$;
б) $Q(z) \rightarrow \neg(\forall x)(\forall y)P(x, y, a)$;
в) $(\forall x)P(x) \wedge (\forall y)Q(x, y)$.

Решение.

а) y, x (x также имеет связанное вхождение); б) z; в) x.

2.12.3. Найдите свободные вхождения переменных в следующих формулах. Какие из этих формул — предложения?

а) $(\forall x)(\forall y)(\forall z)[x > y \wedge y > z] \rightarrow (\exists w)[w > w]$;
б) $(\exists x)$ (красный $(x)) \vee [(\forall y)$голубой $(y) \vee$ желтый $(x)]$;
в) $x + x = x + x$;
г) $(\exists y)[x + x = x + x]$;
д) $(\exists x)(\exists y)$ [учитель$(x, y) \wedge$ учит_предмету (x, y, z)].

Решение. а) Свободных вхождений нет. Формула является предложением.

б) Переменная x имеет свободное вхождение во вторую дизъюнктивную подформулу.

в) Переменная x имеет свободное вхождение.

г) Переменная x имеет свободное вхождение.

д) Переменная z имеет свободное вхождение.

2.12.4. Используя арифметический символ «<» (меньше) и язык логики предикатов, сформулируйте следующие фразы:

а) Существует число x, меньшее, чем 5 и большее, чем 3.

б) Для любого числа x существует число y, меньшее x.

в) Для любого числа x существует число y, большее x.

г) Для любых чисел x и y, суммы $x+y$ и $y+x$ равны.

д) Для любого числа x, существует такое число y, что для любого z, если разность $z-5$ меньше, чем y, то разность $x-7$ меньше 3.

Р е ш е н и е. Введем функции $-(x, y)$ и $+(x, y)$, выражающие разность и сумму x, y, и предикат «число (x)», обозначающий, что x является числом. Тогда предложения запишутся следующим образом:

а) $(\exists x)[\text{число}\,(x) \wedge <(x, 5) \wedge <(3, x)]$;

б) $(\forall x)(\exists y)[\text{число}\,(x) \wedge \text{число}\,(y) \wedge <(y, x)]$;

в) $(\forall x)(\exists y)[\text{число}\,(x) \wedge \text{число}\,(y) \wedge <(x, y)]$;

г) $(\forall x)(\exists y)[(\text{число}\,(x) \wedge \text{число}\,(y)) \rightarrow =(+(x, y), +(y, x))]$;

д) $(\forall x)(\exists y)(\forall z)[\text{число}\,(x) \wedge \text{число}\,(y) \wedge \text{число}\,(z) \wedge$
$\wedge (<(-(z, 5), y) \rightarrow <(-(x, 7), 3))]$.

2.12.5. Пусть θ, ψ, ξ — подстановки, E — тождественная подстановка, и σ — атомарная формула. Докажите, что

а) $\theta E = E\theta = \theta$;

б) $(\sigma\theta)\psi = \sigma(\theta\psi)$;

в) $(\theta\psi)\xi = \theta(\psi\xi)$.

Р е ш е н и е. б) Пусть $\theta = \{u_1/s_1, \ldots, u_m/s_m\}$, $\psi = \{v_1/t_1, \ldots, v_n/t_n\}$. Сначала докажем, что для всех термов τ верно свойство

$$(\tau\theta)\psi = \tau(\theta\psi). \qquad (*)$$

Доказательство проведем индукцией по длине τ.

1. Если τ — константа, тогда $(*)$, очевидно, справедливо.

2. Если τ — переменная x, то

а) если $x \notin \{u_1, \ldots, u_m\} \cup \{v_1, \ldots, v_n\}$, тогда $x(\theta\psi) = x = (x\theta)\psi$;

б) если $x \in \{u_1, \ldots, u_m\}$, то существует $1 \leqslant i \leqslant m$ такое, что $(x\theta)\psi = s_i\psi = x(\theta\psi)$;

в) если $x \in \{v_1, \ldots, v_n\} - \{u_1, \ldots, u_m\}$, то существует $1 \leqslant j \leqslant n$ такое, что $(x\theta)\psi = v_i\psi = x(\theta\psi)$.

3. Если τ — это терм $f(t_1, \ldots, t_k)$, тогда $(*)$ справедливо по предположению индукции и по определению 2.2.19.

Тогда если σ — атом, то по свойству $(*)$ и определению 2.2.19 (почему?) верно $(\sigma\theta)\psi = \sigma(\theta\psi)$.

2.12.6. Определите, какие из следующих пар формул являются вариантами:

а) $P(f(x, y), g(z), a)$ и $P(f(y, x), g(u), a)$;

б) $P(x, x)$ и $P(x, y)$.

Решение. а) да (почему?); б) нет (почему?).

2.12.7. Докажите, что

а) если $free(x, a, A)$, то $\vdash A(x/a) \to (\exists x)A$;

б) $\vdash (\forall x)A \to A$;

в) $\vdash A \to (\exists x)A$;

г) $\vdash (\forall x)A \to (\exists x)A$;

д) $\vdash (\forall x)(\exists y)(x = y)$;

е) $\vdash (\forall x)(\forall y)(x = y \to y = x)$;

ж) $\vdash (\forall x)(\forall y)(\forall z)[(x = y \wedge y = z) \to x = z]$.

Решение. а) Если $free(x, a, A)$, то $free(x, a, \neg A)$. По аксиоме 4 верно заключение

$$\vdash (\forall x)\neg A \to \neg A(x/a).$$

Затем по аксиоме 3 и правилу Modus Ponens получим, что

$$\vdash \neg\neg A(x/a) \to \neg(\forall x)\neg A.$$

Формула $\neg\neg A \leftrightarrow A$ является тавтологией. Из замечания 2.3.4(4) и теоремы о подстановке эквивалентных формул следует, что

$$\vdash A(x/a) \to \neg(\forall x)\neg A.$$

Тогда также

$$\vdash A(x/a) \to (\exists x)A.$$

б) По аксиоме 4, так как $free(x, a, A)$.

г) Ввиду в) верно $\vdash A \to (\exists x)A$, а ввиду б) $\vdash (\forall x) \to A$. Тогда, используя тавтологию,

$$(B \to C) \to [(A \to B) \to (A \to C)]$$

мы получим

$$\vdash (A \to (\exists x)A) \to [((\forall x)A \to A) \to ((\forall x)A \to (\exists x)A)].$$

Двухкратное применение правила Modus Ponens приведет нас к желаемому результату.

е) Пусть A — это формула $x = y$. Тогда по а) имеем

$$\vdash x = x \to (\exists y)(x = y).$$

По аксиоме 6, замечанию 2.3.7 и правилу Modus Ponens мы получим, что $\vdash (\exists y)(x = y)$. Тогда по правилу обобщения верно $\vdash (\forall x)(\forall y)(x = y)$.

е) По аксиоме 7 мы имеем

$$\vdash x = y \to (x = x \to y = x).$$

Применим тавтологию

$$[A \to (B \to C)] \leftrightarrow [B \to (A \to C)],$$

что даст нам

$$\vdash x = x \to (x = y \to y = x).$$

По аксиоме 6 и правилу Modus Ponens мы получим $\vdash x = y \to y = x$. Двукратное применение правила Modus Ponens приведет нас к цели.

ж) По аксиоме 7 имеем

$$\vdash y = x \to (y = z \to x = z),$$

а по е)

$$\vdash x = y \to (y = z \to x = z).$$

Применим тавтологию

$$[A \to (B \to C)] \leftrightarrow [(A \wedge B) \to C],$$

что даст нам

$$\vdash (x = y \wedge y = z) \to x = z.$$

Затем три раза применим правило Modus Ponens.

2.12.8. Определите теоретико-множественную форму следующих утверждений.

а) Джои любит еду.

б) Яблоки — это еда.

в) Цыплята — это еда.

г) Все, что можно съесть и при этом не умереть, — это еда.

д) Билл ест и все еще жив.

е) Мэри ест все, что ест Билл.

Р е ш е н и е. Введем следующие предикаты:

Любит(x, y) — x любит y; Пища(x) — x является едой;

Ест(x, y) — x ест y; Живет(x) — x жив.

Тогда

а) Утверждение выражается формулой

$$(\forall x)[\text{Пища}(x) \to \text{Любит}(\text{Джон}, x)] \leftrightarrow$$
$$\leftrightarrow (\forall x)[\neg \text{Пища}(x) \vee \text{Любит}(\text{Джои}, x)].$$

Множество $S = \{\{\neg\text{Пища}(x), \text{Любит}(\text{Джон}, x)\}\}$ — ее теоретико-множественная форма.

б) Пища(Яблоки). Следовательно, $S = \{\{\text{Пища}(\text{Яблоки})\}\}$.

г) Утверждение можно записать формулой

$$(\forall x)(\forall y)[\text{Ест}(x, y) \wedge \text{Живет}(x) \rightarrow \text{Пища}(y)] \leftrightarrow$$
$$\leftrightarrow (\forall x)(\forall y)[\neg\text{Ест}(x, y) \vee \neg\text{Живет}(x) \vee \text{Пища}(y)].$$

Следовательно, $S = \{\{\neg\text{Ест}(x, y), \neg\text{Живет}(x), \text{Пища}(y)\}\}$.

2.12.9. Предположим, что драконы существуют на самом деле, и мы поймали одного из них. Запишите следующие предложения, данные на обыденном языке, в виде хорновских дизъюнктов.

а) Всякий дракон, живущий в зоопарке, несчастен.

б) Всякое животное, встречающееся с вежливыми людьми, счастливо.

в) Все люди, посещающие зоопарк, вежливы.

г) Животные, живущие в зоопарке, встречаются с людьми, посещающими зоопарк.

Р е ш е н и е. Введем следующие предикаты:

Дракон (x): x — это дракон;

Человек (x): x — человек;

Счастлив (x): x счастлив;

Живет (x, y): x живет в y;

Животное (x): x является животным;

Вежливый (x): x вежлив;

Посещает (x, y): x посещает y;

Встречает (x, y): x встречается с y.

Тогда

а) $\leftarrow$ Дракон(x), Счастлив (x), Живет $(x,$ Зоопарк$)$;

б) Счастлив$(x) \leftarrow$ Животное (x), Человек (y), Вежливый (y), Встречает (x, y);

в) Вежливый $(x) \leftarrow$ Человек (x), Посещает $(x,$ Зоопарк$)$;

г) Встречает $(x, y) \leftarrow$ Животное (x), Человек (y), Живет $(x,$ Зоопарк$)$, Посещает $(y,$ Зоопарк$)$.

2.12.10. Сформулируйте следующие высказывания, данные на обыденном языке, в виде хорновских дизъюнктов.

а) x — мать y, если x — женщина и родитель y.

б) x — отец y, если x — мужчина и родитель y.

в) x — человек, если его родитель — человек.

г) x — человек, если отец x — человек.

д) Никто не родитель самому себе.

2.12.11. Пусть у нас есть два пустых сосуда в 5 и 7 литров. Мы хотим найти последовательность действий, которая приведет к тому, что в семилитровом сосуде будет ровно 4 литра воды. Разрешены только два действия:

1) наполнить сосуд;

2) переливать воду из одного сосуда в другой, пока один из них не будет полон или пуст.

Сформулируйте задачу и разрешенные действия с помощью хорновских дизъюнктов.

Р е ш е н и е. Введем предикат

Содержат (u, v): 7-литровый сосуд содержит u литров,
а 5-литровый — v литров.

Если мы предположим, что отношения $x+y=z$ и $x \leqslant y$ уже определены, то задачу можно выразить следующими предложениями:

C_1: Содержат$(0, 0) \leftarrow$

C_2: $\leftarrow$ Содержат$(4, y)$

C_3: Содержат$(7, y) \leftarrow$ Содержат(x, y)

C_4: Содержат$(x, 5) \leftarrow$ Содержат(x, y)

C_5: Содержат$(0, y) \leftarrow$ Содержат(x, y)

C_6: Содержат$(x, 0) \leftarrow$ Содержат(x, y)

C_7: Содержат$(0, y) \leftarrow$ Содержат(u, v), $u+v=y$, $y \leqslant 5$

C_8: Содержат$(x, 0) \leftarrow$ Содержат(u, v), $u+v=x$, $x \leqslant 7$

C_9: Содержат$(7, y) \leftarrow$ Содержат(u, v), $u+v=w$, $7+y=w$

C_{10}: Содержат$(x, 5) \leftarrow$ Содержат(u, v), $u+v=w$, $5+x=w$

Предложение C_1 соответствует начальной ситуации, а C_2 — наша цель. Предложения C_3 и C_4 обозначают соответственно наполнение первого и второго сосуда. Предложения C_5 и C_6 обозначают опустошение первого и второго сосудов, C_7 и C_8 — переливание воды, пока один из сосудов не будет пуст, а C_9 и C_{10} — пока один из сосудов не наполнится.

2.12.12. Пусть $\mathcal{L}=\{a, b, P\}$ — некоторый язык логики предикатов, а $\mathcal{A}$ — такая интерпретация $\mathcal{L}$ на области $D=\{a, b\}$, что (a, a) и (b, b) принадлежат $\varepsilon(P)$, а (a, b) и (b, a) не принадлежат $\varepsilon(P)$. Определите, являются ли следующие формулы истинными в $\mathcal{A}$:

а) $(\forall x)(\exists y)P(x, y)$;

б) $(\exists x)(\forall y)P(x, y)$;

в) $(\forall x)(\exists y)[P(x, y) \to P(y, x)]$;

г) $(\forall x)(\forall y)P(x, y)$;

д) $(\exists y)\neg P(a, y)$;

е) $(\forall x)P(x, x)$.

Р е ш е н и е. Мы обозначаем факт $(a, b) \in \varepsilon(P) \subseteq D^2$, приписывая значение t (истина) выражению $P(a, b)$, и факт $(a, b) \notin \varepsilon(P) \subseteq D^2$, приписывая значение f (ложь) этому же выражению. Тогда для заданной интерпретации мы можем использовать следующую таблицу значений предиката P на области интерпретации:

$P(a, a)$	$P(a, b)$	$P(b, a)$	$P(b, b)$
t	f	f	t

а) Если x принимает значение a или b, то по таблице мы можем убедиться, что существуют такие значения y (a и b соответственно), что предикат $P(x, y)$ имеет значение «истина». Таким образом, формула $(\forall x)(\exists y)P(x, y)$ истинна в $\mathcal{A}$.

б) Формула ложна (почему?).

в) Построим таблицу, в которой даны значения подформул формулы в) для всевозможных значений x и y:

(x, y)	(y, x)	$P(x, y)$	$\to$	$P(y, x)$
(a, a)	(a, a)	t	t	t
(a, b)	(b, a)	f	t	f
(b, a)	(a, b)	f	t	f
(b, b)	(b, b)	t	t	f

Следовательно, формула в) истинна в $\mathcal{A}$.

г) Если x придать значение a, а y — значение b, то формула $P(x, y)$, очевидно, не истинна в $\mathcal{A}$.

д) Формула $\neg P(a, y)$ истинна, когда y принимает значение b.

2.12.13. Пусть дано предложение σ: $(\exists x)P(x) \to (\forall x)P(x)$.

а) Докажите, что предложение σ всегда истинно в интерпретации, область которой состоит только из одного элемента.

б) Найдите интерпретацию с областью, состоящей из двух элементов, в которой предложение σ ложно.

Р е ш е н и е. а) Пусть $\mathcal{A}=(\{a\}, \varepsilon(P))$ — любая интерпретация языка $\mathcal{L}=\{P\}$, на котором выражена σ. Тогда по определению 2.5.5 мы получим

$$\begin{aligned}\mathcal{A}\models(\exists x)P(x)\to(\forall x)P(x)&\Leftrightarrow\mathcal{A}\models(\forall x)P(x)\text{ или }\mathcal{A}\not\models(\exists x)P(x)\\&\Leftrightarrow\mathcal{A}\models P(a)\text{ или }\mathcal{A}\models\neg P(a)\\&\Leftrightarrow\mathcal{A}\models P(a)\vee\neg P(a).\end{aligned}$$

Но последняя дизъюнкция верна согласно тавтологии $A\vee\neg A$ (см. замечание 2.3.4).

б) Пусть $\{a, b\}$ — область искомой интерпретации, причем $a\neq b$. Ложность предложения σ в $\mathcal{A}$ предполагает, что $\mathcal{A}\not\models(\forall x)P(x)$ и $\mathcal{A}\models(\exists x)P(x)$. Построим следующую таблицу значений предиката P для a и b:

$P(a)$	$P(b)$
t	f

В этой интерпретации предложение σ не истинно (почему?).

2.12.14. Найдите интерпретацию с областью, состоящей из двух элементов $\{a, b\}$, в которой предложение $(\exists x)(\exists y)P(x, y)$ было бы истинным, а предложение $(\forall x)(\exists y)P(x, y)$ — ложным.

2.12.15. Пусть $\mathcal{L}=\{\Delta, c_1, c_2, \ldots, c_9\}$ — это язык, где Δ — символ двуместного предиката, а $c_1, \ldots, c_9$ — константы; и пусть $\mathcal{A}=(\{1, 2, \ldots, 9\}, /)$ — это интерпретация, где $/$ — отношение делимости, $\varepsilon(\Delta)=/$ и $\varepsilon(c_i)=i$, $i=1, \ldots, 9$. Определите, какие из следующих выражений истинны в заданной интерпретации[1]). Ответ аргументируйте.

а) $(\forall y)\Delta(c_1, y)$;

б) $(\forall x)[\Delta(x, c_5)\leftrightarrow(x=c_1)\vee(x=c_5)]$;

в) $(\exists x)(\forall y)\Delta(x, y)$;

г) $(\exists x)(\forall y)\Delta(y, x)$.

Р е ш е н и е. По определению 2.5.5 мы получим

а) истинно (почему?);

б) истинно (почему?);

в) истинно (для какого x?);

г) ложно (почему?).

2.12.16. Пусть $\mathcal{L}=\{\leqslant, =, +, \cdot, 0, 1\}$ — это язык арифметики.

а) Напишите на языке $\mathcal{L}$ предложение $P(x)$, которое в обычном математическом смысле интерпретируется как «x — простое число».

б) Используя $P(x)$ как предикат, выразите в языке $\mathcal{L}$, что сумма двух простых чисел, отличных от 2, — четное число.

Р е ш е н и е. а) Число x является простым тогда и только тогда, когда x и 1 — единственные делители x. Тогда:

[1]) Выражение $\Delta(x, y)$ интерпретируется как «y делится на x» — *Прим. перев.*

$P(x) \leftrightarrow (\forall y)(\forall z)[(x = y \cdot z) \rightarrow (x = y \wedge z = 1) \vee (x = z \wedge y = 1)]$;

б) $(\forall x)(\forall y)[(P(x) \wedge P(y) \wedge \neg(x = +(1, 1)) \wedge \neg(y = +(1, 1))) \rightarrow$
$\rightarrow (\exists z)(+(x, y) = +(z, z))]$.

2.12.17. Пусть дан язык логики предикатов $\mathcal{L} = \{=, \leqslant\}$ и набор аксиом

$$\Sigma = \{(\forall x)[x \leqslant x], \quad (\forall x)(\forall y)[(x \leqslant y) \wedge (y \leqslant x) \rightarrow (x = y)],$$
$$(\forall x)(\forall y)(\forall z)[(x \leqslant y) \wedge (y \leqslant z) \rightarrow (x \leqslant z)],$$
$$(\forall x)(\forall y)[\neg(x = y) \wedge (x \leqslant y) \rightarrow$$
$$\rightarrow (\exists z)[(x \leqslant z) \wedge (z \leqslant y) \wedge \neg(x = z) \wedge \neg(y = z)]],$$
$$(\forall x)(\forall y)[(x = y) \vee (x \leqslant y) \vee (y \leqslant x)]\}.$$

Определите такие две различные интерпретации $\mathcal{A}_1$ и $\mathcal{A}_2$ языка $\mathcal{L}$, что $\mathcal{A}_1 \models \Sigma$ и $\mathcal{A}_2 \not\models \Sigma$.

Р е ш е н и е. Первые три аксиомы из Σ вводят отношение упорядоченности, четвертая утверждает, что упорядоченность плотна, а пятая — то, что упорядоченность полна. Мы знаем, что действительные, рациональные и натуральные числа с обычным отношением порядка полностью упорядочены, и, более того, упорядоченность плотна для действительных и рациональных чисел, но не плотна для натуральных чисел (почему?).

2.12.18. Пусть $\mathcal{A}_1 = (Q, \varepsilon_1(P))$ и $\mathcal{A}_2 = (N, \varepsilon_2(P))$ — две интерпретации языка $\mathcal{L} = \{P\}$ логики предикатов, где $\varepsilon_1(P)$ и $\varepsilon_2(P)$ — это обычные отношения порядка рациональных чисел Q и натуральных чисел N, где $\theta(\mathcal{A}_1)$ и $\theta(\mathcal{A}_2)$ — соответствующие теории (см. определение 2.5.10).

а) Найдите два таких предложения σ_1 и σ_2 языка $\mathcal{L}$, что

$$\mathcal{A}_1 \models \sigma_1 \wedge \sigma_2 \quad \text{и} \quad \mathcal{A}_2 \not\models \sigma_1 \vee \sigma_2.$$

б) Определите, какие из следующих предложений

$$\theta(\mathcal{A}_1) = \theta(\mathcal{A}_2), \quad \theta(\mathcal{A}_1) \not\subseteq \theta(\mathcal{A}_2), \quad \theta(\mathcal{A}_2) \subseteq \theta(\mathcal{A}_1)$$

истинны, и аргументируйте ответ.

Р е ш е н и е. а) По определению 2.5.5 как σ_1, так и σ_2 должны быть истинными в $\mathcal{A}_1$, а в $\mathcal{A}_2$ предложения σ_1 и σ_2 не могут быть истинными. Пусть σ_1 — выражение, обозначающее плотность упорядочения:

σ_1: $(\forall x)(\forall y)[\neg(x = y) \wedge (x \leqslant y) \rightarrow$
$\rightarrow (\exists z)[(x \leqslant z) \wedge (z \leqslant y) \wedge \neg(x = z) \wedge \neg(y = z)]]$.

Другое важное различие между Q и N состоит в том, что существует наименьший элемент в N, а у Q нет наименьшего элемента. Пусть σ_2 — предложение, утверждающее, что наименьшего элемента нет:

$$\sigma_2\colon \neg(\exists x)(\forall y)P(x, y).$$

Тогда $\mathcal{A}_1 \models \sigma_1 \wedge \sigma_2$ и $\mathcal{A}_2 \not\models \sigma_1 \vee \sigma_2$ (почему?).

б) $\mathcal{A}_1 \models \sigma_1$, но $\mathcal{A}_2 \not\models \sigma_1$. Следовательно, $\theta(\mathcal{A}_1) \neq \theta(\mathcal{A}_2)$.

$\mathcal{A}_2 \models \neg\sigma_2$ (почему?), но $\mathcal{A}_1 \not\models \neg\sigma_2$. Следовательно, $\theta(\mathcal{A}_1) \not\subseteq \theta(\mathcal{A}_2)$.
$\mathcal{A}_2 \models \neg\sigma_1$ (почему?), но $\neg\sigma_1 \notin \theta(\mathcal{A})$ (почему?). Следовательно, $\theta(\mathcal{A}_2) \not\subseteq \theta(\mathcal{A}_1)$.

2.12.19. Приведите к предваренной нормальной форме следующие предложения:

а) $(\forall x)P(x) \to (\exists x)Q(x)$;

б) $(\forall x)(\forall y)[(\exists z)P(x, y) \wedge P(x, z)] \to (\exists u)Q(x, y, u)$;

в) $(\forall x)(\forall y)[(\exists z)P(x, y, z) \wedge [(\exists u)Q(x, u) \to (\exists u)Q(y, u)]]$.

Р е ш е н и е. Используем формулы с (1) по (6), (9) и (10) из раздела 2.5.

в) $(\forall x)(\forall y)[(\exists z)P(x, y, z) \wedge [(\exists u)Q(x, u) \to (\exists u)Q(y, u)]]$

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

$\leftrightarrow (\forall x)(\forall y)[(\exists z)P(x, y, z) \wedge [(\forall u)\neg Q(x, u) \vee (\exists w)Q(y, w)]]$

$\leftrightarrow (\forall x)(\forall y)(\exists z)[P(x, y, z) \wedge (\forall u)(\exists w)[\neg Q(x, u) \vee Q(y, w)]]$

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

2.12.20. Определите теоретико-множественную форму следующих предложений:

σ: $(\exists x)(\forall y)(\exists z)[(P(x, y) \vee \neg Q(x) \vee R(z)) \wedge$
$\wedge (\neg P(x, y) \vee \neg Q(x)) \wedge (\neg P(x, y) \vee R(z))]$;

σ': $\neg(\forall x)(\exists y)[P(x, y) \to Q(y)]$;

φ: $\neg[(\forall x)P(x) \to (\exists y)(\forall z)Q(y, z)]$.

Р е ш е н и е. Предложение σ уже находится в предваренной нормальной форме. Так что нам нужно определить сколемовскую нормальную форму (СНФ).

σ^*: $(\forall y)(\exists z)[(P(a, y) \vee \neg Q(a) \vee R(z)) \wedge$
$\wedge (\neg P(a, y) \vee \neg Q(a)) \wedge (\neg P(a, y) \vee R(z))]$ (почему?)

σ^{**}: $(\forall y)[(P(a, y) \vee \neg Q(a) \vee R(f(y))) \wedge$
$\wedge (\neg P(a, y) \vee \neg Q(a)) \wedge (\neg P(a, y) \vee R(f(y)))]$,

где a — новая константа, а $f(y)$ — новая одноместная функция. Тогда по определению 2.6.10 теоретико-множественная форма σ имеет вид:

$S_\sigma = \{\{P(a, y), \neg Q(a), R(f(y))\},$
$\{\neg P(a, y), \neg Q(a)\}, \{\neg P(a, y), \boldsymbol{R(f(y))}\}\}$.

Теперь преобразуем φ в ПНФ:

$$\begin{aligned}\varphi &\leftrightarrow \neg[\neg(\forall x)P(x) \vee (\exists y)(\forall z)Q(y,z)]\\ &\leftrightarrow (\forall x)P(x) \wedge \neg(\exists y)(\forall z)Q(y,z)\\ &\leftrightarrow (\forall x)P(x) \wedge (\forall y)(\exists z)\neg Q(y,z)\\ &\leftrightarrow (\forall x)(\forall y)[P(x) \wedge (\exists z)\neg Q(y,z)]\\ &\leftrightarrow (\forall x)(\forall y)(\exists z)[P(x) \wedge \neg Q(y,z)].\end{aligned}$$

Соответствующая сколемовская форма имеет вид

$$\varphi^*\colon\ (\forall x)(\forall y)[P(x) \wedge \neg Q(y, f(x,y))],$$

где $f(x,y)$ — новая двухместная функция. Множество

$$S_\varphi = \{\{P(x)\}, \{\neg Q(y, f(x,y))\}\}$$

есть теоретико-множественная форма φ.

2.12.21. Даны следующие предложения:

F_1: Все, кто сберегают деньги в банке, получают проценты.

F_2: Если бы процентов не было, никто бы не сберегал деньги в банке.

Введем следующие предикаты:

$A(x,y)$: x сберегает в банке y; $T(x)$: x — это проценты;

$K(x,y)$: x зарабатывает y; $X(x)$: x — это деньги.

Сформулируйте F_1 и F_2 на языке, определенном вышеуказанными предикатами, и определите сколемовскую форму для F_1 и $\neg F_2$.

2.12.22. Пусть $(G, \#)$ — группа. Опишите выбранными Вами предикатами свойства группы G, а именно, что G замкнута относительно $\#$, операция $\#$ ассоциативна, имеется единичный элемент и обратные элементы. Определите СНФ формул, которые выражают эти свойства.

Р е ш е н и е. Введем предикат $P(x,y,z)$, который интерпретируется следующим образом: $x\#y = z$. Тогда формула

$$\sigma_1\colon\ (\forall x)(\forall y)(\exists z)P(x,y,z)$$

выражает, что G замкнута относительно $\#$. Ассоциативность выражается формулой

$$\begin{aligned}\sigma_2\colon\ &(\forall x)(\forall y)(\forall z)(\forall u)(\forall v)(\forall w)[P(x,y,u) \wedge\\ &\wedge P(y,z,v) \wedge P(u,z,w) \to P(x,v,w)] \wedge\\ &\wedge (\forall x)(\forall y)(\forall z)(\forall u)(\forall v)(\forall w)[P(x,y,u) \wedge\\ &\wedge P(y,z,v) \wedge P(x,v,w) \to P(u,z,w)].\end{aligned}$$

Существование единичного элемента:

$$\sigma_3\colon\ (\exists y)\,(\forall x)[P(x, y, x) \wedge P(y, x, x)].$$

Существование обратного элемента:

$$\sigma_4\colon\ (\exists y)\,(\forall x)\,(\exists z)[P(x, y, x) \wedge P(y, x, x) \to P(x, z, y) \wedge P(z, x, y)].$$

СНФ этих предложений суть:

$$\sigma_1^*\colon\ (\forall x)\,(\forall y)P(x, y, f(x, y)),$$

где $f(x, y)$ — новая функция от двух переменных;

$$\begin{aligned}\sigma_2^*\colon\ & (\forall x)(\forall y)(\forall z)(\forall u)(\forall v)(\forall w)\\ & [[P(x, y, u) \wedge P(y, z, v) \wedge P(u, z, w) \to P(x, v, w)] \wedge\\ & \wedge [P(x, y, u) \wedge P(y, z, v) \wedge P(x, v, w) \to P(u, z, w)]]\\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(почему?)}\\ \leftrightarrow & (\forall x)(\forall y)(\forall z)(\forall u)(\forall v)(\forall w)\\ & [[\neg(P(x, y, u) \wedge P(y, z, v) \wedge P(u, z, w)) \vee P(x, v, w)] \wedge\\ & \wedge [\neg(P(x, y, u) \wedge P(y, z, v) \wedge P(x, v, w)) \vee P(u, z, w)]]\\ \leftrightarrow & (\forall x)(\forall y)(\forall z)(\forall u)(\forall v)(\forall w)\\ & [[\neg P(x, y, u) \vee \neg P(y, z, v) \vee \neg P(u, z, w) \vee P(x, v, w)] \wedge\\ & \wedge [\neg P(x, y, u) \vee \neg P(y, z, v) \vee \neg P(x, v, w) \vee P(u, z, w)]],\end{aligned}$$

которая совпадает с ПНФ предложения σ_2 (почему?);

$$\begin{aligned}\sigma_3^*\colon\ & (\forall x)[P(x, a, x) \wedge P(a, x, x)];\\ \sigma_4^*\colon\ & (\forall x)[\neg P(x, b, x) \vee \neg P(b, x, x) \vee\\ & \qquad \vee (P(x, g(x), b) \wedge P(g(x), x, b))],\end{aligned}$$

где a и b — новые константы, а g — новая функция.

Другой возможный путь решения задачи — использовать язык логики предикатов $\mathcal{L} = \{=, f, i, e\}$, где f — символ функции от двух переменных с интерпретацией $f(x, y) = x \# y$, i — функция от одной переменной, причем $i(x)$ — обратный элемент к x, константа e — единичный элемент группы, а $=$ — предикат, обозначающий равенство, например, как в [Hami78].

2.12.23. Пусть S_1 и S_2 — два множества дизъюнктов:

$$\begin{aligned}S_1 &= \{\{P(a)\}, \{P(x), P(f(x))\}\};\\ S_2 &= \{\{P(x), Q(x)\}, \{R(z)\}, \{T(y), \neg W(y)\}\}.\end{aligned}$$

Определите эрбрановские множества H_3 для S_1 и $H_1, \ldots, H_{10}$ для S_2.

Р е ш е н и е. Для S_2. В S_2 не встречаются символы переменных. Введем константу c. Тогда $H_0 = \{c\}$. Более того, в S_2 нет функциональных символов. Таким образом, $H_0 = H_1 = \ldots = H_{10} = \{c\}$.

2.12.24. Определите эрбрановские множества H_0 и H_1 для множества дизъюнктов $S = \{\{P(f(x), a, g(f(x), b))\}\}$. Определите все основные примеры (по определению 2.4.3) для S на множествах H_0 и H_1.

Р е ш е н и е.

$$H_0 = \{a, b\},$$
$$H_1 = \{a, b, f(a), f(b), g(f(a), b), g(f(b), b)\}.$$

Тогда примеры на множестве H_0 для S суть

$$S_{11} = \{\{P(f(a), a, g(f(a), b))\}\},$$
$$S_{12} = \{\{P(f(b), a, g(f(b), b))\}\}.$$

Примеры на множестве H_1 для S суть

$$S_{21} = \{\{P(f(a), a, g(f(a), b))\}\},$$
$$S_{22} = \{\{P(f(b), a, g(f(b), b))\}\},$$
$$S_{25} = \{\{P(f(g(f(a), b)), a, g(f(g(f(a), b)), b))\}\},$$
$$S_{26} = \{\{P(f(g(f(b), b)), a, g(f(g(f(b), b)), b))\}\}.$$

2.12.25. Докажите при помощи замкнутых систематических таблиц общезначимость следующих предложений:

а) $[(\exists x)P(x) \to (\forall x)Q(x)] \to (\forall x)[P(x) \to Q(x)]$;

б) $\neg(\forall x)P(x) \leftrightarrow (\exists x)(\neg P(x))$;

в) $(\forall x)(P(x) \wedge Q(x)) \leftrightarrow ((\forall x)P(x) \wedge (\forall x)Q(x))$;

г) $(\exists x)(P(x) \vee Q(x)) \leftrightarrow (\exists x)P(x) \vee (\exists x)Q(x)$;

д) $(\exists x)(\forall y)P(x, y) \to (\forall y)(\exists x)P(x, y)$.

Р е ш е н и е. а) См. рис. 2.7 По определению 2.3.7, предложение а) доказуемо по Бету, то есть его можно доказать посредством ЗСТ.

в) См. рис. 2.8. Построенная таблица замкнута и противоречива, следовательно, она содержит доказательство предложения в).

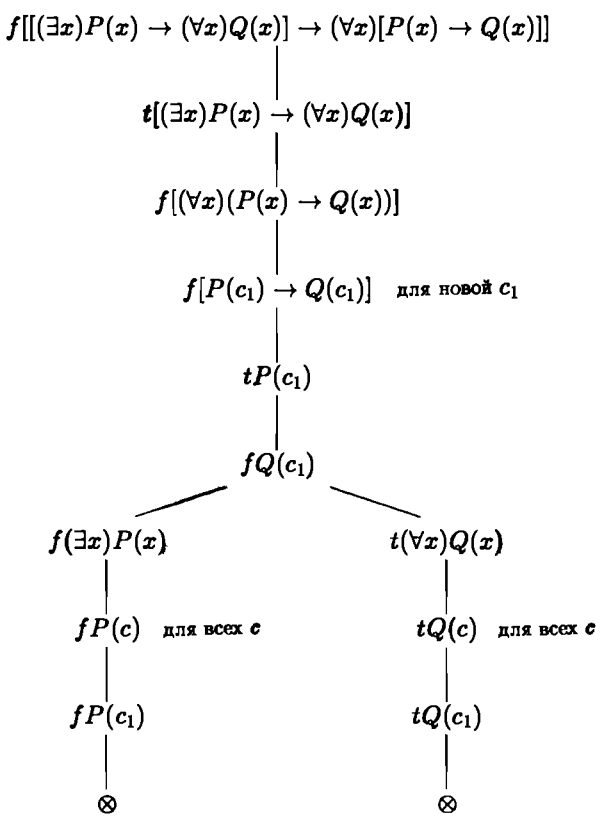


Рис. 2.7

2.12.26. Докажите средствами ЗСТ, что следующие предложения недоказуемы по Бету:

σ: $(\forall x)(P(x) \vee Q(x)) \to (\forall x)P(x) \vee (\forall x)Q(x)$;

φ: $(\exists x)P(x) \wedge (\exists x)Q(x) \to (\exists x)(P(x) \wedge Q(x))$.

Найдите две интерпретации $\mathcal{A}_\sigma$ и $\mathcal{A}_\varphi$, в которых соответственно σ и φ ложны.

Р е ш е н и е. Предположим, что σ доказуемо по Бету. Тогда ЗСТ для σ с корнем $f\sigma$ должна быть противоречива. Построим ЗСТ (см. рис. 2.9). Эта ЗСТ непротиворечива (почему?), поэтому σ не может быть доказуемо по Бету.

Чтобы определить интерпретацию $\mathcal{A}_\sigma$, отметим на непротиворечивых ветвях вершины tP и tQ знаком $*$ (почему?). Константы c_1 и c_2 — область искомой интерпретации. Пусть $\varepsilon(P) = \{c_1\}$ и $\varepsilon(Q) = \{c_2\}$. Тогда предложение σ ложно в $\mathcal{A}_\sigma = (\{c_1, c_2\}, \varepsilon(P), \varepsilon(Q))$

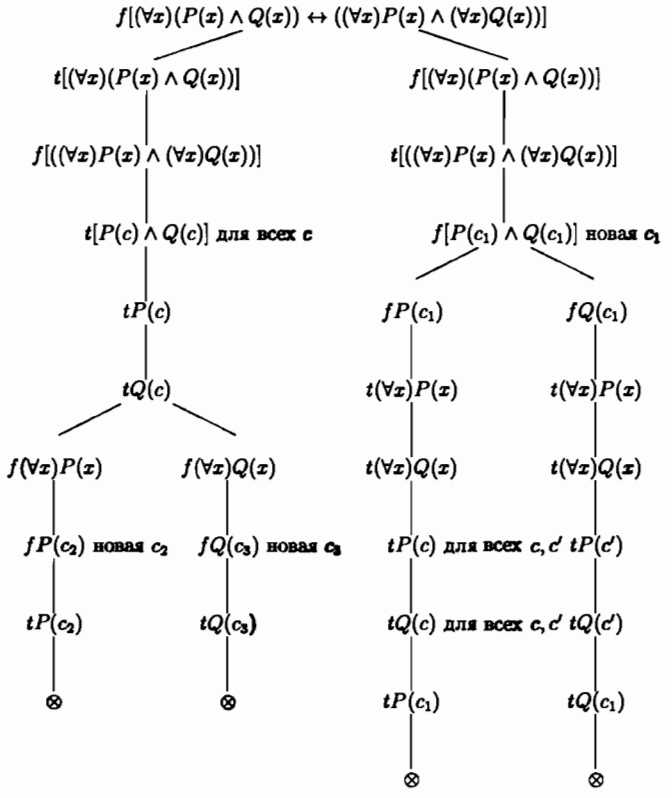


Рис. 2.8

(приведите полное доказательство последнего утверждения, опираясь на определение 2.5.5).

2.12.27. Найдите опровержения приведенных ниже множеств методом семантических деревьев:

а) $S = \{\{P(x)\}, \{\neg P(x), Q(x, a)\}, \{\neg Q(y, a)\}\}$;

б) $S' = \{\{P(x), Q(f(x))\}, \{\neg P(a), R(x, y)\}, \{\neg R(a, x)\}, \{\neg Q(f(a))\}\}$.

Решение.

а) $S = \{\{\underbrace{P(x)}_{1}\}, \{\underbrace{\neg P(x), Q(x, a)}_{2}\}, \{\underbrace{\neg Q(y, a)}_{3}\}\}$. Эрбранов универсум для S имеет вид $H = \{a\}$. Далее см. рис. 2.10.

2.12.28. Постройте семантические деревья и определите невыполнимые основные примеры, соответствующие следующим формулам:

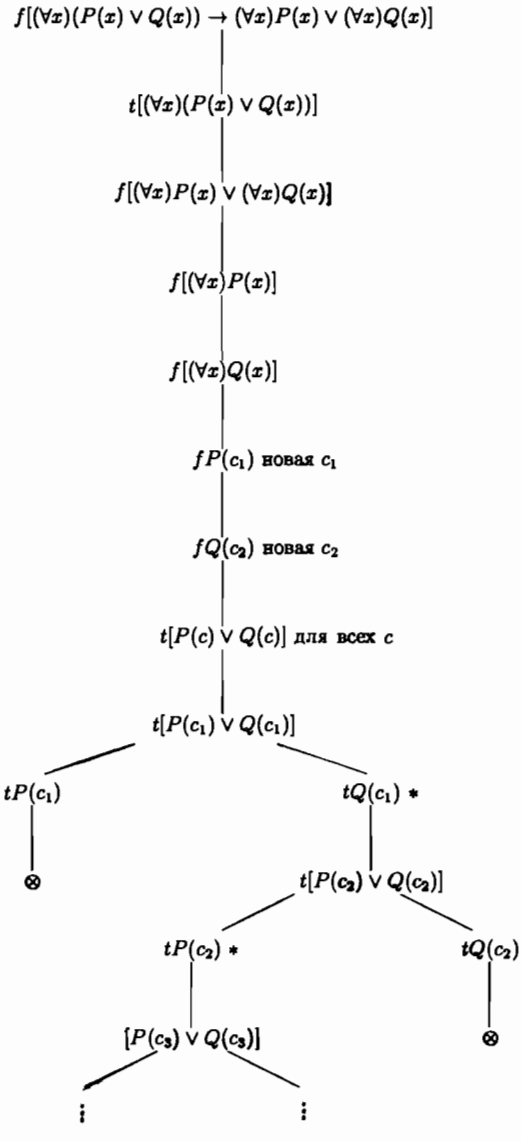


Рис 2.9

а) σ: $(\exists x)(\forall y)P(x, y) \wedge (\forall y)(\exists x)\neg P(y, x)$;

б) φ: $(\exists x)(\forall y)(\forall z)(\exists w)[P(x, y) \wedge \neg P(y, x)]$;

в) τ: $(\exists x)(\forall y)(\forall z)(\exists w)[P(x, y) \wedge \neg P(z, w)]$.

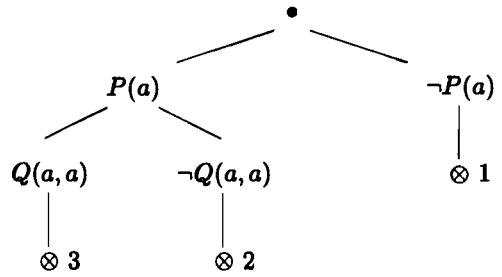


Рис. 2.10

Р е ш е н и е. а) Определим СНФ формулы σ:

$\sigma \leftrightarrow (\exists x)(\forall y)P(x, y) \wedge (\forall z)(\exists w)\neg P(z, w)$

(переименование связанных переменных)

$\leftrightarrow (\exists x)(\forall y)(\forall z)(\exists w)[P(x, y) \wedge \neg P(z, w)]$ (почему?).

Тогда

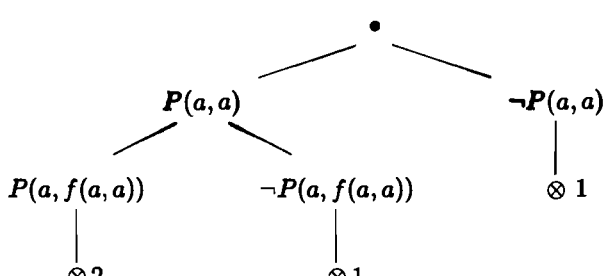
σ^*: $(\forall y)(\forall z)[P(a, y) \wedge \neg P(z, f(y, z))]$.

Эрбрановский универсум есть множество

$$H = \{a, f(a, a), f(a, f(a, a)), f(f(a, a), a), f(f(a, a), f(a, a)), \ldots\},$$

и соответствующее множество дизъюнктов —

$$S = \{\{P(a, y)\}, \{\neg P(z, f(y, z))\}\}.$$



Один из невыполнимых основных примеров формулы σ есть $\{\{P(a, f(a, a))\}, \{\neg P(a, f(a, a))\}\}$.

б) По теореме 2.3.6

$\varphi \leftrightarrow (\exists x)(\forall y)(\forall z)(\exists w)[P(x, y) \wedge \neg P(y, x)]$

$\leftrightarrow (\exists x)(\forall y)[P(x, y) \wedge \neg P(y, x)]$.

Тогда φ^*: $(\forall y)[P(a, y) \wedge \neg P(y, a)]$. Теоретико-множественная форма φ есть множество $\{\{P(a, y)\}, \{\neg P(y, a)\}\}$, и эрбрановский универсум $H = \{a\}$. Невыполнимый основной пример — $\{\{P(a, a)\}, \{\neg P(a, a)\}\}$.

2.12.29. Определите, является предложение

σ: $(\exists x)[G(x) \vee F(x)] \to [(\exists x)F(x) \to (\exists x)\neg G(x)]$

логически истинным или нет, и постройте соответствующее доказательство или контрпример.

Решение. Мы построим ЗСТ с корнем $f\sigma$. Если эта таблица противоречива, мы применим определение 2.6.7 и теорему 2.10.6. Если таблица непротиворечива, то непротиворечивая ветвь этой таблицы даст нам интерпретацию, в которой предложение σ ложно.

2.12.30. Найдите невыполнимые основные примеры для следующих множеств дизъюнктов:

а) $S_1 = \{\{P(x, a, g(x, b)\}, \{\neg P(f(y), z, g(f(a), b))\}\}$;

б) $S_2 = \{\{P(x)\}, \{\neg P(x), Q(f(x))\}, \{\neg Q(f(a))\}\}$;

в) $S_3 = \{\{P(x)\}, \{Q(x, f(x)), \neg P(x)\}, \{\neg Q(g(y), z)\}\}$.

Решение. Определите соответствующий эрбрановский универсум и постройте соответствующее семантическое дерево.

2.12.31. Создадим язык логики предикатов $\mathcal{L}$ для изучения евклидовой геометрии. Специальными символами $\mathcal{L}$ будут:

$P(x)$ — унарный предикат, интерпретируемый как «x является точкой»;

$E(y)$ — унарный предикат, интерпретируемый как «y является прямой линией»;

$I(x, y)$ — бинарный предикат, интерпретируемый, как «x принадлежит y».

а) Какова будет интерпретация $\mathcal{L}$? Опишите область этой интерпретации и интерпретации P, E и I: соответственно $\varepsilon(P)$, $\varepsilon(E)$ и $\varepsilon(I)$.

б) Для предложения σ: $(\forall x)[P(x) \to (\exists y)(E(y) \wedge I(x, y))]$ языка $\mathcal{L}$ сформулируйте его интуитивную интерпретацию и рассмотрите вопрос о его истинности.

в) Добавим к языку $\mathcal{L}$ бинарный предикат $\Pi(x, y)$ с интуитивной интерпретацией «прямые x и y параллельны». Опишите интерпретацию языка $\mathcal{L} \cup \{\Pi\}$. Сформулируйте аксиомы для $\mathcal{L} \cup \{\Pi\}$.

Решение. Основные аксиомы о точках и прямых таковы.

1) Существует по крайней мере одна точка, которая не принадлежит к заданной прямой.

2) Любая пара точек принадлежит ровно к одной прямой.

3) На каждой прямой находятся хотя бы две точки.

Из этих аксиом мы можем вывести, например, теорему: существуют по крайней мере две прямые, проходящие через заданную точку.

Параллельность прямых можно определить так: две прямые на плоскости, которые не имеют общих точек, называются параллельными (параллельность *stricto sensu*, в отличие от параллельности *lato sensu*, которая утверждает, что две прямые параллельны, если они совпадают или не имеют общих точек).

а) Интерпретация $\mathcal{L}$ имеет вид $\mathcal{A}=(A, \varepsilon(P), \varepsilon(E), \varepsilon(I))$, где

$A=\{x$: x точка на плоскости$\}\cup\{x$: x прямая линия на плоскости$\}$

$\varepsilon(P)=\{x$: x — точка$\}$,

$\varepsilon(E)=\{x$: x — прямая$\}$,

$\varepsilon(I)=\{(x, y)$: x — точка, y — прямая, и x лежит на $y\}$.

б) По определению 2.5.5

$$\mathcal{A}\models(\forall x)[P(x)\to(\exists y)(E(y)\wedge I(x, y))] \qquad (1)$$

$\Leftrightarrow$ для любой точки $c\in A$ $\mathcal{A}\models P(c)\to(\exists y)(E(y)\wedge I(c, y))$

$\Leftrightarrow$ для любой точки c и любой прямой c

$\mathcal{A}\models P(c)\to(\exists y)(E(y)\wedge I(c, y))$

$\Leftrightarrow$ для любой точки c $\mathcal{A}\models P(c)\to(\exists y)(E(y)\wedge I(c, y))$

и для любой прямой c $\mathcal{A}\models P(c)\to(\exists y)(E(y)\wedge I(c, y))$.

Но для любой прямой c верно $\mathcal{A}\models P(c)\to(\exists y)(E(y)\wedge I(c, y))$, как показано, например, в доказательстве следствия 1.5.4 и комментариях к этому доказательству.

Тогда по тавтологии $A\wedge truth\leftrightarrow A$ мы получим

(1) $\Leftrightarrow$ для любой точки c $\mathcal{A}\models P(c)\to(\exists y)(E(y)\wedge I(c, y))$

$\Leftrightarrow$ для любой точки c $\mathcal{A}\models(\exists y)(E(y)\wedge I(c, y))$ или $\mathcal{A}\not\models P(c)$

$\Leftrightarrow$ для любой точки c существует такая прямая c', что

$\mathcal{A}\models(E(c')\wedge I(c, c'))$ (по тавтологии $A\vee(B\wedge\neg B)\leftrightarrow A$).

Однако последнее предложение следует из аксиом.

Пусть c_1 — прямая из A. Тогда на c_1 лежат по крайней мере две точки c_2 и c_3. Пусть c' будет прямой, проходящей через c и, например, через c_2.

Следовательно, предложение σ истинно в $\mathcal{A}$.

в) Предикат $\Pi(x, y)$ может быть определен средствами языка $\mathcal{L}$. Формально, на языке $\mathcal{L}\cup\{\Pi\}$ аксиомы Евклида таковы:

1) $(\forall x)(\exists y)[E(x) \wedge P(y) \wedge \neg I(x, y)]$;

2) $(\forall x)(\forall y)(\exists z)\{P(x) \wedge P(y) \wedge E(z) \wedge \neg(x = y) \rightarrow$
$\rightarrow [I(x, z) \wedge I(y, z) \wedge ((\forall w)(E(w) \wedge I(x, w) \wedge I(y, w)))$
$\rightarrow (z = w)]\}$;

3) $(\forall x)(\exists y)(\exists z)[E(x) \wedge \neg(y = z) \wedge P(y) \wedge P(z) \rightarrow I(y, x) \wedge$
$\wedge I(z, x)]$

4) $\Pi(x, y) \leftrightarrow \neg(\exists z)[P(z) \wedge E(x) \wedge E(y) \wedge I(z, x) \wedge I(z, y)]$.

Последняя аксиома является определением предиката Π.

Теорема в начале этого решения имеет такой вид:

$$(\forall x)(\exists z)(\exists w)[P(x) \wedge E(z) \wedge E(w) \rightarrow I(x, y) \wedge I(x, w) \wedge \neg(z = w)].$$

Средствами предиката Π мы можем, например, сформулировать транзитивность и симметричность отношения параллельности:

$(\forall x)(\forall y)(\forall z)[\Pi(x, y) \wedge \Pi(y, z) \rightarrow \Pi(x, z)]$;

$(\forall x)(\forall y)[\Pi(x, y) \leftrightarrow \Pi(y, x)]$.

(Докажите, что последние две формулы истинны в интерпретации $\mathcal{A}'$ языка $\mathcal{L} \cup \{\Pi\}$, где $\mathcal{A}' = (A, \varepsilon(P), \varepsilon(E), \varepsilon(I), \varepsilon(\Pi))$ и $\varepsilon(\Pi) = \{(x, y)$: x, y — прямые, и x параллельна $y\}$).

2.12.32. Определите, какие из следующих множеств предложений унифицируемы. Если они унифицируемы, найдите наиболее общий унификатор (НОУ).

а) $S = \{\{P(f(a), g(x))\}, \{P(y, y)\}\}$;

б) $S = \{\{P(a, x, h(g(z)))\}, \{P(z, h(y), h(y))\}\}$;

в) $S = \{\{Q(f(w), a, z)\}, \{Q(w, b, f(z))\}\}$;

г) $S = \{\{\text{любит}(w, f(y))\}, \{\text{любит}(\text{Джордж}, \text{футбол})\}\}$;

д) $S = \{\{R(w, y), Q(w, f(z), z), \neg R(w, w)\}, \{R(w, z),$
$\neg Q(f(w), w, z)\}\}$;

е) $S = \{\{P(f(x), a)\}, \{P(y, f(w))\}\}$;

ж) $S = \{\{P(f(x), z)\}, \{P(y, a)\}\}$.

Р е ш е н и е. Применим алгоритм унификации.

а) $DS_1 = \{f(a), y\}$.

Введем $\theta_1 = \{y/f(a)\}$, $S_1 = \{\{P(f(a), g(x))\}, \{P(f(a), f(a)\}\}$, $DS_2 = \{g(x), f(a)\}$.

Мы видим, что в DS_2 не содержится элементарных вхождений переменных, так как x встречается только в $g(x)$. Следовательно, S не унифицируемо.

б) $DS_1 = \{a, z\}$.

Введем $\theta_1 = \{z/a\}$, $S_1 = \{\{P(a, x, h(g(a)))\}, \{P(a, h(y), h(y))\}\}$, $DS_2 = \{x, h(y)\}$.

Введем $\theta_2 = \{x/h(y)\}$,

$S_2 = \{\{P(a, h(y), h(g(a)))\}, \{P(a, h(y), h(y))\}\}$, $DS_3 = \{g(a), y\}$.

Введем $\theta_3 = \{y/g(a)\}$,

$S_2 = \{\{P(a, h(g(a)), h(g(a)))\}, \{P(a, h(g(a)), h(g(a)))\}\}$.

Следовательно, S унифицируемо, и его НОУ есть $\theta_1\theta_2\theta_3 = \{z/a, x/h(g(a)), y/g(a)\}$.

2.12.33. Определите, какие из следующих предложений унифицируемы. В случае, когда предложения унифицируемы, найдите НОУ.

а) $S = \{\{Q(a)\}, \{Q(b)\}\}$;

б) $S = \{\{Q(a, x)\}, \{Q(a, a)\}\}$;

в) $S = \{\{Q(a, x, f(x))\}, \{Q(a, y, y)\}\}$;

г) $S = \{\{Q(x, y, z)\}, \{Q(u, h(v, v), u)\}\}$;

д) $S = \{\{P(x, g(x), y, h(x, y), z, f(x, y, z))\},$

$\{P(u, v, e(v), w, k(v, w), s)\}\}$.

(Мы полагаем, что a, b — константы, а f, g, h, e, k — функции.)

2.12.34. Пусть θ — общий унификатор для множества дизъюнктов S. Докажите, что θ — НОУ для S и $\theta\theta = \theta$ тогда и только тогда, когда для любого унификатора ω для S справедливо $\omega = \theta\omega$.

Р е ш е н и е. Если θ — НОУ, то для любого унификатора ω найдется такая подстановка γ, что

$$\omega = \theta\gamma = (\theta\theta)\gamma = \theta(\theta\gamma) = \theta\omega.$$

Наоборот, если ω — унификатор и $\omega = \theta\omega$, то для некоторой подстановки γ справедливо

$$\theta\gamma = (\theta(\theta\gamma)) = \theta\theta\gamma. \qquad (1)$$

Предположим, что $\theta \neq \theta\theta$. Тогда найдется такая переменная x и такой терм q, что $x/q \in \theta$ и $x/q \notin \theta\theta$. Это означает, что $(x/q)\gamma \in \theta\gamma$ и $(x/q)\gamma \notin \theta\theta\gamma$, а это приводит к противоречию ввиду (1).

2.12.35. Определите невыполнимый основной пример для следующего множества дизъюнктов:

$S = \{\{P(x, a, g(x, b))\}, \{\neg P(f(y), z, g(f(a), b))\}\}$.

Р е ш е н и е. Заметим, что мы могли бы применить метод резолюции (и узнать, при каких условиях S невыполнимо), если бы переменная x совпадала с $f(y)$, переменная z — с a, и переменная x —

с $f(a)$. Используем подстановку $\theta = \{z/a, x/f(a), y/a\}$ Тогда S примет вид $S' = \{\{P(f(a), a, g(f(a), b))\}, \{\neg P(f(a), a, g(f(a), b))\}\}$. Очевидно, что S' — искомый невыполнимый основной пример (почему?).

2.12.36. Предположим, что верны утверждения, представленные ниже.

а) Существует хотя бы один дракон.

б) Дракон либо спит в своей пещере, либо охотится в лесу.

в) Если дракон голоден, он не может спать.

г) Если дракон устал, он не может охотиться.

Примените метод резолюции, чтобы ответить на следующие вопросы.

А) Что делает дракон, когда он голоден?

Б) Что делает дракон, когда он устал?

В) Что делает дракон, когда он голоден и устал?

Р е ш е н и е. Введем предикаты:

Дракон (x) — x — дракон;

Может (x, y, z) — y может делать x в z;

Делает (x, y, z) — y делает x в z;

Голоден (x) — x голоден;

Устал (x) — x устал.

Мы также предположим, что

$$\text{Может}(x, y, z) \leftarrow \text{Делает}(x, y, z).$$

Выразим утверждения а), б), в) и г) при помощи дизъюнктов:

а) Дракон$(A) \leftarrow$;

б) Делает(спит, A, пещера), Делает(охотится, A, лес) $\leftarrow$;

в) $\neg$Может(спит, A, пещера) $\leftarrow$ Голоден(A);

г) $\neg$Может(охотится, A, лес) $\leftarrow$ Устал(A).

Обратим предложения а)–г) и (*) в теоретико-множественную форму:

1) {Дракон(A)};

2) {Делает(спит, A, пещера), Делает(охотится, A, лес)};

3) {$\neg$Голоден(A), $\neg$Может(спит, A, пещера)};

4) {$\neg$Устал(A), $\neg$Может(охотится, A, лес)};

5) {Может(x, y, z), $\neg$Делает(x, y, z)}.

А) Добавим к предложениям 1–5 два предложения:

6) {Голоден(A)};

7) {¬Делает(x, y, z)}.

Предложение 7 является целью наших поисков. Попробуем вывести □ методом резолюции, соответствующим образом подбирая значения для x, y и z.

8) {Делает(спит, A, пещера)}, x = охотится, $y = A$, z = лес — из 2 и 7;

9) {Может(спит, A, пещера)} — из 5 и 8;

10) {¬Может(спит, A, пещера)} — из 3 и 6;

11) □ — из 9 и 10.

Следовательно, если дракон голоден, то он охотится в лесу.

Б) Добавим следующие два предложения к предложениям 1–5:

6) {Устал(A)};

7) {¬Делает(x, y, z)}.

Тогда можно вывести

8) {Делает(охотится, A, лес)}, x = спит, $y = A$, z = пещера — из 2 и 7;

9) {Может(охотится, A, лес)} — из 5 и 8;

10) {¬Может(охотится, A, лес)} — из 4 и 6;

11) □ — из 9 и 10.

В) Добавим к 1–5 такие предложения:

6) {Голоден(A)};

7) {Устал(A)};

8) {¬Делает(x, y, z)}.

Предложение 8 — цель резолютивного поиска. Имеем:

9) {¬Может(спит, A, пещера)} — из 3 и 6;

10) {¬Может(охотится, A, лес)} — из 4 и 7;

11) {¬Делает(спит, A, пещера)}, x = спит, $y = A$, z = пещера — из (5) и (9);

12) {¬Делает(охотится, A, лес)}, x = охотится, $y = A$, z = лес — из (5) и (10);

13) {Делает(охотится, A, лес)} — из (2) и (11);

14) {Может(охотится, A, лес)}, x = охотится, $y = A$, z = лес — из (5) и (13);

15) □ — из (10) и (14).

Заметим, что противоречивость предложений с 1 по 8 была доказана без помощи предложения 8, которое является нашей целью.

Соответственно условиям задачи, предположение, что дракон одновременно голоден и устал, приводит к противоречию. Таким образом, мы не можем узнать, что делает дракон, когда он и голоден, и устал.

2.12.37. Пусть следующие предикаты интерпретируются так:

$T(x, y, u, v)$ — «$xyuv$ — это трапеция, x, y, u и v — соответственно ее верхняя левая, верхняя правая, нижняя правая и нижняя левая вершины»;

$P(x, y, u, v)$ — «отрезки xy и vu параллельны»;

$E(x, y, z, u, v, w)$ — «$\widehat{xyz} = \widehat{uvw}$»;

и пусть даны предложения:

A_1: $(\forall x)(\forall y)(\forall u)(\forall v)[T(x, y, u, v) \to P(x, y, v, u)]$

(определение трапеции);

A_2: $(\forall x)(\forall y)(\forall u)(\forall v)[P(x, y, u, v) \to E(x, y, v, u, v, y)]$

(если $xy \| vu$, то $\widehat{xyv} = \widehat{uvy}$);

A_3: $T(a, b, c, d)$ ($abcd$ — трапеция).

Докажите методом резолюции, что из A_1, A_2 и A_3 следует $E(a, b, d, c, d, b)$.

Р е ш е н и е. Нам достаточно доказать, что предложение

$$A_1 \wedge A_2 \wedge A_3 \wedge \neg E(a, b, d, c, d, b) \qquad (*)$$

невыполнимо (почему?).

Теоретико-множественная форма $(*)$ такова:

$$S = \{\{\neg T(x, y, u, v), P(x, y, v, u)\}, \{\neg P(x, y, v, u), E(x, y, v, u, v, y)\}, \{T(a, b, c, d)\}, \{\neg E(a, b, d, c, d, b)\}\}.$$

Теперь мы видим, что

1) $\{\neg T(x, y, u, v), P(x, y, v, u)\}$;
2) $\{\neg P(x, y, v, u), E(x, y, v, u, v, y)\}$;
3) $\{T(a, b, c, d)\}$;
4) $\{\neg E(a, b, d, c, d, b)\}$;
5) $\{\neg P(a, b, d, c)\}$ — унификация и резолюция 2 и 4;
6) $\{\neg T(a, b, c, d)\}$ — унификация и резолюция 1 и 5;
7) $\square$ — из 3 и 6.

2.12.38. Докажите при помощи метода резолюции и ЗСТ:

а) $\{\{(\forall x)[A(x) \to (B(x) \wedge C(x))], (\exists x)[A(x) \wedge \wedge D(x)]\}\} \models (\exists x)[D(x) \wedge C(x)]$;

б) $\{(\exists x)[P(x) \wedge (\forall y)(T(x, y) \to Q(x, y))], (\forall x)[P(x) \to \to (\forall y)(W(y) \to \neg Q(x, y))]\} \models (\forall x)[T(x) \to \neg W(x)]$.

Доказательство. а) Построим ЗСТ с корнем $f[(\exists x)[D(x) \wedge C(x)]]$. Затем мы объединим конъюнкцией (в одну линию!) $t[(\forall x)[A(x) \to (B(x) \wedge C(x))]]$ и $t[(\exists x)[A(x) \wedge D(x)]]$.

Для метода резолюции, мы обратим в СНФ конъюнкцию

$$((\exists x)[A(x) \wedge D(x)]) \wedge ((\forall x)[A(x) \to (B(x) \wedge C(x))]) \wedge (\neg(\exists x)[D(x) \wedge C(x)]).$$

(пользуясь коммутативностью $\wedge$, мы в первую очередь берем формулу с квантором существования, чтобы ее СНФ была проще). Затем мы построим теоретико-множественную форму а).

Глава 3

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ: ПАРАДИГМА ПРОЛОГа

Τό ὑφεστηχός δεῖ τέλος ἐπιλογιζεσθαι χαί πάσαν τήν ἐνάργειαν, ἐφ ἠν τά δοξαζόμενα ἀνάγομεν εί δέ μή, πάντα ἀκρίσιας χαί ταραχῆς ἐσται μεστά

Надо осмыслить установленное здание и всякую очевидность, к которой мы возводим наши предположения. Иначе все будет полно путаницы и смятения.

Эпикур

§ 3.1. Пролог и логическое программирование

3.1.1. *Введение*

До сих пор мы занимались исследованием и анализом основных понятий логики высказываний и логики предикатов, которые играют ключевую роль при изучении логического программирования. Теперь мы представим *логическое программирование* как непосредственное приложение всех тех элементарных понятий и результатов математической логики, которые были затронуты в предыдущих главах, подчеркивая, по мере возможности, прямую взаимосвязь математической логики с языком программирования ПРОЛОГ.

Термин *язык программирования* не совсем точно отражает реальное положение вещей. ПРОЛОГ точно так же, как ФОРТРАН и ПАСКАЛЬ, является скорее не языком, а системой понятий и обозначений [DiSc90], при помощи которых формализуются и представляются данные и вычислительные процедуры. Так называемые «языки программирования» очень сильно отличаются от естественных языков по своей выразительной силе, области применения и интерпретации. Мы будем постоянно использовать словосочетание *язык программирования* как устоявшийся термин, но при этом всегда будем соотносить его с тем набором выражений, которые задействованы в каждой конкретной разновидности программирования.

Главное значение пропозициональной логики и логики предикатов состоит в том, что с их помощью определяются математические модели, позволяющие доказывать истинность или ложность одних утверждений путем построения вывода из множества других утверждений или предположений. Следуя этим принципам, мы уже имели дело с моделями, описывающими правило Modus Ponens, с табличными доказательствами и методом резолюций. Именно метод резолюций служит связующим звеном между математической логикой и логическим программированием и в то же время восполняет пробел между абстрактными математическими моделями и операциями языков программирования, такими как ПРОЛОГ.

В первой главе мы рассматривали резолюцию как процедуру доказательства в логике высказываний. Далее, во второй главе, ознакомившись с каноническими формами, сколемовскими формами, а также с понятиями подстановки и унификации, мы распространили резолютивную процедуру доказательства и на логику предикатов. Теперь мы собираемся описать язык программирования, средства которого позволили бы нам уточнить и представить в завершенном виде события окружающего мира, а также наш обыденный практический опыт, используя при этом элементы математики настолько широко, насколько это возможно.

Характерными чертами этого языка являются формализация заданной информации, а также интенсивное использование аппарата вывода во взаимосвязи с математической логикой.

Наши знания формально представляются хорновскими дизъюнктами, которые не только позволяют описывать в простых терминах события окружающего мира, но и легко преобразуются в сколемовскую стандартную форму. Аппарат вывода с подстановкой, унификацией и резолюцией, задействованный в нашем языке, предоставляет удобное алгоритмическое средство для обработки данных и извлечения заключений из множества хорновских дизъюнктов.

Нам известно, что в процессе резолютивного вывода проводится исследование исходных данных задачи и обеспечивается получение ее решения при условии, что это решение существует. Имея в виду эти обстоятельства, мы можем начать предварительное знакомство с ПРОЛОГом на уровне интуитивного понимания. Обратимся к следующему примеру.

Предположим, что мы располагаем некоторыми сведениями, представленными в виде хорновских дизъюнктов логики предикатов

p_1: Лицо(Марк) ←
p_2: Лицо(Аврелий) ←
p_3: Смертен($x1$) ← Лицо($x1$)
p_4: Жил_в_Помпеях(Аврелий) ←
p_5: Жил_в_Помпеях(Марк) ←
p_6: Родился(Аврелий,45) ←
p_7: Родился(Марк, 40) ←

p_8: Умер($x2$, 79) ← Жил_в_Помпеях($x2$)
p_9: Извержение(вулкан,79) ←
p_{10}: Мертв($x3$, $t2$) ← Смертен($x3$), Родился($x3$,$t1$), $t2-t1>150$
p_{11}: Не_жив($x4$, $t3$) ← Мертв($x4$, $t3$)
p_{12}: Мертв($x5$, $t4$) ← Не_жив($x5$, $t4$)
p_{13}: Мертв($x6$, $t6$) ← Не_жив($x6$, $t5$), $t6 > t5$
p_{14}: ← Не_жив(x, 1987)

Для соблюдения некоторых формальных требований ПРОЛОГа, о которых еще пойдет речь в дальнейшем, мы записываем индексы рядом с переменными или термами; так, например, мы записываем $x1$ вместо x_1.

Первые тринадцать хорновских дизъюнктов задают информацию, которую мы предоставляем программе. Последний дизъюнкт p_{14} — это запрос, с которым мы обращаемся к ней:

Кто не жив в 1987 году?

Соответствующая теоретико-множественная форма представления предложений $p_1, \ldots p_{14}$ такова:

(1) {Лицо(Марк)}
(2) {Лицо(Аврелий)}
(3) {Смертен($x1$), ¬Лицо($x1$)}
(4) {Жил_в_Помпеях(Аврелий)}
(5) {Жил_в_Помпеях(Марк)}
(6) {Родился(Аврелий,45)}
(7) {Родился(Марк, 40)}
(8) {Умер($x2$, 79), ¬Жил_в_Помпеях($x2$)}
(9) {Извержение(вулкан, 79)}
(10) {Мертв($x3$, $t2$), ¬Смертен($x3$), ¬Родился($x3$, $t1$), ¬($t2-t1 > 150$)}
(11) {Не_жив($x4$, $t3$), ¬Мертв($x4$, $t3$)}
(12) {Мертв($x5$, $t4$), ¬Не_жив($x5$, $t4$)}
(13) {Мертв($x6$, $t6$), ¬Умер($x6$, $t5$), ¬($t6 > t5$)}
(14) {¬Не_жив(x, 1987)}.

Эти же предложения, будучи записанными на ПРОЛОГе, примут следующий вид:

```
(1)   лицо(марк).
(2)   лицо(аврелий).
(3)   смертен (X1):- лицо(X1).
(4)   жил_в_Помпеях(аврелий).
(5)   жил_в_Помпеях(марк).
(6)   родился(аврелий,45).
(7)   родился(марк,40).
(8)   умер(X2,79) :- жил_в_Помпеях(X2).
(9)   извержение(вулкан,79).
(10)  мертв(X3,T2) :- смертен(X3), родился(X3,T1),
                      gt(T2-T1,150).
```

```
(11)  не_жив(X4,T3) :- мертв(X4,T3).
(12)  мертв(X5,T4) :- не_жив(X5,T4).
(13)  мертв(X6,T6) :- умер(X6,T5), gt(T6,T5).
(14)  ? не_жив(X,1987).
```

где `gt(X, Y)` обозначает $X > Y$. Следуя системе обозначений ПРОЛОГа, мы записывали константы прописными буквами, а переменные — заглавными. Знак пунктуации «.» в большинстве версий ПРОЛОГа обозначает конец очередного хорновского дизъюнкта.

ПРОЛОГ, подобно логике предикатов, пытается доказать целевое утверждение. Если целевое утверждение справедливо, то в полученном ответе ПРОЛОГ учитывает все частные случаи решения, т. е. он дает все возможные значения переменных, при которых целевое утверждение становится справедливым. Давайте посмотрим, как это происходит, и каким образом здесь применяются подстановки, унификация и резолюция.

Этап 1:

Наша исходная цель имеет вид

$$\text{Не\_жив}(X, 1987).$$

ПРОЛОГ просматривает данные в поисках хорновского дизъюнкта, заголовок которого может быть унифицирован с текущей целью. Подстановка $\theta_1 = \{x4/x,\ t3/1987\}$ унифицирует цель с заголовком формулы (11). ПРОЛОГ тут же пытается проверить тело формулы (11), объявляя текущими целями одну за другой все посылки этой формулы.

Резолюция формул (14) и (11) дает

(15) $\{\neg\text{Мертв}(x, 1987)\}$ **из (11) и (14)**

Этап 2:

Наша цель теперь имеет вид

$$\text{Мертв}(x, 1987).$$

ПРОЛОГ, следуя резолютивному выводу, пытается унифицировать эту цель с заголовком некоторого хорновского дизъюнкта, содержащегося в программе. Это достигается применением подстановки $\theta_2 = \{x3/x,\ t2/1987\}$ к формуле (10).

Этап 3:

Новое целевое утверждение для ПРОЛОГа представлено тройкой посылок из формулы (10), т. е. оно имеет вид

$$\text{Смертен}(x), \quad \text{Родился}(x, t1) \quad \text{и} \quad \text{gt}(1987 - t1, 150).$$

Эти подцели будут рассматриваться в том порядке, в котором они входят в состав правила (10). Согласно методу резолюций это утверждение представляется в виде правила

(16) $\{\neg\text{Смертен}(x),\ \neg\text{Родился}(x, t1),\ \neg\ \text{gt}(1987 - t1, 150)\}$

Этап 4:

ПРОЛОГ унифицирует атом «Смертен(x)» с заголовком предложения (3) при помощи подстановки $\theta_3 = \{x1/x\}$. И метод резолюций использовал бы ту же самую подстановку для унификации атома «Смертен(x)» и заголовка «Смертен($x1$)» правила (3).

Этап 5:

Наша очередная цель теперь

$$\text{Лицо}(x).$$

Она, в свою очередь, унифицируема посредством подстановки $\theta_4 = \{x1/\text{Марк}\}$с фактом, представленным формулой (1).

Воспользовавшись методом резолюций, мы бы получили

(17) $\{\neg\text{Лицо}(x), \neg\text{Родился}(x, t1), \neg\text{gt}(1987 - t1, 150)\}$ из (3) и (16)

(18) $\{\neg\text{Родился}(\text{Марк}, t1), \neg\text{gt}(1987 - t1, 150)\}$ из (1) и (17)

Этап 6:

ПРОЛОГ продолжает выполнять процедуру доказательства и пытается выполнить следующую цель из той тройки, которая была выработана на этапе 3, то есть

$$\text{Родился}(\text{Марк}, t1)$$

Унификация успешно применяется к правилу (7) с подстановкой $\theta_5 = \{t1/40\}$. Метод резолюций, соответственно, образовал бы формулу

(19) $\{\neg\text{gt}(1987 - 40, 150)\}$ из (7) и (18)

Этап 7:

Последней целью из нашей тройки, которую нужно выполнить, остается

$$\text{gt}(1987 - 40, 150)$$

и она успешно выполняется ПРОЛОГом, поскольку $1987 - 40 > 150$.

Этап 8:

Целевое утверждение, полученное на этапе 2, «$\neg$Мертв(x,1987)» было выполнено с означиванием x =Марк. Цель первого этапа выполнена, таким образом, с тем же означиванием. На этом этапе резолютивный метод выводит дизъюнкт

(20) $\square$ из (19)

Этап 9:

ПРОЛОГ не останавливается на достигнутом. Он продолжает вычисления, пытаясь унифицировать последнюю из выбранных целей другим способом, для того чтобы отыскать все правильные решения. Как только эта последняя цель будет выполнена всеми возможными способами и все правильные решения будут получены,

процедура повторится, уже применительно к непосредственно предшествующей ей цели. Если и эту цель удастся выполнить каким-либо другим способом, то процедура продолжит попытки выполнить оставшуюся «последнюю» цель, используя подстановки, которая были получены к этому моменту. Следуя этой стратегии и не имея альтернативных возможностей выполнения заключительной цели «gt(1987 – 40, 150)», ПРОЛОГ попытается выполнить непосредственно предшествующую цель «Родился(Марк, $t1$)», воспользовавшись другой подстановкой. Таким образом, рано или поздно, ему доведется унифицировать подцель «Смертен(x)» с фактом (2), посредством подстановки $\theta_4 = \{x1/\text{Аврелий}\}$. Далее вычисления последуют тем же путем, как и на этапе (6), однако, решением на этот раз будет

¬Не_жив(Аврелий,1987)

для означивания x =Аврелий.

Этот хитроумный прием ПРОЛОГа, позволяющий ему отыскать все возможные решения, называется *откатом*. Он будет детально разобран в соответствующем разделе.

3.1.2. *Логика и программирование*

Основное отличие логического программирования и языка ПРОЛОГ от традиционного программирования и алгоритмических языков типа ФОРТРАН, БЕЙСИК, КОБОЛ и ПАСКАЛЬ заключается в основополагающих принципах логического программирования, а именно, в организации и выполнении логической программы.

Программа состоит из двух компонентов — *логики* и *управления* [Kowa79, Lloy87]. Под «логикой» здесь подразумевается система основных понятий, касающихся того, ЧТО делает программа, а под «управлением» — свод синтаксических понятий, определяющих, КАК именно она это делает (например, как именно алгоритм решает задачу). Если бы мы захотели выразить это одним уравнением, то мы могли бы написать

ПРОГРАММА = ЛОГИКА + УПРАВЛЕНИЕ

Привычная нам программа, написанная на БЕЙСИКе или какомнибудь другом традиционном языке программирования, состоит из команд, описывающих действия, которые должна последовательно выполнить вычислительная машина для того, чтобы программа могла произвести желаемый результат. Например, команда программы на БЕЙСИКе

```
10 LET X = X + 5
```

увеличивает на 5 содержимое ячейки памяти, соответствующей переменной X.

Для языка программирования, подобного БЕЙСИКу, характерны *императивные* команды, которые *шаг за шагом* описывают поведе-

ние программ так, что после выполнения конечной последовательности таких команд получается правильный и ожидаемый результат[1]). Устройство этих команд в процессе проектирования программы, находится в ведении компоненты УПРАВЛЕНИЯ.

А ЛОГИКУ указанной команды составляет выражение «$X + 5$», которое само по себе является не командой, а всего лишь небольшой *дескриптивной* программой. Эта программа явно описывает арифметическое значение, которое должно быть вычислено, и неявно указывает, как именно оно должно быть вычислено.

Поэтому БЕЙСИК относится к числу *императивных* языков, содержащих *дескриптивные* элементы.

ПРОЛОГ-программа, напротив, может содержать предложение

милый(X) :– **любит**$(X$,**людей**$)$.

которое есть всего лишь логическое определение взаимосвязи предикатов «милый» и «любит».

Поэтому проектирование ПРОЛОГ-программы основано на правильном подборе предикатов, определяющих отношения между объектами, так, чтобы связь между входными данными и информацией, ожидаемой нами на выходе, была бы описана полностью.

Вообще говоря, программа на традиционном языке программирования задает функцию, отображающую входные данные в выходные данные программы, в то время как программа на логическом языка программирования задает *отношение между данными* [Watt90]. И поскольку отношения является более общим понятием, чем функция (всякая функция есть отношение, но отношение, в общем случае, не обязано быть функцией), логическое программирование наделено большими возможностями, нежели традиционное программирование.

Выбор предикатов, а значит и отношений, выражаемых этими предикатами, относится к логической компоненте ПРОЛОГа. А управление осуществляется путем выбора порядка расположения программных утверждений, а также при помощи ряда структурных средств управления, таких как отсечение «!» (см. п.3.4.4), используемых в качестве синтаксических конструкций ПРОЛОГа. По этой причине ПРОЛОГ относится к числу *дескриптивных* языков, содержащих *императивные* структурные элементы в качестве элементов управления.

Давайте рассмотрим пример программы, которая считывает два числа и выдает на печать наибольшее из них. Для того чтобы наглядно продемонстрировать различие между традиционным программированием и логическим программированием, мы приведем сначала программу на БЕЙСИКе, а затем ту же самую программу на ПРОЛОГе.

[1]) Точнее было бы сказать, *получается некоторый результат*, поскольку он может отличаться от ожидаемого, и корректность его еще предстоит обосновать. — *Прим. перев.*

Программа на БЕЙСИКе

```
10   INPUT "NUMBER1", X
20   INPUT "NUMBER2", Y
30   IF X  >  Y THEN 60
40   PRINT Y
50   GOTO 70
60   PRINT X
70   END
```

Программа на ПРОЛОГе

```
программа : – write("NUMBER1"), read(X), real(X), nl,
             write("NUMBER2"), read(Y), real(Y), nl,
             больше(X,Y,Z), write(Z).
больше(X,X,X).
больше(X,Y,Y) : – X < Y.
больше(X,Y,X) : – Y < X.
? программа.
```

Здесь **«real»** — специальный предикат ПРОЛОГа, который проверяет, является ли его аргумент действительным числом (см. п.3.5.4). Используя **«nl»**, мы считываем каждое число с новой строки.

Программа на Бейсике — это просто последовательность команд. Эти команды, исполняемые в порядке их нумерации в программе управляют вычислением, т. е. определяют структуру и ход исполнения программы. Логическая компонента в БЕЙСИК-программе присутствует в виде отношения **«<»**.

ПРОЛОГ-программа, в отличие от БЕЙСИК-программы, представляет собой совокупность дизъюнктов, которые полностью описывают отношение, определяющее большее из двух чисел, т. е. предикат **«больше»**. Это семейство дизъюнктов выражает логическое наполнение программы, которое преобладает в ПРОЛОГе, в то время как управление проявляет себя в порядке расположения предикатов в дизъюнктах и дизъюнктов в программе.

Типичная программа задается данными, которые мы хотим обрабатывать, и последовательностью действий, которые мы хотим выполнить. Как только мы описали исходные данные нашей задачи и процедуры, которые позволят нам получить результат, структура управления сразу определяет порядок, в котором должны применяться различные процедуры. Другими словами, мы имеем однозначно определенную последовательность процедур, некоторые из которых могут выполняться неоднократно (например, в цикле). Выбор подходящего формализма представляется весьма существенным, поскольку от него зависит автоматизация обработки данных. Поэтому одной из основных компонент типичной программы является *структура управления*, то есть, порядок, в соответствие с которым будут выполняться процедуры.

3.1.3. *Логическое программирование*

Один из наиболее существенных изъянов традиционного программирования проявляется в тех случая, когда возникает необходимость в постоянной модификации программ и адаптации их к новым требованиям. Ведь даже самое незначительное изменение классической программы обычно оказывает влияние на всю структуру управления. Поэтому проверка всего лишь слегка видоизмененной программы требует больших затрат и усилий.

Одно из решений сложившейся проблемы предоставляет логическое программирование на языке логики предикатов. Задача, которую необходимо решить с использованием компьютера, описывается совокупностью дизъюнктов логики предикатов. Некоторые из этих дизъюнктов, как, например, дизъюнкт **«больше(X,X,X)»** в рассмотренной ранее программе, выражают факты. Другие, как, например, дизъюнкт **«больше(X,Y,Y): - X < Y»**, выражают правила обработки фактов. А третьи выражают запросы, на которые нужно ответить, чтобы найти решение задачи. И в результате в логическом программировании вообще и в ПРОЛОГе в частности решением задачи, вследствие применения правил программы, будет либо ответ **«да»** или **«нет»** на запрос, либо *множество значений*, которые и представляют искомое решение.

Вот поэтому в логическом программировании распространены не только команды чисто функциональной природы, как в классическом программировании, но и предикаты, конструкции логики предикатов, которые могут принимать значения истины и лжи в зависимости от их интерпретации и значения их аргументов. Истинность или ложность того или иного предиката в соответствие с конкретными значениями его аргументов и определяет, какой из ответов **«да»** или **«нет»** будет выдан программой на поставленный вопрос.

Ответы на запросы к программе зависят ИСКЛЮЧИТЕЛЬНО от той информации, которой мы снабдим программу. Поэтому, если мы спросим программу из примера п. 3.1.1, является ли число 4 полным квадратом, т. е. квадратом некоторого целого числа, то программа, не имея никаких предикатов, характеризующих полные квадраты, ответит **«нет»**. Это означает, что цель

```
? полный_квадрат(4)
```

терпит неудачу, см. Замечание 1.9.9. На это следует обращать особое внимание. Неудача, постигшая ПРОЛОГ при попытке проверить целевое утверждение, вовсе не означает, что это утверждение и в самом деле неверно. Она лишь свидетельствует о том, что используя факты программы и аппарат вывода, имеющийся в нашем распоряжении, мы не можем заключить, что предъявленное целевое утверждение истинно. Этот эффект носит название *допущение замкнутости мира* (см. также п.3.6.1) [Watt90].

Программа считает предикат Q логики предикатов ложным, если она не может доказать, что Q является истинным

На основании этого мы можем рассматривать каждую ПРОЛОГ-программу как завершенное описание некоторого мироздания, которое полностью охарактеризовано данными программы. Все, что есть существенное в этом мире, описано данными, и соответствующие этим данным цели успешно достигаются; а все, что не описано в программе, считается неизвестным, и относящиеся к этому запросы терпят неудачу.

3.1.4. *История развития*

После того, как Эрбран в 1930 г. предложил алгоритм, позволяющий нам подобрать интерпретацию, опровергающую формулу φ в случае, если φ не является логически истинной, Гилмор впервые в 1960 г. попытался реализовать эрбрановский метод на компьютере [ChLi73]. В своей реализация этого метода Гилмор основывался на том, что дизъюнкт логически непротиворечив (см. Определение 2.5.19) тогда и только тогда, когда его отрицание невыполнимо в некоторой интерпретации. В соответствие с этим он завел в своей программе процедуру, которая формировала предложения логики предикатов и время от времени проверяла их на противоречивость (см. Определение 2.5.16). Программа Гилмора была, однако, не способна анализировать слишком сложные формулы логики предикатов.

После того, как Робинсон [Robi65] опубликовал свой алгоритм унификации, Лавленд в 1970 г. впервые предложил линейную резолюцию с правилом выбора. В этом варианте резолютивного метода мы выбираем дизъюнкт, строим для него резольвенту с другим дизъюнктом, затем применяем резолюцию к полученной резольвенте, используя какой-либо третий дизъюнкт, и продолжаем этот процесс, применяя правило резолюции к последней полученной резольвенте, до тех пор, пока не будет построен пустой дизъюнкт [ChLe73, Lloy87].

Считается, что первыми, кто вступил в область логического программирования (1972), были Ковальский и Колмероэ [StSh86]. Ковальскому впервые удалось описать процедурную интерпретацию хорновских логических дизъюнктов и показать, что правило вида

$$A :- B_1,\ B_2,\ \ldots,\ B_n$$

может быть воспринято и реализовано на языке рекурсивного программирования.

В это же самое время Колмероэ и его исследовательская группа разработали при помощи системы программирования ФОРТРАН язык программирования, предназначенный для доказательства теорем и реализующий процедурную интерпретацию Ковальского. Так

было заложено основание ПРОЛОГа (ПРОграммирование посредством ЛОГики).

Первый интерпретатор ПРОЛОГа, т. е. программа, которая переводит текст, понимаемый людьми (исходный файл), на язык, воспринимаемый машиной (машинные коды), был написан на АЛГОЛе в Марселе в 1972 г. Русселем, который основывался на теоретических разработках Колмероэ.

Однако в целом на вопрос о том, может ли логика, и, следовательно, математика быть использованы в качестве языка программирования, мы должны дать отрицательный ответ [Watt90]

Задача, выраженная на языке программирования, должна быть решена компьютером, тогда как в логике и математике имеются проблемы, для которых нет алгоритмического решения, и которые, таким образом, не могут быть решены компьютером.

Одним из примеров такой проблемы является проблема разрешимости логики предикатов, см. § 2.11.

По этой причине логическое программирование, которое мы будем понимать как концепцию программирования, базирующуюся на логике, всегда ограничивается лишь частью логических формул. ПРОЛОГ, например, имеет дело только с хорновскими дизъюнктами, одним из подмножеств множества предложений логики предикатов[1]).

Многие исследователи в последние годы сосредоточили свое внимание на поиске взаимосвязи между различными классами языков программирования, такими как классы языков функционального программирование, т. е. программирования, основанного на λ-исчислении [Thay88], логического программирования и объектно-ориентированного программирования, т. е. программирования, основанного на теории типов, согласно которой каждый объект представляется набором значений, свойств и процедур [Thay88]. Немало усилий было положено на то, чтобы найти подходящие преобразования и эквивалентности, позволяющие переходить от одного языка программирования к другому или применять различные формализмы в рамках одной программы.

Большинство языков логического программирования, которые по существу есть ни что иное, как системы доказательства логических теорем, использующие метод резолюций, рассматриваются нами как первые шаги в направлении оптимального логического программирования.

---

[1]) Хорновские дизъюнкты были выбраны в первую очередь потому, что резолютивные доказательства в этом классе формул всегда конструктивны, т. е. помимо существования контрмодели для них еще устанавливаются и значения переменных, на которых опровергается исследуемая формула. Это обстоятельство является наиболее существенным, если мы хотим применять логическое программирование для *вычислений*, а не только для для *доказательств*. — *Прим. перев.*

Для оптимального программирования, однако, мы должны решить проблему управления, то есть

а) сделать доступными более приемлемые и более хитроумные процедуры управления в каждом языке логического программирования;

б) найти возможность переложить все управление проектированием и выполнением программы на компьютер.

Решение проблемы управления позволит будущим программистам и пользователям ограничить свое взаимодействие с компьютером разработкой полных и строгих формулировок задачи. Программа, в свою очередь, возьмет на себя решение задачи и все управление ходом вычисления программы.

§ 3.2. Структура программы

3.2.1. *Элементы программы*

ПРОЛОГ-программа состоит из данных и запросов, на которые она должна дать ответ. Данные и запросы являются хорновскими дизъюнктами согласно определению 2.4.4.

Данные представляются одной из следующих форм:

$$A(c_1, \ldots, c_k) \leftarrow \qquad (*)$$

и символически записываются на ПРОЛОГе

$$A(c_1, \ldots, c_k).$$

Здесь A — предикат, а $c_1, \ldots, c_k$ — константы логики предикатов[1]). Знак пунктуации «.» указывает место окончания заданной формулы.

$$A(a_1, \ldots, a_k) \leftarrow B_1(b_1, \ldots, b_l), \ldots B_j(d_1, \ldots, d_m) \qquad (**)$$

и символически записываются на ПРОЛОГе

$$A(a_1, \ldots, a_k) :- B_1(b_1, \ldots, b_l), \ldots, B_j(d_1, \ldots, d_m),$$

где $A, B_1, \ldots, B_j$ — предикаты, а $a_1, \ldots, a_k, b_1, \ldots, b_l, d_1, \ldots, d_m$ — термы логики предикатов (см. Определение 2.2.1).

Данные вида (*) согласно определению 2.4.6 называются *фактами* программы.

Данные вида (**) называются *правилами* программы. При помощи правил программы выражаются отношения между предикатами, встречающимися среди фактов. Правила, являющиеся импликативными формулами логики предикатов, по отношению к которым применяется унификация и резолюция, образуют процедурную часть

[1]) Вообще говоря, аргументы $c_1, \ldots, c_k$ могут быть и произвольными термами, содержащими переменные и функциональные символы — *Прим. перев.*

программы. При этом $A(a_1, \ldots, a_k)$ называется *заголовком* правила, а $B_1(b_1, \ldots, b_l), \ldots, B_j(d_1, \ldots, d_m)$ — *хвостом*, или *телом* правила.

Факты и правила образуют *базу данных* программы [Brat90].

Запросы, или *целевые утверждения*, (см. Определение 2.4.5) представляют собой хорновские дизъюнкты вида

$$B_1(b_1, \ldots, b_l), \ \ldots, B_j(d_1, \ldots, d_m)$$

и символически записываются в ПРОЛОГе

$$? \ B_1(b_1, \ldots, b_l), \ \ldots, B_j(d_1, \ldots, d_m).$$

Здесь $B_1, \ldots, B_j$ — предикаты, а $b_1, \ldots, b_l, d_1, \ldots, d_m$ — термы логики предикатов (см. Определение 2.2.1).

Разделитель «.» следующий за каждым фактом, правилом или запросом обозначает конец соответствующего хорновского дизъюнкта.

Давайте обратимся к ПРОЛОГ-программе из примера 2.4.9 с запросом, который рассматривался в примере 2.10.12, — «Что может похитить Питер?»

```
вор(питер).
любит(мэри,шоколад).
любит(мэри,вино).
любит(питер,деньги).
любит(питер,X) :- любит(X,вино).
может_похитить(X,Y) :- вор(X), любит(X,Y).
? может_похитить(петр,Y).
```

Первые четыре дизъюнкта являются фактами, последующие два дизъюнкта — правилами, а заключительный дизъюнкт — это целевое утверждение (запрос) программы. ПРОЛОГ выдаст ответ

```
Y = деньги
Y = мэри
```

означающий, что Петр может похитить деньги или Мэри, о чем мы были уже осведомлены, изучив пример 2.10.12.

Итак, ПРОЛОГ-программа есть ни что иное, как собрание фактов, правил и запросов, лишенное каких-либо указаний и ориентиров для управления вычислением. Такое устройство программы свидетельствует о чрезвычайной гибкости ПРОЛОГа: мы можем модифицировать и расширять программу путем добавления и изъятия фактов и правил, применяя команды «assert» и «retract», о которых пойдет речь в п. 3.5.1.

Факты, фигурирующие в ПРОЛОГ-программе, можно воспринимать как *декларации* о предметном мире программы; это беспрекословные заявления, описывающие то или иное положение вещей. А вот правила [Xant90] можно интерпретировать не только как декларации, но и как процедуры программы. Классические языки программирования допускают по существу лишь процедурную интерпретацию, тогда как ПРОЛОГ — вероятно, единственный язык программи-

рования [Xant90, Brat90, ClMe84], который допускает как процедурную, так и декларативную интерпретацию[1]).

3.2.2. *Факты*

Как было отмечено в п. 3.2.1, факты — это отношения, т. е. предикаты, между разными объектами, представленными термами языка логики предикатов. Таким образом, факт — это предложение языка логики предикатов (см. Определение 2.2.17). Факты ПРОЛОГа могут быть напрямую соотнесены с констатацией фактов на разговорном языке.

Разговорный язык	*ПРОЛОГ*
канарейка — это птица	`птица(канарейка).`
Джон — это человек	`человек(джон).`
Петр любит Мэри	`любит(петр,мэри).`
Петр может похитить Мэри	`может_похитить(петр,мэри).`
Джон и Анна — родители Мэри	`родители(мэри,джон,анна).`

Наименования предикатов и констант всегда начинаются с прописной буквы, и слова в сложных словосочетаниях, обозначающих предикаты, соединяются символом подчеркивания «_». В конце записи каждого факта ставится точка.

Расположение термов в предикате упорядочено. Факт

```
любит(мэри,энди).
```

означает, что

«Мэри любит Энди»

и отличается от факта

```
любит(энди,мэри).
```

Выбор предикатов, а также наделение их названиями, которые будут далее использоваться — это прерогатива программиста. Поэтому важную роль играет истолкование фактов. Например, факт

---

[1]) Это не совсем верно, поскольку языки функционального программирования (ЛИСП, ХОУП, МИРАНДА и др.) также имеют декларативную семантику. В отличие от ПРОЛОГа, декларативная семантика языков функционального программирования имеет алгебраическую природу. Программы, написанные на этих языках, можно воспринимать как системы функциональных уравнений, поиск решения которых обеспечивается процедурной семантикой. Элементами декларативной семантики можно наделить и операторные языки программирования. Но здесь декларативная семантика играет второстепенную, вспомогательную роль и применяется главным образом при анализе операторных программ. Действительно замечательным свойством логического программирования является то, что обе семантики — декларативная и процедурная — равноправны и в значительной мере независимы, хотя и полностью соответствуют друг другу (см. теоремы о корректности и полноте). Поэтому на одну и ту же логическую программу человек-программист и компьютер-исполнитель могут смотреть с различных позиций, наилучшим образом соответствующих стилю мышления каждого из участников процесса программирования. — *Прим. перев.*

«отец (Джон,Крис)» можно понимать двояко: «Джон — отец Криса» или «Крис — отец Джона». Всякий раз мы должны остановиться на каком-либо одном варианте, и ни в коем случае не пытаться использовать оба сразу! Нам не приходится рассчитывать на получение правильного результата программой, содержащей факты

```
отец(джон,крис).
отец(джордж,ник).
```

если первый факт понимается нами как

«Крис — отец Джона»,

а второй

«Джордж — отец Ника»

3.2.3. *Правила*

Правилом ПРОЛОГа является импликативная формула (см. п. 3.2.1) логики предикатов. При помощи правил выражается информация и отношения более общие и более сложные, нежели простые сведения, представимые фактами. Например,

```
отец(крис,джордж).
отец(ник,джон).
отец(X,Y) :- сын(Y,X).
```

Первые два дизъюнкта — факты. Они указывают нам на взаимосвязь между конкретными людьми, Крисом и Джорджем, Ником и Джоном, соответственно. Третий дизъюнкт выражает общее безличное правило, в котором задействованы предикаты над данными: если Y — сын X, то X — отец Y.

Переменные, как в правилах, так и в запросах, пишутся с заглавной буквы. В конце каждого правила ставится точка. Заголовок отделяется от тела символьной комбинацией «: —». Если тело правила содержит более одного предиката, то эти предикаты отделяются друг от друга запятой «,».

Рассмотрим пример

```
любит(джон,мороженое).
любит(джон,мэри).
любит(джон,музыку).
съедобно(мороженое).
ест(X,Y) :- съедобно(Y), любит(X,Y).
```

Здесь тело правило образуют два предиката

«съедобно(Y)» и «любит(X,Y)»

Это правило утверждает, что для любых X и Y, если Y съедобно, и X любит Y, то X поедает Y (см. Определение 2.4.1).

В ПРОЛОГе два и более правил могут иметь один и тот же заголовок, и тогда мы имеем право записать их как одно новое правило

с одним общим заголовком, разделяя тела символом «;». Например правила

```
любит(X,Y) :- любит(Y,X).
любит(X,Y) :- хороший(Y).
```

можно объединить в одно правило

```
любит(X,Y) :- любит(Y,X); хороший(Y).
```

которое будет означать «X любит Y, если Y любит X, или если Y — хороший».

ПРОЛОГ нумерует факты и правила в каждой базе данных. Например, в базе данных из последнего примера «**любит(джон,мороженое)**» является первым дизъюнктом базы данных, а правило **ест**(X,Y) : – **съедобно**(Y), **любит**(X,Y) является пятым дизъюнктом. Как мы увидим в п. 3.4.1, порядок, в котором факты и правила заносятся в базу данных очень важен для вывода заключений.

3.2.4. *Запросы*

Запросы, или *целевые утверждения* ПРОЛОГа, — это вопросы обращенные к тем условиям и предикатам логики предикатов, которые фигурируют среди данных программы. Например, для данных п. 3.2.3 такими вопросами могут быть

```
? любит(джон,X).
? любит(X,Y), съедобно(Y).
```

и ПРОЛОГ ответит на первый запрос

```
X = мороженое
X = мэри
X = музыку
```

а на второй запрос —

```
X = джон
Y = мороженое
```

Запросы начинаются с вопросительного знака и оканчиваются точкой. На некоторые из запросов, как например, на запрос

```
? любит(джон,X).
```

можно ответить многими разными способами. *Составной запрос*, наподобие

```
? любит(X,Y), съедобно(Y).
```

может содержать более одной подцели, которые отделяются друг от друга запятой. В составных запросах, одни и те же переменные обычно относятся к одним и тем же термам. Например, спрашивая

```
? любит(X,Y), съедобно(Y).
```

мы просим программу отыскать такие X и Y, чтобы X любил Y, и Y было съедобно. А запрос

```
? любит(X,Y), съедобно(X).
```

потерпит неудачу. Это означает, что ПРОЛОГу не **удастся** отыскать подходящее значение X, и поэтому он вынужден будет ответить **нет**.

Взгляните на описание одной вечеринки.

```
любит(джес,элен).
любит(джон,мэри).
любит(джейн,энди).
любит(кейт,энди).
слушает(джон,рок).
слушает(тим,рег-тайм).
слушает(анна,джаз).
пьет(джес,пиво).
пьет(джон,джин).
пьет(джейн,вино).
пьет(кейт,вино).
пьет(анна,пепси).
курит(джес).
курит(кейт).
дружит(анна,X) :- слушает(X,рок).
дружит(энди,X) :- пьет(X,вино),любит(X,энди),
                  курит(X).
(1)  ? любит(джон,мэри).
(2)  ? любит(джек,мэри).
(3)  ? пьет(джейн,вино), слушает(джейн,рок).
(4)  ? дружит(энди,X).
(5)  ? слушает(X,Y).
(6)  ? любит(кейт,X), любит(джейн,X).
(7)  ? любит(X,Y), слушает(X,рок), пьет(Y,пиво).
```

ПРОЛОГ ответит следующим образом

```
(1)  да
(2)  нет
(3)  нет
(4)  X = кейт
(5)  X = джон
     Y = рок
     X = тим
     Y = рег-тайм
     X = анна
     Y = джаз
(6)  X = энди
(7)  нет
```

Запросы (1), (2) и (3) относятся к разряду *запросов проверки данных*, тогда как запросы с (4) по (7) — к разряду *запросов обработки данных*. Запросы (3), (6) и (7) являются *составными*, или *конъюнктивными запросами*.

Запросы (2), (3) и (7) терпят неудачу (Замечание 1.9.9 (2)) по следующим причинам. Для положительного ответа на запрос (2) среди данных программы нет информации, касающейся отношений Джека и Мэри. При попытке найти ответ на запрос (3) первая подцель выполняется успешно, т. к. Джейн любит вино, но зато отсутствуют сведения о том, какого рода музыку предпочитает слушать Джейн. Поэтому вторая подцель и, следовательно (почему?), весь запрос (3) терпит неудачу. Запрос (7) не будет выполнен поскольку информация о вечеринке не содержит сведений о таких значениях X и Y, которые удовлетворяли бы одновременно всем трем подцелям.

§ 3.3. Синтаксис данных

3.3.1. *Объекты ПРОЛОГа*

Как мы уже говорили, в ПРОЛОГе используется лишь собственное подмножество языка логики предикатов, обогащенное некоторыми вспомогательными символами. Таким образом термами в ПРОЛОГе являются *переменные*, *константы* и *функции*, зависящие от переменных и констант (см. Определение 2.2.1). Константы, в свою очередь, подразделяются на *атомы*, *числа* и *последовательности специальных символов*. Основными объектами языка ПРОЛОГ являются предикаты и списки. В этом разделе мы займемся изучением термов и основных объектов ПРОЛОГа.

3.3.2. *Алфавит ПРОЛОГа*

Алфавит ПРОЛОГа состоит из

а) заглавных и прописных букв латинского алфавита[1])
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z

б) цифр 0 1 2 3 4 5 6 7 8 9,

в) специальных символов, наподобие ! # $ % & * ( ) _ = ^ + < > ? / ., набор которых зависит от конкретной версии ПРОЛОГа,

г) вспомогательных символов, которые играют важную функциональную роль, но не отображаются на экран, наподобие пробела (SPACE) или перехода к новой строке (ENTER).

[1]) Не умаляя корректности и общности, мы позволили себе добавить к этому списку и буквы русского алфавита — *Прим. перев.*

3.3.3. *Переменные*

Переменные в ПРОЛОГе обозначаются символами или словами, начинающимися с заглавных букв алфавита ПРОЛОГа. Переменные всегда начинаются с заглавной буквы или со знака подчеркивания. Например, переменными являются

X, *Y*, Result, Отец, Who, _мэри, *A*3

Напротив, выражения

x, *y*, result, отец

переменными НЕ являются. Нельзя считать переменными и такие выражения

1-ый_атлет	(начинается цифрой)
"Result"	(заключено в кавычки)
3579	(цифровая запись)

Поэтому, если мы запишем

```
(1)   женщина(Мать).
(2)   женщина(мать).
```

то факт (1) будет означать для ПРОЛОГа, что всякая константа, имеющаяся в программе есть женщина (почему?), тогда как факт (2) говорит о том, что одна выделенная константа «мать» является женщиной.

ПРОЛОГ предоставляет дополнительную возможность использования *анонимных*, безымянных переменных в тех случаях, когда нам нет необходимости знать их точное имя; мы можем просто поставить на их место символ подчеркивания _ . Например, записывая в ПРОЛОГе

```
мать(элен,_).
```

мы сообщаем тем самым, что Элен — это мать некоего лица, имя которого в этой программе несущественно. Задавая вопрос

```
? родители(ник,_,Мать).
```

мы интересуемся тем самым только именем матери Ника, обозначая его отца анонимной переменной. Как правило, мы используем анонимную переменную в тех случаях, когда эта выделенная переменная встречается в программе лишь один раз [Xant90]. Как будет показано в соответствующем параграфе, применяя анонимные переменные, мы облегчаем процесс унификации и экономим время.

3.3.4. *Константы*

Константы, которые в ПРОЛОГе изображаются символьными строками, подразделяются на три класса в соответствие с классификацией [Brat90]. Согласно этой классификации константы в ПРОЛОГе могут быть

а) *атомарными термами*,
б) *числами*,
в) *последовательностями специальных символов*.

Атомарный терм — это последовательность букв, цифр и знаков «_» (которая, однако, не должна начинаться символом подчеркивания, иначе это будет переменная). Первый символ должен быть при этом строчной буквой. Например,

*y*342, tom, красное_яблоко

А вот такие выражения НЕ являются атомарными термами

Питер	(начинается заглавной буквой)
_*x*5	(начинается подчеркиванием)
3t	(начинается цифрой)
a.32	(содержит точку)

Всякая последовательность символов, заключенная в одинарные кавычки, также является атомом, как то

'x', 'Петр', '5tx', 'У Джона красивый галстук'

Числа бывают двух типов — целые и действительные. Например,

211345, −32, 0.000013, −5.7

Первоначально язык ПРОЛОГ разрабатывался для описания символьных преобразований, и его возможности выполнять арифметические вычисления были очень ограничены. Однако последняя редакция ПРОЛОГа [Colm90] предоставляет большие возможности для проведения математических вычислений.

Используя в ПРОЛОГе *последовательности специальных символов*, мы можем ввести в обиход символьные конструкции, которым придается особое истолкование. Например,

- - - > или = = = >

можно использовать для обозначения импликации →.

3.3.5. *Предикаты*

Предикаты (см. Определение 2.2.1) выражают отношения и свойства термов. Например, предикат

```
больше(+(X, 2), · (Y, 3))
```

означает, что терм $+(X, 2)$, или то же самое $X+2$, больше, чем терм $:(Y, 3)$, или то же самое $Y:3$ (см. Определение 2.2.1). Символы «+» и «:» соответствуют сложению и делению, X и Y — переменные, а 2 и 3 — константы. Наименования предикатов начинаются в ПРОЛОГе с прописной буквы.

В ПРОЛОГе имеется возможность строить *составные предикаты*, или *структуры*, из предикатов, которые уже имеются в программе. Возьмем, например, факты, описывающие состав семьи

```
отец(ник).
мать(мэри).
ребенок(анна).                    (*)
ребенок(тим).
```

В этом примере «отец», «мать» и «ребенок» — предикаты, а «ник», «мэри», «анна» и «тим» — константы.

Факты этой базы данных можно выразить, используя один предикат

```
семья(отец(ник), мать(мэри), дети(анна,тим)).   (**)
```

В этом выражении «семья» — предикат, но «отец», «мать», «дети» — уже не предикаты, а функции [Thay88]. Это означает, что выражения «отец(ник)», «мать(мэри)» и «дети(анна,тим)» рассматриваются как определенные значения функций, т. е. как константы. Поэтому «отец(ник)» в (*) — совсем другой синтаксический объект, нежели «отец(ник)» в (**).

Точно так же факт

```
любит(джордж,мороженое).
```

где «любит» — предикат а «джордж» и «мороженое» — константы, может быть выражен при помощи предиката

```
мороженое(любит(джордж)).   (1)
```

или

```
джордж(любит(мороженое)).   (2)
```

причем в (1) «мороженое», «любит», «джордж» — соответственно, предикат, функция и константа, тогда как в (2) те же самые выражения «джордж», «любит», «мороженое» будут предикатом, функцией и константой соответственно.

ПРОЛОГ распознает, является ли заданное выражение константой, функцией или предикатом в зависимости от его местоположения в факте, правиле или запросе. Метод распознавания будет наглядно продемонстрирован в следующем параграфе после того, как мы рассмотрим древесное представление предикатов.

Значительное преимущество использования составных предикатов можно увидеть на примерах баз данных семьи (*) и (**). Составные предикаты дают возможность упрощать выражения и гибко манипулировать с данными. Например, если мы интересуемся родителями семьи, то соответствующие запросы к программе (*) будут иметь вид

```
? отец(X).
? мать(Y).
```

в то время как воспользовавшись одним предикатом, мы могли бы позволить себе единственный запрос к (**)

```
? семья(отец(X), мать(Y), _).   (**)
```

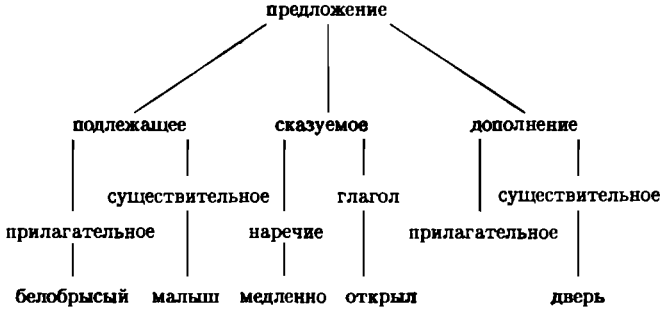
в котором анонимная переменная (см. п. 3.3.3) обозначает, что мы не интересуемся значениями функции «дети».

3.3.6. *Представление предикатов деревьями*

Рассмотрим пример синтаксического анализа некоторого предложения естественного языка. Мы можем для простоты считать, что предложение образовано из подлежащего, сказуемого и дополнения. Подлежащее и дополнение являются выражениями, состоящими из существительного и прилагательного, а группа сказуемого — выражением, состоящим из глагола и наречия. Поэтому для синтаксического анализа фразы

белобрысый малыш медленно открыл дверь

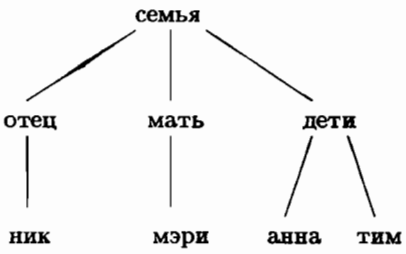
можно воспользоваться деревом [Thay88, Xant90], висячим вершинам которого приписаны слова этой фразы



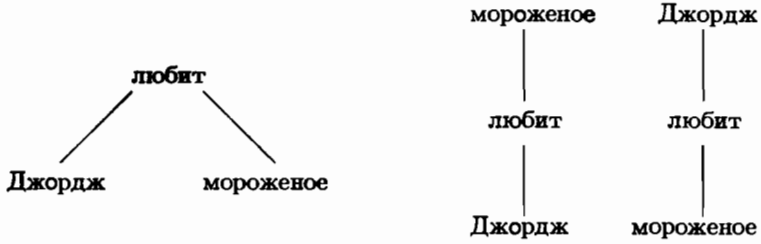
Та же самая методика синтаксического разбора применима и к предикатам: каждый предикат представляется деревом. Корнем дерева (Определение 2.8.9) служит рассматриваемый предикатный символ. Внутренние вершины помечены функциональными символами, входящими в состав предиката. Висячие вершины соответствуют переменным или константам, встречающимся среди аргументов функций и предиката.

Поэтому данные (*) из предыдущего параграфа могут быть представлены четырьмя деревьями

тогда как сложный предикат (**) соответствует дереву



Выражения
«любит(джордж, мороженое)»,
«мороженое(любит(джордж))» и
«джордж(любит(мороженое))»
представляются следующими структурами



Как мы увидим в § 3.4, используя древесное представление предикатов, ПРОЛОГ осуществляет унификацию. После проведения унификации двух предикатов образуются деревья, которые будут в точности совпадать.

3.3.7. *Списки*

Список — это просто перечень данных, заключенный в квадратные скобки «[» и «]», как например

```
[вино,мыло,джон,X,дом]
```

Те самые списки, которые служат основным элементом конструкций ЛИСПа [WiBe89], были встроены и в ПРОЛОГ. Используя списки, мы имеем возможность собирать информацию в одном объекте ПРОЛОГа. Например, три факта

```
любит(джордж,футбол).
любит(джордж,пиво).
любит(джордж,чипсы).
```

можно заменить, воспользовавшись списком, одним фактом

```
любит(джордж,[футбол,пиво,чипсы]).
```

в котором набор термов **футбол,пиво,чипсы** представлен объектом

[футбол,пиво,чипсы].

В общем случае, список образован последовательностью термов[1]).

Применяя списки, мы экономим память и получаем удобное средство для представления и обработки сложных данных.

Списки, вообще говоря, определяются рекурсивно следующим образом. Списком может быть

а) *пустой список*, т. е. список, не содержащий ни одного элемента (обозначается «[ ]»).

б) *непустой список*, образованный парой термов

1) первый терм называется *заголовком* списка.

2) второй терм называется *хвостом*, или *телом списка*.

Заголовком списка может быть произвольный терм ПРОЛОГа: константа, переменная или функция[2]). *Хвостом списка обязательно должен быть список*. Например, заголовком списка

[футбол,пиво,чипсы]

является константа **футбол**, а хвостом этого списка является список

[пиво,чипсы]

Обратившись к рекурсивному определению функций в логике предикатов, Определение 2.2.1, мы видим, что список не является функцией. Более того, списки формируются на основе общего свойства или отношения, которым обладают все элементы списка. Например, список

[мясо,хлеб]

может обозначать продукты, которые нужно купить, и в этом случае он выражается предикатом

купить(мясо,хлеб)

Поэтому список можно рассматривать как предикат, воспользовавшись *предикатным символом* «.», обозначающим наличие некоторого предиката на соответствующем месте. Например запись

.(салат,винегрет)

обозначает список

[салат,винегрет]

без указания на качественные особенности термов «**салат**» и «**винегрет**» и взаимосвязь между ними. Поэтому выражение «**.(салат,винегрет)**» можно рассматривать как *обобщенный предикат*, точная форма которого заранее неизвестна. Поэтому нельзя обращаться к ПРОЛОГу с запросами

? [салат,винегрет].

---

[1]) В числе которых могут быть и списки. — *Прим. перев.*

[2]) А также некоторый другой список. — *Прим. перев.*

или

```
? [X,Y].
```

или

```
? [X,винегрет].
```

поскольку нам неизвестно имя предиката, соответствующего «.». Более того, в одной и той же программе могут присутствовать несколько списков, и все они выражаются при помощи специального символа «.».

В общем случае список может быть представлен как предикат

```
.(Head,Tail)
```

Хвост списка, в свою очередь, является списком с заголовком и хвостом[1]). Если же мы рассматриваем «.» как *функтор*, т. е. символ обозначающий некоторую функцию, то общий вид списка будет таков

```
.(Head1,.(Head2, .((Head3,....(Head,[])...))),
```

где [ ] обозначает пустой список.

Например список [*a*, *b*, *c*] в общем виде представляется как

```
.(a,.(b, .(c,[])))
```

Список, состоящий из одного элемента может иметь вид

```
[элемент]   или   .(элемент,[])
```

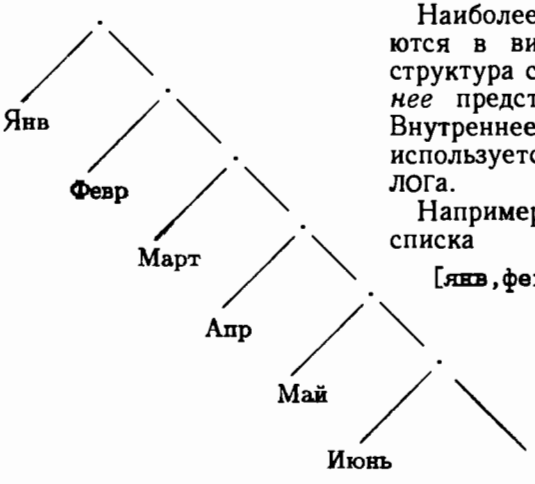


Рис. 3.1

Наиболее часто списки реализуются в виде деревьев. Древесная структура списка образует *внутреннее* представление этого объекта. Внутреннее представление списков используется интерпретатором ПРОЛОГа.

Например, древесная структура списка

```
[янв,февр,март,апр,май,июнь]
```

изображена на рис. 3.1, где символ «.», стоящий в корне дерева, обозначает предикат, а тот же самый символ «.», стоящий в промежуточных вершинах обозначает функцию.

[1]) **Или пустым** списком. — *Прим. перев.*

Во всех редакциях ПРОЛОГа имеется очень развитая и гибкая система обозначений для списков, облегчающая обработку длинных списков и списков переменной длины. Представление списков основано на применении символов «[», «]», а также «:» или «|». Например список

$$[a, b, c, d]$$

с заголовком A и хвостом $[b, c, d]$ записывается в виде

$$[H: T]$$

где переменные H и T соответствуют термам a и $[b, c, d]$.

§ 3.4. Аппарат вычислений

3.4.1. *Процедура унификации в ПРОЛОГе*

Процедура *унификации* термов (§ 2.9) вместе с реализующим ее алгоритмом (см. п. 2.9.9) является одной из наиболее важных операций в ПРОЛОГе.

Процедура унификации отсутствует в других языках программирования высокого уровня. И хотя в ЛИСПе унификация выступает в качестве основного свойства операций, в этом языке не применяется метод резолюций для построения вывода. Вместе с тем, способность ПРОЛОГа строить эффективные выводы основывается на удивительно слаженном взаимодействии процедуры унификации и резолюции.

Унификация позволяет связывать переменные с заданными объектами языка ПРОЛОГ. Например, сообщение ПРОЛОГа

$$X = \textbf{футбол}$$

означает, что переменная X приняла указанное значение и стала связанной с константой **«футбол»**.

По существу, проводя унификацию, мы проверяем, можно ли привести два предиката к единому виду. Если это невозможно, то унификация считается неудавшейся. Если же эти предикаты можно привести к общему виду, то унификация успешно завершается. В результате ее выполнения переменным обоих предикатов присваиваются такие значения, при которых эти предикаты будут отождествлены (т. е. будут совпадать).

В математической литературе термин «унификация» нередко заменяется термином «отождествление», а вместо термина «присвоить значение» используются термины «связать» или «подставить». Связывание переменных понимается здесь совсем в другом смысле, нежели связывание свободных переменных (Определения 2.2.12 и 2.2.13), поскольку все переменные в ПРОЛОГе уже связаны квантором всеобщности (Определения 2.4.1 и 2.4.4). Термин «связать» означает в ПРОЛОГе, что переменная приняла некоторое определенное значение в процессе выполнения программы.

При помощи алгоритма унификации и общего унификатора (Алгоритм 2.9.7 и Определение 2.9.4.) ПРОЛОГ может выполнять сложные подстановки. Наилучшим образом эту процедуру можно описать, воспользовавшись древесным представлением предикатов.

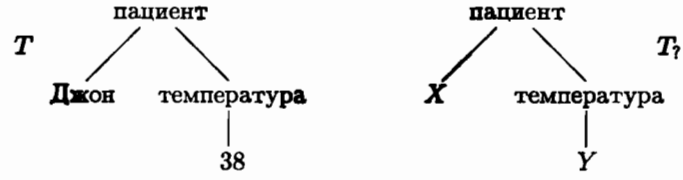
Предположим, что база данных в ПРОЛОГе содержит факт

```
пациент(джон, температура(38)).        (*)
```

и мы обращаемся к ней с запросом

```
? пациент(X, температура(Y)).
```

Древесные структуры T и $T_?$, соответствующие факту (*) и нашему запросу выглядят следующим образом



Если мы наложим дерево $T_?$ прямо поверх дерева T, то для того, чтобы добиться полного совпадения, переменные дерева $T_?$ должны быть унифицированы, или отождествлены, с соответствующими термами дерева T, поскольку наименования предикатов и количество дуг в обоих деревьях одинаковы. Поэтому в ответ на наш запрос ПРОЛОГ образует подстановку

X = джон
$Y = 38$

В общем случае в процессе унификации, проверяя возможность отождествления A и B, необходимо соблюдать следующие правила [Xant90].

1) если A и B — константы, то A и B отождествимы только в том случае, когда они соответствуют одному и тому же объекту,

2) если A — переменная, а B — произвольное выражение, то A отождествляется с B. Если B — переменная, то B отождествляется с A[1]).

3) если A и B — функции, то A и B отождествимы только тогда, когда

а) A и B имеют одинаковые заглавные функциональные символы,

[1]) Это правило унификации — так называемая *унификация без проверки вхождения* — в общем случае некорректно и применяется исключительно в целях повышения эффективности выполнения ПРОЛОГ-программ. Для того, чтобы переменная A и сложный терм B были унифицируемы необходимо также потребовать, чтобы A не имела вхождений в B. — *Прим. перев.*

б) соответствующие подтермы A и B попарно отождествимы[1]).

4) если A и B — предикаты, то A и B отождествимы тогда и только тогда, когда

а) A и B имеют одинаковые заглавные предикатные символы,

б) соответствующие термы-аргументы A и B попарно отождествимы[2]).

Перечисленные правила согласования свидетельствуют о том, что процедура унификации выполняется *рекурсивно*. Правила (3) и (4), или точнее подпункты (3а) и (4а) этих правил, обращаются к пунктам 1, 2, 3 и 4 неформального описания унификации, приведенного в § 2.9. Рекурсивную сущность этих правил демонстрирует следующий пример.

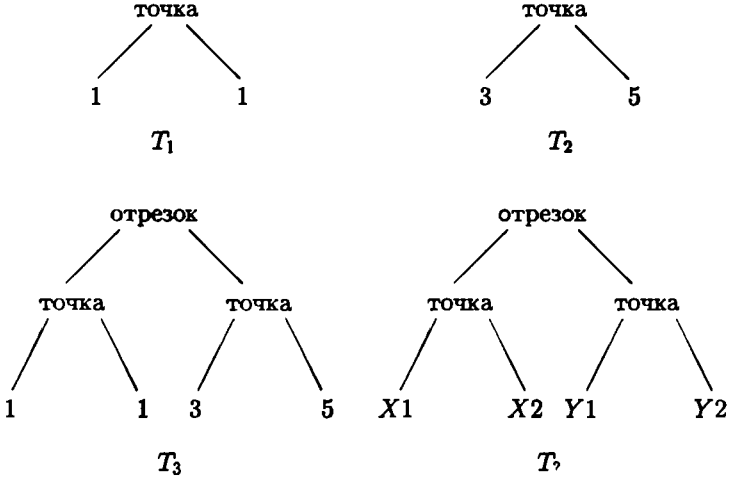
Нам предъявлены следующие геометрические данные

```
точка(1,1).
точка(3,5).
? отрезок(точка(1,1), точка(3,5)).
```

а также запрос

```
отрезок(точка(X1,X2), точка(Y1,Y2)).
```

Соответствующие им древесные структуры таковы



Тогда правила унификации будут применяться в следующем порядке

[1]) Имеется в виду отождествимы одновременно при помощи одной и той же подстановки — *Прим. перев.*

[2]) И здесь также имеется в виду, что эти пары термов отождествимы одновременно при помощи одной и той же подстановки. — *Прим. перев.*

(1)	отрезок	отрезок	(4а)
(2)	точка	точка	(4б, 3а в (1))
(3)	$X1$	1	
	$X2$	1	(3б в (2))
(4)	точка	точка	(4б, 3а в (1))
(5)	$Y1$	3	
	$Y2$	5	(3б в (2))

Поэтому деревья T_3 и T_5 могут быть унифицированы, или отождествлены, путем означивания переменных

$$X1 = 1$$
$$X2 = 1$$
$$Y1 = 3$$
$$Y2 = 5$$

Рекурсивное определение отношений составляет одну из динамических характеристик ПРОЛОГа, которая будет рассматриваться более подробно в § 3.4.5.

3.4.2. *Вывод и процедура отката*

Как мы уже упоминали, программа в ПРОЛОГе допускает двоякое истолкование — *логическое*, или *декларативное*, и *процедурное*. В этом параграфе мы займемся процедурной интерпретацией ПРОЛОГа. Мы опишем, *как именно* ПРОЛОГ осуществляет вывод заключений.

Мы уже ознакомили читателя с процедурой унификации и резолютивным методом. Применяя процедуру унификации, мы строим множество основных атомов программы (Определение 2.2.15). На следующем шаге мы повторно применяем правило резолюции к этому множеству основных атомов и устанавливаем в итоге справедливость заключений, выводя пустой дизъюнкт. Используя метод резолюций, мы уверены, что любое высказывание, которое может принять истинное значение, получит это значение. Однако, в ПРОЛОГе мы не можем ориентировать вывод на интересующее нас высказывание, а это чрезвычайно важно для эффективного выполнения программ.

Рассмотрим, например, следующие высказывания пропозициональной логики

(1) D
(2) E
(3) $B \vee \neg E$
(4) $A \vee \neg D$
(5) $A \vee \neg B$

и зададимся вопросом, всегда ли в этом случае будет справедливо A. Мы добавляем высказывание

(6) $\neg A$

и получаем следующее резолютивное доказательство A.

(7)	B	из (2) и (3)
(8)	A	из (7) и (5)
(9)	$\square$	из (8) и (6)

Но с равным успехом можно построить и другое резолютивное доказательство A.

(7')	A	из (1) и (4)
(8')	$\square$	из (7') и (6)

Первое доказательство длиннее второго, и для его построения требуется больше времени. Кроме того, в первом доказательстве нам пришлось вывести высказывание B, не имеющее никакого отношения к A.

Аппарат вывода в ПРОЛОГе содержит специальную процедуру поиска в пространстве атомов программы, которая, с одной стороны, позволяет программисту управлять последовательностью вычислений, а с другой стороны, гарантирует, что все возможные решения задачи будут найдены [Lloy87]. Эта процедура называется *откатом*. Рассмотрим, например, следующую базу данных в ПРОЛОГе.

```
вор(джон)                                          /* 1 */
любит(мэри,шоколад)                                /* 2 */
любит(мэри,вино)                                   /* 3 */
любит(джон,X):-любит(X,вино)                       /* 4 */
может_похитить(X,Y):-вор(X),любит(X,Y)             /* 5 */
```

В большинстве редакций ПРОЛОГа комбинация символов /* открывает, а комбинация */ закрывает комментарии, не исполняемые программой. В приведенной программе мы использовали в качестве комментариев номера дизъюнктов.

Предположим, что мы обратились к программе с целевым запросом

```
? может_похитить(джон,Z) /* что может похитить Джон */
```

Для того, чтобы ответить на этот запрос придется выполнить следующие действия.

Этап 1.

```
? может_похитить(джон,Z).
```

X/Джон Z/Y вор(Джон)	(5)	Проверяется возможность унификации целевого утверждения с фактом или заголовком какого-либо правила базы данных. Унификация возможна с заголовком правила (5) при помощи подстановки X/Джон, Z/Y. Запоминается порядковый номер правила (5). Все переменные, входящие в тело программного утверждения (5), изменяют свои значения в соответствии с унифицирующей подстановкой

Этап 2.

? **вор(*джон*), любит(джон,*Y*).**

да

(1) ПРОЛОГ пытается выполнить поочередно каждую подцель. В базе данных, начиная с первого дизъюнкта (1), проводится поиск программного утверждения, заголовок котрого может быть унифицирован с первой подцелью. Унификация успешно применима к дизъюнкту (1) Этот дизъюнкт является фактом, и поэтому подцель выполнена

Этап 3.

? **любит(джон,*Y*).**

X/Y
любит(Y,вино)

(4) ПРОЛОГ пытается выполнить вторую подцель. В базе данных, начиная с первого дизъюнкта (1), проводится поиск программного утверждения для унификации с этой подцелью Унификация успешно применима к дизъюнкту (4) с использованием подстановки X/Y Переменные в теле правила (4) немедленно изменяют свои значения в соответствие с этой подстановкой.

Этап 4.

? **любит(*Y*,вино).**

Предпринимается попытка выполнить дизъюнкт, образующий тело правила (4). База данных вновь просматривается с самого начала в поисках подходящего для унификации программного утверждения.

Этап 5.

Y/Мэри

Попытка унифицировать подцель с фактом (2) терпит неудачу, поскольку "вино" и "шоколад" не унифицируемы Порядковый номер дизъюнкта (2), по отношению к которому была предпринята унификация, запоминается

(2) Y/Мэри (3)

нет да

Этап 6.

ПРОЛОГ пытается найти иной способ выполнения подцели и переходит к следующему дизъюнкту программы (3) Здесь унификация успешна

Этап 7.

Исходная цель «**может_похитить**(джон, *Z*)» была достигнута посредством подстановки *Z*/Мэри. Вслед за этим ПРОЛОГ предпринимает попытку достичь этой цели другим путем. Для этого производится *откат к ближайшей предшествующей подцели*, которую удалось достичь, т. е. к «**любит**(*Y*,**вино**)», с тем, чтобы провести ее унификацию с другим хорновским дизъюнктом этой программы. На этом этапе вычисления ПРОЛОГ *освобождает все переменные от тех значений, которые были присвоены им ранее.* Затем просматриваются все дизъюнкты, расположенные в программе вслед за последним дизъюнктом, с которым удалось унифицировать эту подцель (в данном случае, следующие за дизъюнктом (3)). В ПРОЛОГе расположение *вслед* означает, что соответствующий дизъюнкт име-

ет списковый номер больший, нежели последний дизъюнкт, с которым была проведена унификация (см. п. 3.1.2). Поскольку номер исполняемого дизъюнкта заносится в память на каждом шаге вычисления, ПРОЛОГ может тем самым помечать все расположенные ниже дизъюнкты. Ввиду того, что в нашей программе отсутствуют другие факты или правила, заголовки которых могут быть унифицированы с данной подцелью, ПРОЛОГ прекращает поиск.

Из приведенного описания функционирования программы видно, что ПРОЛОГ пытается выполнить *одну цель на каждом шаге вычисления*. Эта особенность механизма вывода в ПРОЛОГе, или точнее, процедуры отката, наделяет ПРОЛОГ свойством *детерминированности*.

Порядок, в котором программные утверждения заносятся в базу данных, играет важную роль при выполнении и управлении ПРОЛОГ-программой. Попытка ПРОЛОГа отыскать решение для заданной (под)цели программы путем просмотра базы данных в глубину, представляет собой одну из основных алгоритмических стратегий обхода дерева поиска — *стратегию поиск в глубину с возвратом*. Мы перейдем сейчас к описанию этого алгоритма, составляющего ядро процедуры отката.

3.4.3. *Поиск в глубину с возвратом*

Применение деревьев для поиска решения задачи основано на представлении начальных данных, равно как и решений самой задачи, в виде *пространства состояний* [Nils71]. Пространством состояний называется дерево, в котором

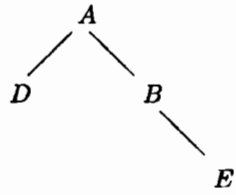
а) вершины соответствуют различным стадиям решения проблемы: *начальное состояние*, *промежуточные состояния*, возникающие в ходе решения задачи, и *заключительные состояния*;

б) дуги, соединяющие вершины, соответствуют допустимым переходам из одного состояния в другое.

Решение проблемы, таким образом, сводится к *поиску пути из начального состояния в заключительное состояние* [Brat90].

Осуществляя процедуру выполнения сложного целевого запроса, ПРОЛОГ просматривает пространство состояний. Целевой запрос становится корневой вершиной соответствующего дерева. Промежуточные вершины соответствуют промежуточным подцелям, которые всякий раз нужно достичь. Заключительные состояния соответствуют фактам или следствиям из фактов базы данных (Определение 2.5.21). Выбор пути из одного состояния в другое равносилен выбору соответствующих подцелей. Поэтому пространство состояний программы в первом примере § 3.2.2 может быть описано следую-

щим деревом,



в котором дизъюнкт A является начальным состоянием, целью задачи, а дизъюнкты D и E — заключительными состояниями, фактами базы данных. Поэтому для решения задачи вывода A имеются два пути решения, которые соответствуют двум различным множествам допустимых переходов

$$\{D \longrightarrow A\}$$

и

$$\{E \longrightarrow B,\ B \longrightarrow A\}.$$

Возникает вопрос, как в заданном пространстве состояний задачи выбрать подходящий *путь решения* задачи, и даже более того, как отыскать *все* альтернативные пути решения. Мы можем получить ответ на эти вопросы, воспользовавшись *алгоритмом поиска в глубину с возвратом* для обхода дерева.

Идея алгоритма поиска в глубину с возвратом основана на следующих очевидных соображениях.

Поиск пути решения, **Путь_решения**, из вершины N дерева в вершину *заключительного* состояния нужно проводить, руководствуясь двумя правилами.

а) Если N — вершина заключительного состояния, то **Путь_решения** представлен списком $[N]$.

б) Если N имеет последователя $N1$, для которого построен **Путь_решения_1**, ведущий из вершины $N1$ в разрешающую вершину, то **Путь_решения** задается списком $[N$: **Путь_решения_1**$]$.

Алгоритм исполняется итеративно до тех пор, пока все возможные пути решения не будут найдены. Каждая вершина-последователь, для которой проверяется наличие пути решения, должна располагаться в дереве глубже и левее своего непосредственного предшественника, считая от начальной вершины. Когда ПРОЛОГу нужно вернуться, для того чтобы отыскать альтернативные пути, совершается *откат* к непосредственно предшествующей вершине. Отсюда и название алгоритма.

Таким образом, алгоритм поиска в глубину с возвратом может быть представлен следующей ПРОЛОГ-программой.

```
найти_путь(N,[N]) : - цель(N).
найти_путь(N,[N: путь_решения]) : -
  допустимый_переход(N,N1),найти_путь(N1,путь_решения).
```

Может показаться странным, что в этой программе, а точнее, во втором программном утверждении, мы используем предикат «найти_путь» для определения самого предиката «найти_путь»! Подобные *рекурсивные определения отношений* между объектами широко используются в ПРОЛОГе и будут рассмотрены более подробно в § 3.4.5.

Способность ПРОЛОГа к определению его собственных структурных и функциональных элементов, наподобие алгоритма поиска в глубину и рекурсии, является одним из признаков динамичности этого языка. Эта способность проистекает из декларативной и логической природы ПРОЛОГ-программ и ее называют ПРОЛОГ *посредством самого* ПРОЛОГа.

3.4.4. *Управление откатом: отсечение*

В языке ПРОЛОГ имеется специальный предикат *отсечения*, обозначаемый символом «!», который служит для управления процедурой вывода, или точнее, для управления откатом. «Отсечение» применяется в ПРОЛОГе для предотвращения выбора определенных путей в процессе обхода пространства состояний задачи. Это бывает необходимо в тех случаях, когда пользователь либо знает заранее, что эти пути не ведут к решению задачи, либо желает сократить время вычисления программы. Помимо этого отсечение применяется императивно (принудительно) в тех случаях, когда пользователь сознательно отказывается от рассмотрения следствий из некоторых определенных фактов (использует отсечение как форму неприятия фактов).

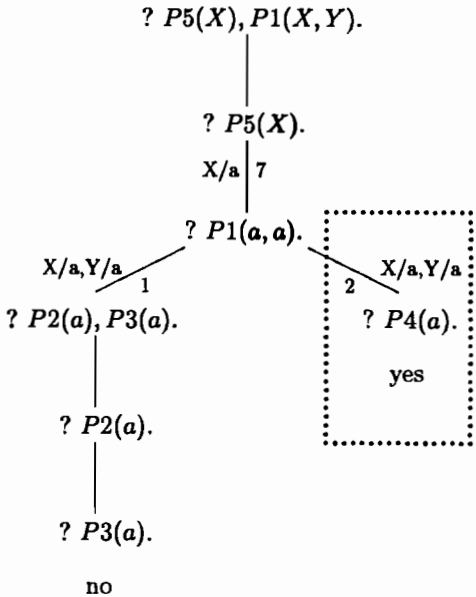
«Отсечение» как логический предикат всегда считается истинным. Это означает, что оно всегда должно быть выполнено в случае его вызова. Интересно проследить последствия использования «отсечения» при вычислении и построении заключений ПРОЛОГ-программ. Рассмотрим, например, следующую базу данных ПРОЛОГа [Thay88].

```
P1(X,Y) : - P2(X), P3(Y).        /* 1 */
P1(X,Y) : - P4(Y).               /* 2 */
P2(a).                           /* 3 */
P3(b).                           /* 4 */
P4(a).                           /* 5 */
P5(a).                           /* 6 */
P5(b).                           /* 7 */
```

и запрос

```
? P5(X), P1(X,Y).
```

Пространство состояний, содержащее единственный путь решения задачи, представлено деревом



ПРОЛОГ решает первоначальный запрос «$P5(X),\ P1(X, Y).$» с использованием подстановки X/a.

Давайте посмотрим теперь, что случится, если мы вставим «отсечение» в тело программного утверждения /* 1 */ так, чтобы этот дизъюнкт принял вид

$$P1(X, Y) :- P2(X),\ !,\ P3(Y). \qquad /* 1' */$$

Сначала ПРОЛОГ попытается разрешить начальную цель, выбрав левый путь. Значит ему придется разрешать подцель «$P1(a, a)$», активизируя дизъюнкт /* 1' */. После того, как будет решена подцель «$P2(a)$» настанет очередь отсечения, !, которое выполнится немедленно, поскольку предикат отсечения всегда является истинным. Вычисление продолжится, и будет предпринята попытка унифицировать $P3(a)$, которая окончится неудачей, поскольку в программе нет дизъюнктов, заголовки которых могут быть унифицированы с $P3(a)$. В обычном случае ПРОЛОГ должен провести откат к подцели «$P1(a, a)$» и продолжить вычисление активизируя дизъюнкт /* 2 */. Но поскольку в теле /* 1' */ содержалось отсечение, которое было выполнено, отката не произойдет. *Отсечение, !, не допускает повторную обработку подцели, которая была унифицирована с заголовком процедуры, содержащей в теле этот предикат отсечения.* И хотя в том случае, когда программа содержит дизъюнкт /* 1 */, первоначальная цель будет выполнена, если

последовать правым путем вычисления, применяя подстановку X/a и Y/a, тем не менее, эта же цель не будут достигнута в том случае, когда программа содержит дизъюнкт /* 1′ */. Это означает, что ПРОЛОГ выдаст ответ «нет». Другими словами, правый путь решения, заключенный в прямоугольную рамку, вырезан и удален из пространства состояний. В пространствах состояний, имеющих более сложное устройство, применение «отсечения» приводит к удалению из пространства целых поддеревьев. Вот этой способности отсекать поддеревья решений и обязан своим названием предикат «отсечения».

Интересно посмотреть на то, как при помощи оператора «отсечения», мы можем избавить ПРОЛОГ-программы от зацикливания, возникающего при бесконечно долгом выполнении операций вследствие использования неправильно описанных данных. Давайте исследуем поведение следующей ПРОЛОГ-программы

```
вор(джон).                          /* 1 */
вор(X) : - вор(X).                  /* 2 */
```

при обращении к ней с запросом

```
? вор(X).                           /* 3 */
```

Унифицируя запрос с фактом /* 1 */, ПРОЛОГ немедленно получит ответ

X =джон,

и затем, освободив переменную X от значения «джон», попытается выполнить запрос другим способом, обратившись к дизъюнкту /* 2 */. После применения правила резолюции образуется новая цель

```
? вор(X).                           /* 4 */
```

совпадающая с первоначальным запросом /* 3 */. Дизъюнкт /* 4 */ унифицируем с заголовком программного утверждения /* 2 */, и после унификации будет вновь получено целевое утверждение /* 4 */[1]). Это означает, что программа попала в бесконечный цикл, из которого нет выхода.

Посмотрим теперь, как та же самая программа, в которой дизъюнкт /* 2 */ снабжен предикатом отсечения

```
вор(джон).                          /* 1 */
вор(X) : - вор(X), !.               /* 2′ */
```

будет отвечать на запрос

```
? вор(X).                           /* 3 */
```

Дизъюнкт /* 1 */ тут же даст ответ

X =джон,

[1]) Если говорить более строго, то ПРОЛОГ, имея запрос /* 4 */, сначала вновь активизирует факт /* 1 */, еще раз получит ответ X =джон, и лишь затем обратится к дизъюнкту /* 2 */ — *Прим. перев*

ПРОЛОГ попытается отыскать все возможные решения: унифицируя запрос с заголовком программного утверждения /* 2' */, разрешит его, получит новую цель

```
? вор(X), !.                                    /* 4' */
```

и остановится, поскольку «отсечение» не позволяет во второй раз обратиться к заголовку правила /* 2' */, в теле которого встречается отсечение[1]). Поэтому единственным ответом будет

X =**джон**,

и программа будет выполнена без зацикливания.

Функционирование «отсечения» может быть описано только в терминах процедурной интерпретации ПРОЛОГ-программ. Это означает, что ответственность за последствия использования отсечения и управление вычислением возлагается на программиста. Необходимость включения «отсечения» в состав языка ПРОЛОГ постоянно оспаривается теми, кто хочет видеть ПРОЛОГ чисто логическим языком программирования. Эти возражения опираются на следующие доводы.

1) ПРОЛОГ-программа без «отсечения» способна, используя откат, находить все правильные ответы на заданный запрос (система вычислений правильных ответов обладает свойством полноты). С введением «отсечения» это свойство полноты утрачивается, так как «отсечение» не позволяет программе вычислить любые правильные ответы.

2) Воспользовавшись «отсечением», мы получаем возможность создавать логически некорректные ПРОЛОГ-программы, которые, несмотря на скрытую возможность логического вывода из них неправильных заключений, работают корректно в случае применения «отсечения».

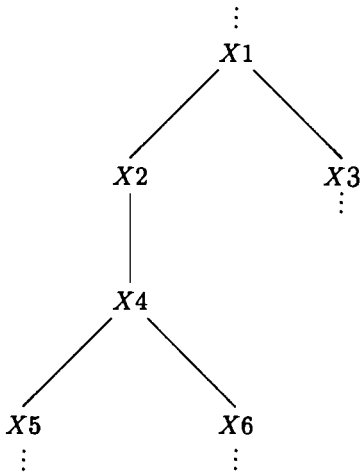
Однако, использование «отсечения» в некоторых программах позволяет повысить их эффективность без нарушения логической целостности.

3.4.5. *Рекурсивные определения в ПРОЛОГе*

Используя *рекурсивные определения*, мы можем определять новые предикаты в ПРОЛОГ-программе .Предположим, например, что мы хотим определить отношение **«предок»** для изучения одного из фрагментов фамильного древа.

---

[1]) На самом деле ПРОЛОГ сначала обратится к факту /* 1 */, еще раз получит тот же самый ответ X =**джон**, и лишь затем сработает отсечение, препятствуя повторному обращению к дизъюнкту /* 2' */. — *Прим. перев.*



Каждая вершина обозначает члена семьи, а связь между вершинами обозначает отношение "родитель"

Мы можем заметить, что для каждой пары лиц X, Z, непосредственно связанных друг с другом в фамильном древе, X является предком Z, если X — родитель Z. Поэтому отношение «быть предком» может быть определено посредством хорновского дизъюнкта

```
предок(X, Z) : – родитель(X, Z).
```
(a)

Таким образом, используя предикат **«родитель»**, мы определяем предикат **«предок»** для людей, отстоящих друг от друга на одно поколение.

При определении отношения «быть предком» для людей, отстоящих друг от друга на два поколения, например, для $X1$ и $X4$, мы замечаем, что для каждой пары лиц X, Z в фамильном древе, X является предком Z, если X является родителем некоторого Y, который, в свою очередь, является родителем Z.

Тогда мы записываем дизъюнкт

```
предок(X, Z) : – родитель(X, Y), родитель(Y, Z).
```
(b)

Проводя попытку проверить отношение «быть предком» между лицами, отстоящими друг от друга более, чем на два поколения, нам придется строить последовательность программных утверждений нарастающей длины.

```
предок(X, Z) : – родитель(X, Y1), родитель(Y1, Y2),
                 родитель(Y2, Z).
предок(X, Z) : – родитель(X, Y1), родитель(Y1, Y2),
                 родитель(Y2, Y3), родитель(Y3, Z).
```

Погружаясь вглубь фамильного древа, мы вынуждены определять все новые и новые правила. Это, вне всяких сомнений, делает нашу программу неэффективной, поскольку независимо от выбранного языка, программа считается полезной и эффективной лишь тогда,

когда она может найти решение общей задачи, а не только ее отдельных специальных примеров.

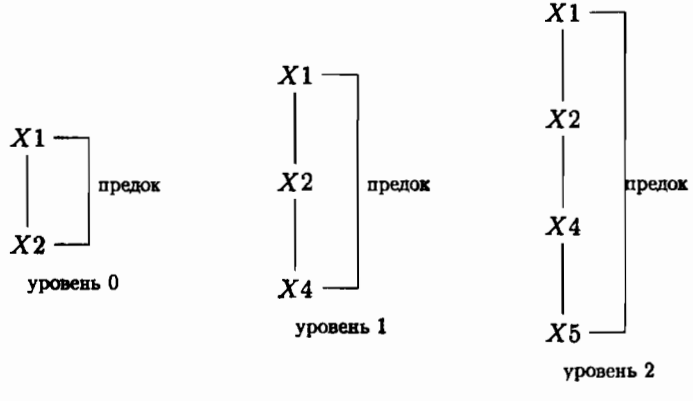
Но существует и более удобный способ определения отношения «быть предком».

Заметим, что для каждой пары лиц X, Z в фамильном древе, X является предком Z, если X является родителем некоторого Y, который, в свою очередь, является предком Z.

Исходя из этого, мы имеем возможность записать следующее программное утверждение ПРОЛОГа

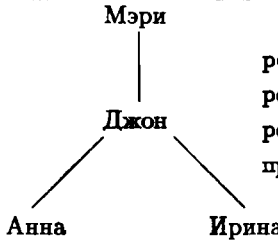
$$\textbf{предок}(X, Z) \ : - \ \textbf{родитель}(X, Y), \ \textbf{предок}(Y, Z). \qquad \text{(c)}$$

В этом описании мы использовали предикат «предок» для определения самого предиката «предок»! Определения такого вида называются *рекурсивными определениями*, а отношения, которые ими обозначаются, называются *рекурсивными отношениями*. Мы понимаем интуитивно, что *рекурсия зарождается в тот момент, когда набросок решения строится путем воспроизведения этого же наброска в ином масштабе или на другом уровне* [Xant90]. Так, в примере с фамильным древом описание отношения «предок» между $X1$ и $X5$ воспроизводится на разных уровнях следующими диаграммами.



Правила (c) недостаточно для определения отношения «предок». Хотя с точки зрения логики в нем выражено все то, что мы хотим определить, оно все же не дает программе необходимой информации для получения ответа на запрос, касающийся отношения «предок».

Обратимся к примеру. Предположим, что нас снабдили фамильным древом и соответствующей базой данных ПРОЛОГа, представленными ниже,

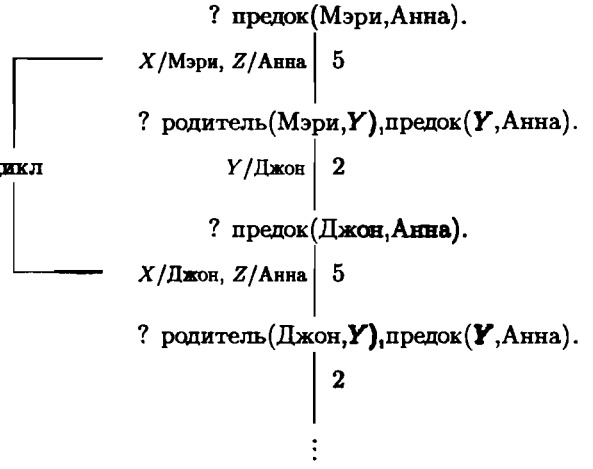



```
родитель(мэри,джон).                /*1*/
родитель(джон,анна).                /*2*/
родитель(джон,ирина).               /*3*/
предок(X,Z):- родитель(X,Y),
              предок(Y,Z).          /*5*/
```

и мы обратились с запросом

```
? предок(мэри,анна).
```

ПРОЛОГ попытается отыскать ответ на этот запрос в пространстве состояний, изображенном в виде следующего дерева



ПРОЛОГ не сможет завершить поиск решения проблемы в пространстве состояний. Эта ветвь дерева будет продолжаться бесконечно долго, и ПРОЛОГ будет все время проходить один и тот же цикл, пытаясь выполнить цель «**предок(*Y*,*Z*)**», означивая поразному ее переменные. И конечно, ПРОЛОГ не прервет выполнение программы, не имея возможности ответить «**да**» или «**нет**». Момент остановки программы будет определяться ограничениями вычислительных ресурсов компьютера, на котором установлена конкретная редакция ПРОЛОГа, а вовсе не естественным завершением вычисления ПРОЛОГ-программы.

В связи с этим нам нужен метод, который позволял бы надежно гарантировать нормальное завершение выполнения заданной рекурсивной процедуры. Для применения этого метода нужно завести еще один хорновский дизъюнкт, обеспечивающий *краевые условия*, при которых выполняется исследуемое отношение. В примере

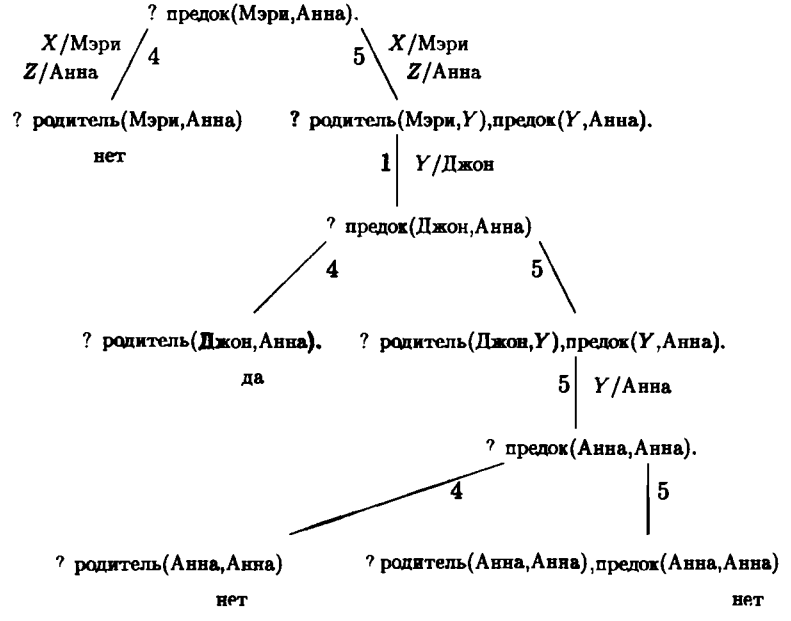
с фамильным древом таким условием будет отношение «родитель», означающее, что одно из лиц является предком другого лица, если оно, по крайней мере, является родителем этого другого лица, т. е.

Для всех X, Y в фамильном древе X — предок Z, если X — родитель Z.

Нам нужно, таким образом, добавить к базе данных программное утверждение

```
предок(X, Z) : - родитель(X, Z).        /* 4 */
```

Дизъюнкт /* 4 */ играет также роль *граничного условия*, описываемого отношения. С введением /* 4 */ пространство состояний, соответствующее ответу на первоначальный запрос, будет представлено деревом



Итак, добавляя дизъюнкт /* 4 */, мы вынуждаем ПРОЛОГ завершить поиск в пространстве состояний и ответить положительно на наш запрос.

Краевое условие рекурсивного отношения называют также *рекурсивным базисом*. К выбору ограничения для рекурсивного отношения и соответствующего ему программного утверждения нужно отнестись очень внимательно. Имея дело с задачей, допускающей рекурсивное решение, следует придерживаться следующих принципов.

Общее решение задачи разбивается на два направления:

1) В первом направлении приводятся решения «простых» частных случаев задачи (обеспечиваем рекурсивный базис).

2) Во втором направлении рассматривается «общий» случай задачи, решение которого осуществляется сведением к упрощенному варианту этой же задачи (обеспечиваем рекурсивный переход).

Перед тем, как обратиться к рассмотрению специальных операций над списками, стоит сказать несколько слов об операции унификации над списками.

3.4.6. *Обработка списков*

Поскольку списки есть ни что иное, как абстрактные предикаты (см. § 3.2.4), они унифицируются точно так же, как и все прочие предикаты, а именно

Список $L1$ унифицируем со списком $L2$, если $L1$ и $L2$ имеют в своем составе одно и то же количество элементов, и все элементы, входящие в состав $L1$, унифицируемы с соответствующими элементами $L2$[1]).

Поэтому два списка могут быть унифицируемы тогда и только тогда, когда их заголовки и хвосты, т. е. собственно элементы списков, могут быть унифицированы. Например, для списков

$$L1 = [a: [b, c]]$$

и

$$L1 = [a: Y]$$

запрос к ПРОЛОГу

$$? \; L1 = L2$$

будет успешно выполнен с подстановкой $Y = [b, c]$. В свою очередь, списки

$$L1 = [a, b]$$

и

$$L2 = [\text{высота, цвет, местоположение}]$$

будут, напротив, не унифицируемы, поскольку имеют разное количество элементов.

Списки можно использовать для представления множеств. Однако, здесь имеется одна существенная особенность. Порядок расположения элементов в множестве не играет никакой роли, тогда как порядок расположения элементов списке очень важен[2]). Но тем не

---

[1]) При помощи одной и той же подстановки, общей для всех пар соответствующих элементов этих списков. — *Прим. перев.*

[2]) Следует также иметь в виду, что все элементы множества считаются попарно различными, в то время как списки могут содержать любое количество одинаковых элементов. Поэтому корректнее было бы сопоставлять списки не с множествами, а с *мультимножествами.* — *Прим перев*

менее, над списками выполняются такие же основные операции, как и над множествами. Поэтому, основываясь на рекурсивном определении списка и процедуре унификации, ниже будут предложены определения следующих операций:

1) Проверка принадлежности произвольного элемента или списка элементов некоторому списку, т. е. отношение принадлежности и включения для списков.

2) Конкатенация (последовательное соединение) в качестве объединения списков.

3.4.6.1. *Предикат* `member`

Предикат `member` применяется для проверки вхождения произвольного элемента в состав некоторого списка. Этот предикат встроен в большинство редакций ПРОЛОГа.

В общем случае считается, что элемент X входит в состав списка L, если

а) X — заголовок L, или

б) X — элемента хвоста списка L.

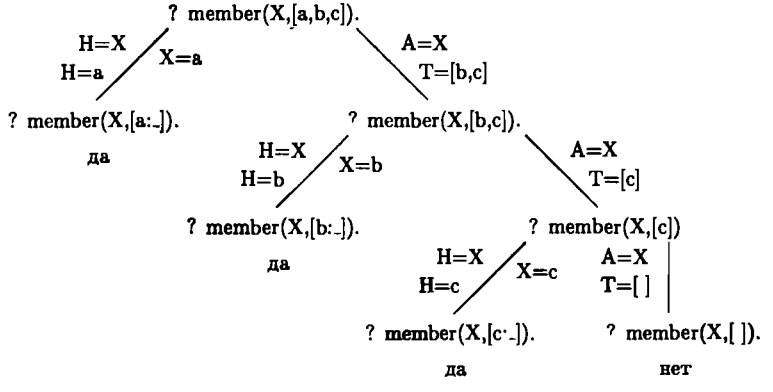
Поэтому мы можем привести следующее рекурсивное определение предиката `member`

(1) `member(`H`, [`H`: _]).`

(2) `member(`A`, [_: `T`]) : - member(`A`, `T`).`

где _ обозначает безымянную переменную. Дизъюнкт (1) задает базис рекурсии.

Например, пространство состояний для запроса `member(`X`, [`a`, `b`, `c`])` . представлено деревом



И поэтому элемент X входит в состав списка $[a, b, c]$ только в том случае, когда

$X = a$, или

$X = b$, или

$X = c$.

Легко видеть, таким образом, что ПРОЛОГ, пытаясь выполнить запрос

`member`(X, [a, b, c]).

подставит на место переменной X конкретную константу.

3.4.6.2. *Предикат* append

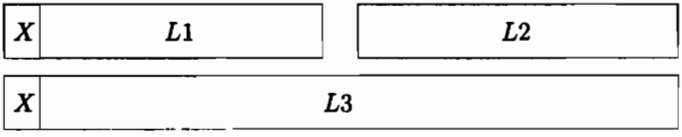
При помощи предиката append мы построим список $L3$, являющийся конкатенацией пары списков $L1$ и $L2$. Рассмотрим этот новый предикат append($L1, L2, L3$).

При описании append обратим внимание на следующие его свойства.

1) Результатом конкатенации некоторого заданного списка и пустого списка будет изначальный список. Поэтому справедлив такой факт

```
append([ ],L,L).                                    /* 1 */
```

2) Допустим, что первый список имеет вид [X: $L1$]. Если мы присоединим к нему список $L2$, то в результате получим список, заголовком которого будет заголовок первого списка, а его хвост образуется в результате конкатенации хвоста первого списка и $L2$. Это свойство можно проиллюстрировать диаграммой



На основании этого наблюдения, мы можем записать

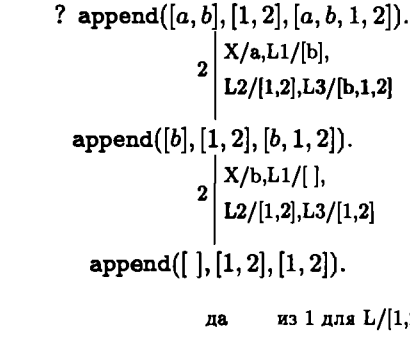
```
append([X:L1],L2,[X:L3]) :- append(L1,L2,L3). /* 2 */
```

Ясно, что дизъюнкты /* 1 */ и /* 2 */ представляют рекурсивное определение предиката append. Дизъюнкт /* 1 */ выступает в роли базиса рекурсии, а дизъюнкт /* 2 */ определяет рекурсивный переход, сводящий задачу конкатенации двух сложных списков к задаче конкатенации более простых списков, и осуществляет тем самым рекурсию.

Давайте обратимся к примеру. Предположим, что мы имеем запрос

```
? append([a,b], [1,2], [a,b,1,2]).
```

Пространство состояний вычисления ответа на это запрос изображено в виде дерева



Поскольку путь решения существует, ПРОЛОГ даст положительный ответ на запрос.

Предикат append встроен в большинство редакций и версий ПРОЛОГа. Некоторые другие предикаты, применяемые для обработки списков, представлены в упражнениях.

§ 3.5. Встроенные предикаты

Встроенными предикатами называются предикаты, определенные в самой системе ПРОЛОГ и предназначенные для обработки, размещения и управления данных, а также для организации взаимодействия пользователей с ПРОЛОГом.

3.5.1. *Предикаты размещения данных*

ПРОЛОГ-программу можно рассматривать как базу данных, содержащую факты и правила в качестве данных. Всякое изменение данных программы требует и обновления самой ПРОЛОГ-программы. В традиционных языках программирования обновление программы, т. е. добавление и изъятие данных, равно как и изменение структуры управления, осуществляется непосредственно самим программистом. В ПРОЛОГе, напротив, программа может обновляться автоматически в ходе своего выполнения, применяя специальные встроенные предикаты **assert** и **retract**.

Предикат **assert** употребляется для внесения в программу фактов и правил. Этот предикат имеет вид

assert(G),

где G — произвольное программное утверждение. Отметим, что предикат **assert**(G) всегда завершается успешно, и в результате его исполнения утверждение G добавляется к совокупности данных программы.

Предикат **assert** можно применять по-разному. Возьмем, например программу, состоящую из единственного дизъюнкта

человек(джордж). (1)

Если мы обратимся с запросом

? смертен(джордж).

ответом будет «нет», поскольку в программе нет определений для свойства быть смертным. Если же мы обратимся с запросом

? assert(смертен(джордж)).

то, благодаря успешному завершению **assert** программное утверждение

смертен(джордж). (2)

будет внесено в программу.

Встроенный предикат **assert** можно применять также для введения новых правил. В нашем примере мы можем вместо

? assert(смертен(джордж)).

обратиться с запросом

assert(смертен(*X*) : − человек(*X*)).

В результате этого вместо факта (2) программа пополнится новым правилом

смертен(*X*) : − человек(*X*). (2')

Мы можем выбрать также и то место, на которое будет поставлено в базе данных новое программное утверждение, применяя следующие вариации предиката **assert**:

asserta(*G*) и **assertz(*G*)**

которые позволяют поместить программное утверждение G соответственно в начало или в конец списка данных программы.

Подобно тому, как мы вносим данные в программу при помощи **assert**, мы можем удалять данные, применяя встроенный предикат

retract(*G*),

где G — программное утверждение. Использование **retract** дает эффект, противоположный действию **assert**.

3.5.2. *Предикаты взаимодействия*

ПРОЛОГ, как и всякий дружественный для пользователя язык программирования, располагает средствами, обеспечивающими взаимодействие программы с периферийными устройствами компьютера, с экраном и с пользователем. Эти средства представлены набором специальных встроенных предикатов, а именно **read**, **write** и **consult**.

Предикат **read** применяется для ввода символьной информации с клавиатуры. Он имеет вид

`read(`X`)`,

где X — переменная, обозначающая последовательность символов, которые программа должна прочитать. Предикат «`read(`X`)`» всегда завершается успешно, проводя унификацию переменной X с последовательностью входных символов. Поэтому, если мы напишем

`read(`X`)`,

то ПРОЛОГ будет ожидать ввода группы символов с клавиатуры. Если мы наберем на клавиатуре компьютера слово «**джон**», то в ПРОЛОГе предикат `read(`X`)` будет выполняться, унифицируя переменную X с символьной строкой **джон**.

Предикат записи

`write(`аргумент`)`,

функционирует подобно предикату «read» и применяется для отображения последовательности символов на экран. Аргументом здесь может быть любая символьная строка. Предикат записи всегда завершается успешно, выдавая значение его аргумента. Если `write` поместить в запрос, то значение его аргумента будет отправлено на печать.

Поэтому, если мы обратимся с запросом

```
?write("Джон - высокий").
```

то ПРОЛОГ выполнит `write`, напечатав на принтере

```
"Джон - высокий"
```

При помощи `read` и `write` мы можем создать целый ряд полезных программ. Например, программа

```
print(Имя,Номер) : - read(Имя), read(Номер),
                     write(Имя), write(" "),
                     write(Номер), nl.
```

заставляет ПРОЛОГ ожидать, пока мы введем две символьные строки, которые будут унифицированы с переменными **Имя** и **Номер**. Затем эти две символьные строки будут напечатаны на одной линии и разделены пробелом благодаря `write(" ")`. После этого выполнится специальный предикат `nl`, переводящий курсор экрана или принтер в начало новой строки.

Предикат `consult(`файл`)` применяется для прочтения ПРОЛОГ-программы из заданного файла.

Предикат `consult(`файл`)` всегда выполняется, помещая программные утверждения из файла в оперативную память компьютера с тем, чтобы ПРОЛОГ мог использовать их для вывода заключений.

3.5.3. *Равенство в ПРОЛОГе:*

В ПРОЛОГе имеется несколько различных предикатов для проверки равенства объектов. Наиболее употребительные предикаты, встроенные во всех редакциях ПРОЛОГа, таковы:

«=» Отношение $A = B$ считается успешно выполнимым, когда A соответствует B, т. е. когда A и B унифицируемы.

«==» Отношение $A == B$ считается выполнимым, когда A и B тождественно совпадают. Например, равенство

`отец(павел, энди)` = `отец(павел,` X`)`

успешно выполняется за счет подстановки X/Энди, в то время как равенство

`отец(павел, энди)` == `отец(павел,` X`)`

не выполняется.

«=:=» Отношение $A1$ =:= A2 выполняется, когда оба $A1$ и $A2$ являются числовыми выражениями, имеющими одинаковое числовое значение.

«is» Отношение «X is A» выполняется, когда X — переменная, а A — числовое выражение. При этом ПРОЛОГ вычисляет значение A и унифицирует его с переменной X. Здесь мы обращаем особое внимание на различие между «is» и «=». Например, ответом ПРОЛОГа на запрос

? $X = 3 - 1$.

будет $X = 3 - 1$. Это свидетельствует о том, что при употреблении «=» арифметические операции в правой части равенства не выполнялись. Если же мы спросим

? X `is` $3 - 1$.

то получим ответ $X = 2$.

«\==» Предикат «\==» является отрицанием «==», т. е. отношение A \== B выполняется тогда и только тогда, когда $A == B$ не выполняется.

«=\=» Предикат «=\=» является отрицанием «=:=», т. е. отношение $A1$ =\= A2 выполняется тогда и только тогда, когда $A1$ =:= A2 не выполняется для числовых выражений $A1$ и $A2$.

3.5.4. *Арифметика в ПРОЛОГе*

Хотя ПРОЛОГ проектировался в основном для нужд символьного программирования, он содержит ряд функций, которые можно использовать для выполнения арифметических операций. В большинстве редакций ПРОЛОГа определены следующие функции:

+	сложение
−	вычитание

*	умножение
/	деление
div	деление нацело
mod	остаток от деления

Имеются также специальные встроенные предикаты сравнения числовых выражений

>	больше
<	меньше
=>	больше или равно
=<	меньше или равно

3.5.5. *Контроль типов*

«var» При помощи предиката `var(X)` можно проверить, верно ли, что X — это переменная, которая еще НИ РАЗУ НЕ получала никакого значения. Если терм X является переменной, ни разу не подвергавшейся изменениям, то предикат успешно выполняется. В противном случае выполнение предиката заканчивается неудачей.

«nonvar» Воспользовавшись предикатом `nonvar(X)`, мы можем проверить, верно ли, что объект X не является переменной.

«integer» Предикат `integer` успешно выполняется, если X является целым числом. Например, составной запрос

```
? integer(I), I is 3+5.
```

потерпит неудачу, тогда как запрос

```
I is 3+5, integer(I).
```

будет успешно выполнен и выдаст ответ $I=8$. Это произойдет потому, что к моменту выполнения `integer(I)` переменная I уже будет унифицирована с результатом сложения $2+3$.

«real» Предикат `real(X)` выполняется, если X — действительное число.

«atom» Предикат `atom(X)` выполняется, если X — атом.

«atomic» Предикат `atomic(X)` выполняется, если X — атом или число.

3.5.6. *Операторы*

Часто для упрощения набора текста программы и облегчения понимания сложных предикатов (см. § 3.3.5) мы записываем предикаты и функции в виде операторов. Так, например, для операции сложения мы могли бы использовать запись $+(x, y)$, где $+$ выступает в роли оператора.

Для того, чтобы полностью описать оператор, мы должны указать его приоритет, позицию, а также уточнить его связь с аргументами. Полное описание оператора осуществляется с помощью встроенного предиката op, который образует простой дизъюнкт вида

: – op(⟨ приоритет ⟩, ⟨ позиция ⟩, [список имен операторов])

Программные утверждения, описывающие операторы, располагаются в начале программы.

Приоритет операторов устанавливает порядок их выполнения в сложных выражениях, содержащих более одного оператора. Приоритет задается целым неотрицательным числом, область значения которого зависит от конкретной редакции языка. Например, в ТУРБО-ПРОЛОГе приоритет может быть любым числом от 1 до 2000. При прочих равных условиях операторы, приоритет которых ближе к 1, исполняются ранее операторов, приоритет которых ближе к верхней допустимой границе.

Например, выражение

8 + 2 * 2

мы предпочитаем воспринимать как

8 + (2 * 2)

чтобы получать в результате «12» , а не как (8 + 2) * 2, дающее в результате «20». Поэтому приоритет «*» должен иметь меньшее числовое значение, нежели приоритет «+».

Позиция оператора обозначает расположение оператора относительно его операндов. Имеется три варианта позиции: *в начале*, *между* и *после* операндов. Поэтому если мы хотим, чтобы предикат отец(a, b) употреблялся как оператор в смысле «a является отцом b», то этот предикат естественно поставить между его аргументами, т. е. в виде «a отец b». Здесь мы имеем случай *инфиксного* оператора, и его позиция обозначается выражением

xfx

В связи с этим, придавая предикату отец приоритет 200, мы должны описать его программным утверждением

: – op(200, *xfx*, отец).

Аналогично вводятся *префиксные* и *постфиксные* операторы. Таким образом, описывая оператор «¬» для отрицания фактов, разумно потребовать, чтобы он располагался перед операндом, т. е. чтобы он был префиксным оператором. Присваивая этому оператору приоритет 500, мы можем описать его программным утверждением

: – op(500, *fx*, ¬).

Если же мы хотим дать определение факториала, применяя символ «!», то было бы естественно обозначать факториал «x!», располагая «!» после операнда. Поэтому «!» будет постфиксным оператором.

Наделяя этот оператор приоритетом 400, мы можем описать его программным утверждением

```
: - op(400, xa, !).
```

Ассоциативность оператора указывает, будет ли этот оператор иметь больший, меньший или равный приоритет по сравнению с его операндами. Явное описание приоритета очень важно в тех случаях, когда не определен приоритет операндов. Например, арифметическое выражение

$$a - b - c$$

обозначает «$(a - b) - c$», а не «$a - (b - c)$».

Описание ассоциативности осуществляется путем отделения символов «x» и «y», обозначающих операнды. Так, выражения «fx» или «xf», содержащие переменную x, означают, что оператор имеет более низкий приоритет, чем его операнд. А выражения «fy» и «yf», содержащие переменную y означают, что приоритет оператора не менее высок, чем у его операнда. Конечно, мы вправе задавать различные комбинации приоритетов у двухместных операторов, как, например, в описании «xfy».

Предикаты для арифметических операций и различных видов равенств встроены в большинство редакций ПРОЛОГа и представлены в виде операторов, имеющих определенные описания приоритета, позиции и ассоциативности.

Примерами могут служить следующие описания.

```
: - op(500, yfx, [+, -]).                                    (1)
: - op(400, yfx, [*, /]).                                    (2)
: - op(500, xfx, [=, is, <, >, =<, =>, =\=, \==, =:=]).
```

В дизъюнкте (2) говорится о том, что «*», умножение, и «/», деление, являются операторами, имеющими два операнда и приоритет 400, тогда как в (1) утверждается, что сложение и вычитание имеют два операнда, и их приоритет 500. Поэтому в арифметических выражениях с «+ >>, «-», «*» и «/», сначала выполняются операции «*» и «/» ввиду их низкого приоритета, а затем — операции «+» и «-».

Если мы хотим определить логические связки «↔», «∨», «∧» и «¬» для записи формул логики предикатов и пропозициональной логики, то мы можем ввести соответствующие операторы.

```
: - op(800, yfx, ↔).
: - op(700, xfy, ∧).
: - op(600, xfy, ∨).
: - op(500, fy, ¬).
```

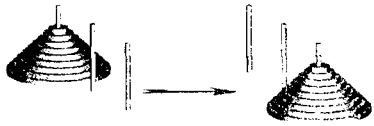
Тогда логическую эквивалентность

$$\neg(A \wedge B) \leftrightarrow \neg A \vee \neg B$$

можно представлять в ПРОЛОГе программным утверждением.

3.5.7. *Ханойские башни*

Игра «Ханойские башни» служит типичным примером рекурсивных описаний. Суть этой игры такова.



Имеются три колышка — левый, средний и правый, — и N дисков разного диаметра с отверстием посередине. Вначале все диски нанизаны на левый колышек так, что большие диски расположены внизу, а маленькие — вверху. Самый большой диск служит основанием всей башни. Задача состоит в том, чтобы перенести все диски на правый колышек, расположив их в том же самом порядке. При этом должны быть соблюдены следующие правила

а) всякий раз разрешается перемещать только один диск с одного колышка на другой,

б) запрещается класть диск большего размера на диск меньшего размера.

По всей видимости, нам нужно создать программу поиска пути из начального состояния в заключительное состояние, в которой были бы соблюдены указанные ограничения на перемещения дисков. Обратим внимание на следующие обстоятельства.

1). В заключительном состоянии на левом колышке не остается дисков.

2). Воспользовавшись правым колышком мы можем переместить $N-1$ диск с левого колышка на средний сообразно с правилами (a) и (b). Это будет один шаг нашей рекурсии. Затем оставшийся на левом колышке самый большой диск нужно перенести на правый колышек.

3). Воспользовавшись левым колышком, перенесем $N-1$ диск со среднего колышка на правый, соблюдая правила (a) и (b), и тем самым решим задачу.

Основываясь на этих наблюдениях, мы можем предложить следующую программу.

```
перенос(0,_,_,_) : - !.
перенос(N, X, Y, Z) : - M is N - 1,
                         перенос(M, X, Z, Y),
```

```
                напечатать_сообщение(X, Z),
                перенос(N1, Y, X, Z).
напечатать_сообщение(X, Y) :- write(«перенесли диск
                со стержня X на Y»).
ханойская_башня(N) :- перенос(N, левый, средний, правый).
```

В первых двух программных утверждениях фактически реализованы высказанные выше замечания. Третье программное утверждение распечатывает при помощи предиката `write` сообщения о перемещении дисков. Четвертый дизъюнкт обращается с вызовом к программе перемещения заданного числа дисков.

Обозреть пространство вычислений этой программы непросто даже для небольших значений N, поскольку в общем случае для решения задачи требуется совершить $2^N - 1$ перемещение дисков. Читателю предоставляется возможность воссоздать в качестве упражнения пространство вычислений программы для $N = 3$.

§ 3.6. Отрицание в ПРОЛОГе

Язык логики предикатов, используя содержащиеся в нем логические связки и кванторы, позволяет формализовать большинство разговорных фраз. Например, утверждение P разговорного языка

P: «Каждая канарейка, которая не больна, летает»

может быть символически представлено на языке логики предикатов предложением

$$P\text{: } (\forall x)[(\text{канарейка}(x) \wedge \neg\text{больна}(x)) \rightarrow \text{летает}(x)]$$

Здесь мы использовали логическую связку «$\neg$» для выражения отрицания предиката «больна». Однако язык ПРОЛОГ, основанный на хорновских дизъюнктах, не позволяет выражать отрицание явно, хотя может давать ответ **«нет»** на некоторые запросы. Поэтому формулировка, интерпретация и всякая обработка отрицаний дизъюнктов в ПРОЛОГе представляется непростым делом. Возникающие при этом проблемы можно разбить на три категории:

1) Как интерпретировать ответ **«нет»**?

2) Как в ПРОЛОГе выводится отрицание дизъюнкта?

3) Как следует применять отрицание в ПРОЛОГе?

3.6.1. *Допущение замкнутости мира и отрицание как неудача*

Согласно допущению замкнутости мира, сокращенно CWA (Closed World Assumption) (§ 3.1.3.), если предложение A ни явно, ни опосредованно не выразимо в программе P, иначе говоря,

если A не является ни фактом, ни следствием, вытекающим из данных P, то мы полагаем, что отрицание A справедливо. Рассмотрим следующую программу:

```
полный_квадрат(4).
полный_квадрат(9).
```

и обратимся к ней с запросом

```
? полный_квадрат(16).
```

Ответом ПРОЛОГа будет **«нет»**. Этот ответ ПРОЛОГа не следует трактовать как «16 не является полным квадратом», полагая, что «не» обозначает обычное для математической логики отрицание. Этот ответ **«нет»** подразумевает всего лишь, что у программы нет оснований считать число 16 полным квадратом. Такая «мыслительная» особенность ПРОЛОГа и логического программирования основывается на допущении замкнутости мира и называется *отрицание как неудача*.

На самом деле мы не можем выразить негативные знания посредством хорновских дизъюнктов: программа в логическом программировании по определению состоит исключительно из хорновских дизъюнктов, пригодных для представления только позитивных знаний, т. е. из фактов, правил и запросов, в которых отсутствует отрицание. Применяя правила резолюции к запросу Q, мы, по существу, доказываем, что $\{\text{факты, правила}, \neg Q\} \vdash \square$, т. е.

$$\{\text{факты, правила}\} \vdash \neg Q \rightarrow \square, \quad Q$$

Это означает, что в общем случае мы не можем доказать, что формула вида $\neg A$, где A не содержит отрицаний, следует из фактов и правил некоторой программы. Предпринимаются попытки распространить методы логического программирования на формулы более сложного вида, нежели хорновские дизъюнкты, для того, чтобы увеличить выразительные возможности логических программ [Shep88].

3.6.2. *Нормальные цели*

Рассмотрим программу P:

```
любит(джон, мэри).
любит(джон, стерлядь).
ест(X,Y) :- любит(X,Y), съедобна(Y).
съедобна(стерлядь).
```

Если мы спросим P верно ли, что Мэри съедобна (!)

```
? съедобна(мэри).                                    (*)
```

то ответом будет **«нет»**, т. е. ¬ **съедобна(мэри)**, вследствие CWA. Не лишено также смысла спросить P

```
? ¬ съедобна(мэри).                                  (**)
```

Чтобы ответить на этот запрос программа начнет проверять, верно ли, что цель

```
? съедобна(мэри).
```

может быть успешно достигнута. Поскольку «**съедобна(мэри).**» не согласуется ни с какими данными P, цель достигнута не будет. Поэтому ответом на запрос (*) будет «**да**».

Целевые утверждения вида «? $\neg A$.» называются *нормальными целями*, [Lloy87, NiMa95]. Процедура согласования, применяемая в ходе выполнения программы, позволяет дать следующее определение успеха и неудачи для нормальных целей:

Если цель «? A.» достигается успешно, то цель «? $\neg A$.» недостижима, т. е. A является следствием рассматриваемой программы.

Если все попытки достичь цель «? A.» оканчиваются неудачей, то цель «? $\neg A$.» успешно достигается, т. е. $\neg A$ является следствием рассматриваемой программы.

Таким образом, отрицание нормальных целей характеризуется неудачей или успехом достижения соответствующей цели без отрицания, и поэтому его называют *отрицание как неудача*, или сокращенно NF (negation by failure). Соответствующее (мета)правило имеет вид:

$$\frac{? \neg A., \quad ? A. \text{ fails}}{\square} \qquad \text{NF}$$

Это означает, что в случае неудачи «? A.» на программе P справедливо

$$\neg\neg A \rightarrow \square$$

т. е. $\neg A$ следует из P.

По всей видимости, нам придется иметь здесь дело с двумя видами отрицаний, одно из которых стандартная логическая связка, а другое имеет процедурный характер и обозначается предложением «? A.**fails**». Процедурное отрицание условимся обозначать $\sim$; тогда то обстоятельство, что ? A. оканчивается неудачей, может быть обозначено $\sim A$. Содержательно, «? $\neg A$.» можно считать эквивалентным A.

Правило NF разрешает выполнять резолюцию между ? $\neg A$., *(т. е.* A*) и* $\sim A$!

Такой вид резолюции называется NF-*резолюцией*.

Попытка дать теоретическое обоснование корректности NF правила и совместить, насколько это возможно, логическое и процедурное отрицания привела к понятию замыкания программ [Clar78, Lloy87, NiMa95]. Основная идея состоит в том, чтобы рассматривать предикаты, описанные в некоторой программе P посредством : –, т. е. $\leftarrow$, как *полностью* определенные при помощи связки $\leftrightarrow$.

3.6.3. *Замыкания программ*

Логическое основание для замыкания ПРОЛОГ-программ предоставляется замкнутыми при помощи кванторов всеобщности аксиомами (см. Замечание 2.3.7.)

(A_6) $x = x$ (аксиома тождественного равенства термов),

(A_7) $x = y \to (A \leftrightarrow A(y/x))$ (правило подстановки равных термов),

тождественно истинными формулами PrL

(1) $(B \to A) \wedge (C \to A) \leftrightarrow (B \vee C) \to A$,

(2) $(A(x, y) \to B(x, y)) \leftrightarrow [(\exists x)(\exists y)(A(x, y) \wedge x = a \wedge y = b) \to B(a, b)]$,

(3) $(A(a, b) \leftrightarrow (x = a \wedge y = b \to A(x, y))$,

а также некоторыми другими аксиомами, которые описывают свойства функций, встречающихся в программах. Условимся $\neg(x = y)$ записывать как $x \neq y$. Все функции, которые встречаются в программе, подвергающейся замыканию, [Clar78, Lloy87, NiMa95], должны удовлетворять следующим специальным, т. е. нелогическим, аксиомам, замкнутым при помощи кванторов всеобщности:

(F_1) $f(x_1, \ldots, x_n) \neq g(y_1, \ldots, y_m)$ для всех f, g таких, что $f \neq g$,

(F_2) $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \to (x_1 = y_1) \wedge \cdots \wedge (x_n = y_n)$,

(F_3) $f(x) \neq x$ для всех x собственных подтермов $f(x)$

Эти аксиомы означают, что различные функциональные символы именуют различные функции. Кроме того каждая функция принимает различные значения при различных наборах значений аргументов, и значение функции не может совпадать со значением одной из ее переменных, [Clar78].

Построим замыкание программы P, представленной в примере из предшествующего параграфа. Для того чтобы замыкание стало интуитивно более понятным, мы будем использовать $\leftarrow$ вместо : $-$.

Применяя формулы (2) и (3) к дизъюнктам P, мы получим следующую форму, эквивалентную P

$$\begin{aligned}
&\textbf{любит}(t_1, t_2) \leftarrow (t_1 = \textbf{джон} \wedge t_2 = \textbf{мэри}),\\
&\textbf{любит}(t_1, t_2) \leftarrow (t_1 = \textbf{джон} \wedge t_2 = \textbf{стерлядь}),\\
&\textbf{ест}(t_3, t_4) \leftarrow (\exists X)(\exists Y)(t_3 = X \wedge t_4 = Y \wedge \textbf{любит}(X, Y) \wedge\\
&\qquad\qquad \textbf{съедобна}(Y),\\
&\textbf{съедобна}(t_5) \leftarrow t_5 = \textbf{стерлядь},
\end{aligned}$$

которую в свою очередь, применяя формулу (1), приведем к виду

$$\begin{aligned}
&\textbf{любит}(t_1, t_2) \leftarrow (t_1 = \textbf{джон} \wedge t_2 = \textbf{мэри}) \wedge\\
&\qquad\qquad (t_1 = \textbf{джон} \wedge t_2 = \textbf{стерлядь}),\\
&\textbf{ест}(t_3, t_4) \leftarrow (\exists X)(\exists Y)(t_3 = X \wedge t_4 = Y \wedge\\
&\qquad\qquad \textbf{любит}(X, Y) \wedge \textbf{съедобна}(Y),\\
&\textbf{съедобна}(t_5) \leftarrow t_5 = \textbf{стерлядь}.
\end{aligned}$$

Теперь мы можем рассматривать выражения, стоящие слева от ← как *полностью* определенные предикаты «любит», «ест», «съедобно». Согласно CWA вся информация об этих предикатах, доступная нам в программе P, выражена формулами, стоящими справа от ←; следовательно, это и есть исчерпывающее определение указанных предикатов. Но ведь по существу это означает, что ← может восприниматься нами как ↔! Таким образом P должна иметь следующую *замкнутую* форму

$$\begin{aligned}
&\texttt{любит}(t_1, t_2) \leftrightarrow (t_1 = \texttt{джон} \wedge t_2 = \texttt{мэри}) \vee \\
&\qquad\qquad (t_1 = \texttt{джон} \wedge t_2 = \texttt{стерлядь}), \\
&\texttt{ест}(t_3, t_4) \leftrightarrow (\exists X)(\exists Y)(t_3 = X \wedge t_4 = Y \wedge \\
&\qquad\qquad \texttt{любит}(X, Y) \wedge \texttt{съедобна}(Y)), \\
&\texttt{съедобна}(t_5) \leftrightarrow t_5 = \texttt{стерлядь}.
\end{aligned}$$

Благодаря этому мы можем немедленно определить отрицание этих предикатов, например,

$$\neg\texttt{съедобна}(t_5) \leftrightarrow t_5 \neq \texttt{стерлядь}$$

Подобный способ введения отрицания запросов и нормальных целей вполне осмыслен. Более того, отрицание как неудача оказывается похожим на обычное логическое отрицание. Следует, тем не менее, иметь в виду, что интерпретация импликации как эквивалентности недопустима в рамках всей логики PrL и может быть корректно осуществлена только в контексте семантики логических программ.

Приведем формальное определение замыкания.

Определение 3.6.3.1.

1) Если предикат A стоит в заголовках хорновских дизъюнктов программы P

$$\begin{aligned}
&A(x_{1,1}, \ldots, x_{1,n}) \leftarrow W_1 \\
&\qquad\qquad \cdots\cdots\cdots\cdots \\
&\qquad\qquad \cdots\cdots\cdots\cdots \\
&A(x_{k,1}, \ldots, x_{k,n}) \leftarrow W_k
\end{aligned}$$

то *полное определение* A в P задается формулой

$$(\forall t_1)\cdots(\forall t_n)(A(t_1, \ldots, t_n) \leftrightarrow E_1 \vee \cdots \vee E_k),$$

в которой каждая подформула E_i имеет вид

$$(\exists y_1)\cdots(\exists y_d)(t_1 = x_{1,i} \wedge \cdots \wedge t_n = x_{n,i} \wedge W_i)$$

где $t_1, \ldots, t_n$ — новые переменные, не встречающиеся в P, $y_1, \ldots, y_d$ — все переменные из дизъюнкта $A(x_{i,1}, \ldots, x_{i,n}) \leftarrow W_i$.

2) Если предикат A не встречается в заголовках хорновских дизъюнктов программы P, то *полное определение* A в P задается формулой

$$(\forall t_1)\cdots(\forall t_n)\neg A(t_1, \ldots, t_n),$$

или, что то же самое, $(\forall t_1)\cdots(\forall t_n)A(t_1,\ldots,t_n)\leftrightarrow\square$.

3) *Замыкание* $compl(P)$ логической программы P образовано совокупностью полных определений всех предикатов, входящих в состав P. □ 3.6.3.1

Теоретическая выгода использования замыкания $compl(P)$ заключается в том, что мы, с одной стороны, усиливаем выразительные возможности программы P без утраты всех ее логических следствий, а с другой стороны, узакониваем правило NF. Это подтверждается следующими теоремами, доказательство которых можно найти, например, в [NiMa95, Lloy87].

Теорема 3.6.3.2. *Если* $P\models A$, *то* $compl(P)\models A$. □ 3.6.3.2

Теорема 3.6.3.3. *Правило* NF *корректно для* $compl(P)$. □ 3.6.3.3

В общем случае вычисления в логическом программировании не позволяют нам отыскивать значения x, удовлетворяющие запросам вида ? $\neg A$., и чаще всего подразумевается, что правило NF применяется только к основным примерам дизъюнктов [NiMa]. Если нормальная цель «? $\neg A$» содержит только основные примеры атомов, то соответствующее отрицание называется *безопасным*, в противном случае оно называется *небезопасным*. Многие языки логического программирования допускают использование только безопасных отрицаний; это обеспечивается встроенным механизмом задержки обработки нормальных целей до тех пор, пока все вхождения переменных не получат конкретных значений.

3.6.4. *Нормальные программы и стратификация*

Итак, целевые запросы с отрицательными предикатами допустимы в логическом программировании и ПРОЛОГе. Если мы к тому же разрешим использовать отрицание в теле правил, то подобные программы называются *нормальными* или *обобщенными программами*, [NiMa95, Lloy87]. Здесь, однако, положение дел более сложное: нормальная программа P может содержать дизъюнкт вида

$$A:-\neg A.$$

т. е. $A\leftarrow\neg A$. Этот дизъюнкт эквивалентен A, в то время как в $compl(P)$ он будет приведен к виду

$$A\leftrightarrow\neg A$$

эквивалентному $A \wedge \neg A$! В таком случае $compl(P)$ становится противоречивым множеством формул.

Вместе с тем нормальные программы, обладающие *расслоенностью* (*стратифицированные* программы), имеют непротиворечивое замыкание. Рассмотрим, например, следующую программу P:

$A : - \neg B.$

$A : - B, C.$

$C.$

Тогда $compl(P) = \{A \leftrightarrow \neg B \vee C\}$ образует, очевидно, непротиворечивое множество формул.

Поясним теперь, что мы подразумеваем под расслоенностью программы. Заметим, что в программе P предикат A определяется формулами $\neg B$ и $B \wedge C$, т. е. зависит от B и C. Таким образом, мы могли бы ввести иерархию уровней описания предикатов так, чтобы предикаты более высокого уровня определялись посредством предикатов лежащих ниже уровней. Конечно, предикат A должен быть отнесен к «наивысшему уровню описания», поскольку A зависит от B и C. В свою очередь, B следует отнести к более низкому, чем A, «уровню описания», так как значение B должно быть известно заранее к тому моменту, когда нужно определять значение A.

Ввиду того, что «уровни описания» должны позволить полностью определять значения предикатов в высоких слоях программы, все эти уровни не должны иметь общих элементов. На самом деле, нам следует уточнить очерченный здесь подход, разрешив относить предикат X к тому же уровню, что и предикат Y, участвующий в описании X, при условии, что именно X, а не его отрицание, используется для определения Y. В нашем примере предикаты, встречающиеся в P, могут быть разбиты на два «уровня описания» $S_0 = \{B\}$ и $S_1 = \{A, C\}$ так, чтобы отрицание предиката B, участвующее в описании предиката A, было определено на более низком уровне, нежели описание A. Поскольку сам предикат C, а не его отрицание, фигурирует в описании A, уровень S_1 может содержать как A, так и C.

Мы можем рассчитывать, что применение данного подхода увенчается успехом, поскольку расслоение призвано устранить нежелательные последствия рекурсии. В общем случае, если отрицание предиката D фигурирует справа от : – в описании предиката F, то D должен быть определен на более низком уровне во избежание рекурсивных обращений, возникающих при откате и приводящих к формулам вида $D : - \neg D$. Возможность возникновения такой нежелательной ситуации видна на примере следующей программы:

$A : - B.$

$B : - \neg A$

Приведем формальное определение (см [NiMa95, Lloy97]).

Определение 3.6.4.1. Нормальная программа P называется *стратифицированной*, если множество предикатов, входящих в P, может быть разбито на подмножества $S_0, \ldots, S_n$ так, чтобы для всякого правила $A : - B_1, \ldots, B_m$ из P, заголовок которого $A \in S_k$, и для каждого B_i, $1 \leqslant i \leqslant m$, либо

$$B_i \in S_0 \cup S_1 \cup \cdots \cup S_k$$

в случае, когда B_i не содержит отрицания, либо

$$C_i \in S_0 \cup S_1 \cup \cdots \cup S_{k-1}$$

в случае, когда B_i имеет вид $\neg C_i$. □ 3.6.4.1

Теорема 3.6.4.2. *Если P — стратифицированная программа, то $compl(P)$ — непротиворечиво.* □ 3.6.4.2

Теорема 3.6.4.3. *Если P — нормальная программа, то правило* NF *корректно для $compl(P)$.* □ 3.6.4.3

Исследование проблемы полноты, касающееся NF, выходит за рамки нашей книги. Заинтересованный читатель может обратиться к [Lloy87, NiMa95].

Рассмотрим теперь пример применения правила NF к нормальной программе, т. е. к программе, в которой могут фигурировать отрицательные знания. Предположим, что мы имеем дело с программой P, в которой содержится обращение к отрицательным знаниям

```
заинтересован(X, Y) : - ¬ ненавидит(X, Y).   /* 1 */
ненавидит(X, Y) : - ¬ любит(X, Y).           /* 2 */
любит(джон,мэри).                            /* 3 */
```

и мы хотим узнать, насколько Мэри интересна Джону,

```
? заинтересован(джон,мэри).                  /* 4 */
```

Выполнение программы происходит следующим образом. Сначала запрос согласуется с левой частью первого правила программы путем применения подстановки $X =$ **джон**, $Y =$ **мэри**. В результате образуется запрос

```
? ¬ненавидит(джон,мэри).                     /* 5 */
```

Далее алгоритм проверяет целевое **утверждение**

```
? ненавидит(джон,мэри).                      /* 6 */
```

Этот запрос согласуется с заголовком **второго правила** программы путем применения подстановки $X =$ **джон**, $Y =$ **мэри**. За счет этого формируется следующий запрос

```
? ¬любит(джон,мэри).                         /* 7 */
```

Алгоритм проверяет целевое утверждение

```
? любит(джон,мэри).
```

и обнаруживает, что оно успешно подтверждается за счет третьего программного утверждения. По правилу NF цель /* 7 */ не достигается. Значит цель /* 6 */ также не достигается. Следовательно, цель /* 5 */ успешно достижима, и окончательным ответом на запрос /* 4 */ будет «да» !

С другой стороны, если мы попытаемся применить резолютивный метод к соответствующим логическим предложениям, мы не сможем столь же легко добиться результата, потому что P содержит не только хорновские дизъюнкты. Рассмотрим следующую эквивалентное представление P

заинтересован(X, Y) $\vee$ **ненавидит**(X, Y),

ненавидит(X, Y) $\vee$ **любит**(X, Y),

любит(джон,мэри),

$\neg$**заинтересован(джон,мэри)**.

Здесь мы можем провести резолюцию только между первым и последним дизъюнктом, получая в качестве единственного результата «**ненавидит(джон,мэри)**». Напротив, применяя метаправило NF, мы можем найти приемлемое решение, которое будет корректным в свете предположения о том, что все данные, полностью описывающие предикаты программы P, явно представлены в P !

3.6.5. *Предикат fail*

Заданная цель в ПРОЛОГе терпит неудачу, то есть принимает истинностное значение «ложь», если попытка найти для нее положительное подтверждение с использованием всех возможных откатов оканчивается неудачей. Неудача, влекущая за собой отрицательный ответ, может быть объявлена в ПРОЛОГе при помощи специального встроенного предиката под названием `fail`, который обозначает эту *неудачу*, имеет постоянное истинностное значение «ложь» и всегда завершается неудачей. Таким образом

$$\texttt{fail} \leftrightarrow Q \wedge \neg Q$$

где Q — произвольная формула логики предикатов.

Возьмем, например предложение

A: «Джон - не канарейка»

Предложение A можно представить в ПРОЛОГе правилом

канарейка(джон) : – `fail`. (*)

означающим, что «цель **канарейка(джон)** терпит неудачу». Предположим, что база данных ПРОЛОГ-программы содержит только правило (*). Тогда запрос

? **канарейка(джон)**.

обязательно потерпит неудачу. После унификации этого запроса с правилом (*) будет образована новая подцель `fail`, которая потерпит неудачу по определению `fail`. Такова операционная интерпретация `fail`: запрос, который состоит из заголовка правила, содержащего в своем теле `fail`, терпит неудачу[1]).

Мы должны обратить особое внимание на то, что дизъюнкт вида

$$A \text{ :- } \texttt{fail}.$$

является тождественно истинным в логике предикатов, поскольку формула $Q \wedge \neg Q \rightarrow A$ остается истинной во всякой интерпретации логики предикатов (см. определения 2.5.5, 2.5.19. и доказательство следствия 1.5.4).

Отрицание в ПРОЛОГе, вообще говоря, можно представить при помощи подходящей комбинации операторов отсечения и неудачи . Возьмем, например, дизъюнкт

$$P\colon (\forall x)[(\textit{канарейка}(x) \wedge \neg \textit{больна}(x)) \rightarrow \textit{летает}(x)],$$

рассмотренный в начале этой параграфа. Предположим, что мы хотим определить отношение между летающими предметами и канарейками. В нашем случае мы имеем дело с канарейками, которые летают, равно как и с канарейками, которые не летают, потому что они, скажем, были покалечены. Поэтому в нашей программе должны быть учтены обе возможности: канарейки, неспособные летать вследствие их болезни, и канарейки, умеющие летать. Эти возможности отражены в следующей ПРОЛОГ программе:

```
летает(X) :- больна(X), !, fail.        /* 1 */
летает(X) :- канарейка(X).              /* 2 */
```

В этой программе отсечение, !, применяется во избежание активизации второго правила в случае, когда наша канарейка X больна. Если канарейка X больна, то отсечение предотвратит откат ПРОЛОГа ко второму правилу, и `fail` приводит заголовок первого правила к неудаче. Если же канарейка не больна, или в базе данных нет никаких упоминаний о том, что она больна, то первое правило не сработает, и ПРОЛОГ совершит откат (отсечения в этом случае не происходит), активизируя второе правило, заголовок которого будет выполнен.

Мы видим, таким образом, что отрицание в ПРОЛОГе может быть представлено *комбинацией отсечения и неудачи.*

3.6.6. *Предикат not*

Существует и другой подход, позволяющий ввести отрицание в ПРОЛОГе посредством специального предиката `not`(Q) (§ 3.3.5.), который обозначает отрицание Q и выполним всякий раз, когда

[1]) Это справедливо при отсутствии в программе иных альтернативных правил с тем же заголовком! — *Прим. перев.*

цель Q терпит неудачу. В сущности, предикат not моделирует выполнение комбинации отсечения и неудачи, и описывается следующими дизъюнктами:

```
not(Q) : - Q, !, fail.                    /* 3 */
not(Q).                                   /* 4 */
```

Поэтому, если мы обратимся с запросом

```
? not(Q).
```

то ПРОЛОГ активизирует правило /* 3 */. Если предикат Q явно не описан в базе данных и не следует из данных логической программы, то подцель Q в теле правила приведет к неудаче[1]). Тогда ПРОЛОГ попытается удовлетворить not(Q), применяя дизъюнкт /* 4 */. Поскольку Q приводит к неудаче, not(Q)[2]) выполняется согласно метаправилу NF, и ПРОЛОГ ответит «да» в знак того, что справедливо отрицание Q. Если, в свою очередь, предикат Q либо явно описан в программе, либо выводится из ее данных, то подцели Q, *!* и fail в правиле /* 3 */ будут выполнены. Отсечение, *!*, предотвращает обращение к дизъюнкту /* 4 */, а fail означает, что цель not(Q) не достигается. Поэтому заголовок правила /* 3 */, not(Q), приводит к неудаче, и ПРОЛОГ выдаст ответ «нет», означающий, что предикат Q выполним, тогда как not(Q) — нет.

Применение not существенно облегчает представление отрицания в программах. В примере из предыдущего параграфа, дизъюнкты /* 1 */ и /* 2 */ могут быть замещены правилом

```
летает(X) : - канарейка(X), not(больна(X)). /* 1' */
```

Предикат not встроен в большинство версий ПРОЛОГа. В тех случаях, когда он отсутствует, мы можем описать not, используя программные утверждения /* 3 */ и /* 4 */. Напомним еще раз, что ПРОЛОГ позволяет определять предикаты на множестве конструкций языка в терминах самого языка (ПРОЛОГ средствами самого ПРОЛОГа)[3]).

3.6.7. *Немонотонные логики*

Рассуждения, основанные на CWA, имеют один внутренний изъян. Рассмотрим, например, одноэлементное множество

$$S = \{\sigma \vee \varphi\},$$

где σ и φ — дизъюнкты логики предикатов. Если мы запишем $S \models$ $\models_{CWA} A$ в знак того, что A логически следует из S при допущении замкнутости мира, то справедливо

---

[1]) При условии,что поиск подтверждения для Q с использованием всех возможных откатов завершится неудачей за конечное(!) число шагов. — *Прим. перев.*

[2]) Точнее, $\sim Q$. — *Прим. перев.*

[3]) Такие средства языка называются *метасредствами*. — *Прим. перев.*

$$S \models_{CWA} \neg\sigma \quad \text{и} \quad S \models_{CWA} \neg\varphi,$$

поскольку σ и φ явно не описаны и не выводимы из данных множества S. Поэтому следствия S в случае использования CWA образуют множество

$$Con_{CWA}(S) = \{\sigma \vee \varphi, \neg\sigma, \neg\varphi\},$$

которое, очевидно, противоречиво (применяя два раза правило резолюции, мы можем вывести $\square$). Таким образом, применение допущения замкнутости мира приводит к противоречию даже в очень простых случаях.

Другая проблема возникает в связи с использованием CWA по отношению к общим правилам программы и их исключениям. Например, согласно правилу /* 1′ */ из п. 3.6.6, для того, чтобы прийти к выводу о том, что некое существо летает, мы должны исключить возможность того, что оно больно. Это означает, что мы должны явно описать все возможные исключения для правила /* 1′ */. Однако, в общем случае добиться этого невозможно, так как некоторые исключения могут быть неизвестны заранее, но будут уточняться в ходе выполнения программы.

В последние годы были предприняты попытки развития формальных методов дедуктивных умозаключений, избавленных от проблем подобных тем, которые возникают в случае CWA. Проводимые исследования направлены на развитие новых логик, получивших название *немонотонные логики*. Основное различие между немонотонными и классическими логиками высказываний и предикатов состоит в том, что в немонотонных логиках мы можем отменять ранее сделанные заключения по мере того, как выводятся новые. В обычных классических логиках

$$S \models A \text{ влечет за собой } S \cup S' \not\models A.$$

Это означает, что дизъюнкт A, являющийся следствием множества дизъюнктов S, будет следовать также и из всякого расширения множества S. В немонотонных логиках (сокращенно, NM)

$$\text{Если } S \models_{\mathbf{NM}} A, \text{ то, вообще говоря, } S \cup S' \not\models_{\mathbf{NM}} A.$$

Это означает, что в немонотонных логиках дизъюнкт A, будучи следствием S, в общем случае не обязан оставаться следствием множества дизъюнктов $S \cup S'$, объемлющего S.

Отмену ранее сделанных умозаключений по мере расширения данных можно уподобить естественному образу мышления людей, т. е. пересмотру их убеждений по мере поступления дополнительных сведений об окружающем мире. Исследователи немонотонных логик стремятся придать новое обоснование языкам логического программирования типа ПРОЛОГ, рассчитывая создать более

«умные» и более производительные системы программирования. В немонотонных логиках предметная область считается незамкнутой.

Предметная область не является ни статичной, ни окончательно сформированной; вместо этого она постоянно расширяется по мере поступления новых данных и информации. Поэтому понятие истины в предметной области всегда относительно и зависит от тех знаний, которые представлены в нашем универсуме в каждый конкретный момент времени.

Механизм вывода в немонотонных логиках основан на допущении замкнутости мира в каждый момент времени.

Мы считаем, что в момент времени t_1 предметная область может быть описана множеством дизъюнктов S_1, тогда как в момент времени t_2, где $t_2 > t_1$, она может быть описана множеством дизъюнктов S_2, причем $S_2 \neq S_1$.

Для того, чтобы найти следствия S_1 и S_2, или, иначе говоря, для получения ответа на вопрос, относящийся к S_1 или S_2, мы полагаем, что допущение замкнутости мира сохраняется как в S_1, так и в S_2. Это означает, что в момент времени t_1 универсум замкнут и описан множеством дизъюнктов S_1, а в момент времени t_2 универсум также замкнут, но описан множеством S_2.

Дальнейшие сведения о немонотонных логиках можно отыскать, например, в [Turn84] или [Besn89].

§ 3.7. Экспертные системы

3.7.1. *Искусственный интеллект*

Раздел математики, носящий название *теория искусственного интеллекта* (сокращенно AI), занимается проблемами разработки и формализации интеллектуальных методов и процедур решения различных задач. Прилагательное «интеллектуальный» означает здесь способность привлекать методику рассуждений и оценок, свойственную рациональной рассудочной деятельности человека. Исследования в AI направлены на разработку символьных описаний окружающей действительности, и охватывают такие области, как роботехника, понимание естественного языка, экспертные системы.

3.7.2. *Экспертные системы и обработка знаний*

Экспертными системами называются вычислительные программы, предназначенные для решения сложных задач, требующих обширных знаний и опыта [BuSc84].

Коренное отличие экспертных систем от обычных программ состоит в том, что традиционные программы заняты обработкой данных,

в то время как экспертные системы обрабатывают знания. Различия между этими двумя разновидностями программирования представлены в следующей таблице

Обработка данных	Обработка знаний
Представление и использование данных Алгоритмические процедуры Итеративные процедуры	Представление и использование знаний Поисковые процедуры Процедуры вывода

Экспертные системы — это специализированные системы, ориентированные на применение в отдельных узких областях человеческой деятельности; к их числу относятся системы медицинской диагностики, системы контроля механических неисправностей, системы синтеза сложных химических соединений и др.

Для решения поставленных задач экспертные системы используют накопленные *факты* и наборы *эвристик*, т. е. правил обработки знаний. Эти факты и правила формируются специалистами той конкретной области деятельности, для которой проектируется экспертная система. Можно сказать, что упомянутая совокупность фактов и эвристик представляет знания системы в специальной области ее применения, и поэтому экспертные системы часто называют *системами баз знаний.*

Экспертная система состоит из *базы данных*, *аппарата вывода* и *интерфейса*, обеспечивающего взаимодействие системы и пользователя.

База данных представляет собой совокупность элементарных состояний и фактов, описывающих предметную область интересующего нас направления деятельности.

Представление знаний [Watt90] в системах, основанных на правилах, достигается за счет использования правил вида

IF ⟨ условие_1 ⟩ and . ⟨ условие_k ⟩ THEN ⟨ действие ⟩

Аппарат вывода представлен множеством целевых правил общего вида, предназначенных для организации и управления процессом вывода. В этом аппарате используются математические модели, имеющие широкую область применения. С их помощью описываются и формализуются общие процедуры обработки и разрешения задач, такие как поиск в глубину с возвратом, откат и другие универсальные методы вывода, представленные в предыдущих разделах.

В случае, когда механизм вывода не зависит от тех специальных знаний, которые содержатся в данной экспертной системе, и работает автономно, его можно использовать при разработке новой экспертной системы. Подобные универсальные системы называются *оболочками.*

Интерфейс обеспечивает взаимосвязь пользователя и системы. Для того, чтобы эта взаимосвязь была как можно более естественной и удобной для пользователя, в большинстве случаев применяются специальные подсистемы анализа и синтеза конструкций естественного языка (системы лексического и синтаксического разбора).

Отбор и формализация знаний, которые должны быть заложены в проектируемую экспертную систему, является основополагающим и наиболее трудным этапом разработки всей системы. Накопление знаний — это узкое место, «бутылочное горлышко», технологии проектирования экспертных систем. Трудность заключается в отыскании и одновременной формализации специальных правил, которыми руководствуется специалист в конкретной области для принятия решения. Поэтому значительный вклад в успешную разработку и последующую эксплуатацию экспертной системы вносит *инженер базы знаний*, лицо, ответственное за отбор, накопление и формализацию знаний.

В связи с особой значимостью этапа накопления знаний в последние годы были разработаны *автоматические системы накопления знаний.*

Использованные при этом методы были во многих случаях выбраны чрезвычайно удачно и вызвали большой теоретический интерес [OlRu87, Quin86].

Инструментарий, применяемый для разработки экспертных систем, может быть разделен на две категории [WeKu84]: *высокоуровневые языки символьной обработки* и *экспертные системы широкого профиля*, или *оболочки.* К числу наиболее распространенных высокоуровневых языков обработки символьных данных относятся ЛИСП [WiBe89] и ПРОЛОГ.

Оболочки используются для облегчения и ускорения процесса разработки специализированных экспертных систем. С помощью специального интерфейса, дружественного по отношению к пользователю, а также структур, образующих оболочку, можно упростить процедуру ввода в экспертную систему фактов и правил и организовать в случае необходимости их пересмотр. Помимо того оболочки включают в себя автономный аппарат построения вывода, который может осуществлять совместную обработку информации и проводить вывод заключений из нескольких различных баз данных. Все это дает возможность быстро разрабатывать прототипы экспертных систем.

3.7.3. *Экспертная система для диагностики почечных заболеваний*

Экспертная система для диагностики почечных заболеваний, Kidney Expert System (KES), была разработана совместными усилиями группы специалистов по логике и логическому программирова-

нию факультета математики университета Патраса и специалистов-врачей госпиталя университета Патраса.

При описании этой системы мы ограничимся рассмотрением только тех дизъюнктов и процедур, которые определяют ее устройство, функционирование и наглядно демонстрируют применение ПРОЛОГа при разработке экспертных систем.

Правила, образующие специализированную базу знаний KES, формализуют приемы, которыми руководствуется врач, диагностирующий заболевание по его симптомам и показаниям пациента. Он принимает, например, во внимание тот факт, что пациент немолод, что в его моче, равно как и в крови, наблюдаются качественные изменения, что почки его увеличены, и т. п. Если все перечисленные выше признаки налицо, то пациент, вероятно, страдает гидронефрозом. Поэтому правила KES имеют следующий общий вид

```
IF              общие сведения о пациенте по стандартной форме
   and          жалобы(X)
      and/or    наблюдение(симптом_1)
      and/or    ........
      ..........
      and/or    наблюдение(симптом_k)
      and/or    исследование(лабораторный анализ_1)
      ..........
      and/or    исследование(лабораторный анализ_n)
THEN              диагноз — (заболевание) с вероятностью P
```

Эти правила были выработаны под руководством медицинских специалистов. Например, для гидронефроза соответствующее правило выглядит так

```
диагноз(гидронефроз, 0.7): -возраст(H), H > 11,
          жалобы(качественные_изменения_мочи),
          наблюдение(кровотечение_при_мочеиспускании),
          наблюдение(двухстороннее_расширение_почки).
```

где 0.7 — вероятность того, что проходящий обследование пациент страдает гидронефрозом. Теоретическая оценка вероятности, сопровождающая каждый диагноз, имеет априорный статистический характер. За счет этого допускается диагностирование одного и того же заболевания различными путями с разной вероятностью. Базовые предикаты, фигурирующие в KES, таковы

данные: Имеет четыре аргумента: пол, возраст, имя, фамилия пациента.

наблюдение: Имеет один аргумент — название симптома, наблюдаемого при обследовании.

исследование: Имеет один аргумент — результат лабораторного анализа.

жалобы: Имеет в качестве аргумента одно из следующих показаний: боль, качественные_изменение_мочи, количественные_изменения_мочи, изменение_частоты_мочеиспускания.

Предикат **жалобы** используется для группирования симптомов. Поэтому правило общего вида, не содержащее в теле предиката **жалобы** с перечисленными аргументами, не будет активизировано, что отразится на быстродействии процедуры вывода.

диагноз: Имеет два аргумента: наименование диагноза и вероятность того, что пациент может страдать установленным заболеванием.

Завершив описание базовых предикатов и специальных правил экспертной системы, мы можем перейти к описанию процедур и соответствующих программных дизъюнктов, запускающих аппарат вывода в нашей системе.

Вначале мы определим процедуру ввода данных о пациенте, или, другими словами, предикат **данные**

```
данные(S, A, L, F) : – write("Пол пациента (м/ж)"),
                        read(S), nl,
                        write("Возраст пациента"), read(A), nl,
                        write("Фамилия пациента"), read(L), nl,
                        write("Имя пациента"), read(F), nl,
                        assertz(пол(S)),
                        assertz(возраст(A)),
                        assertz(фамилия(L)),
                        assertz(имя(F)).
```

Затем определим процедуру ввода показаний пациента, иначе говоря, предикат **жалобы**

```
жалобы(X) : – write("Показания пациента",X,"(да/нет)?"),
                        read(Ответ), nl,
                        Ответ="да",
                        assertz(жалоба(X)).
```

Теперь перейдем к процедурам ввода симптомов, наблюдаемых при осмотре пациента, и результатов лабораторных исследований, имеющихся в распоряжении у врача

```
наблюдение(S): –write("Обнаружено",S,"(да/нет)?"),
                        read(Ответ), nl,
                        Ответ="да",
                        assertz(наблюдение(S)).

исследование(R): –
                        write("Результат анализа",R, "(да/нет)?"),
                        read(Ответ), nl,
                        Ответ="да",
                        assertz( исследование(R)).
```

В приведенных выше описаниях оператор **assertz** заносит в базу данных заголовок соответствующего правила вместе с подстановочным вариантом его аргументов. За счет этого информация, полученная в ходе диалога системы и пользователя, помещается в память и сохраняется для дальнейшей обработки.

И в завершение мы опишем главный предикат **поставить_диагноз**.

```
поставить_диагноз : – !, данные(S, A, L, F),
                       диагноз(D, P),
                       assertz(диагноз(D, P)).
```

Если описанный предикат поставить в качестве запроса и обеспечить базу данных нужными эвристиками, то ПРОЛОГ приступит к выводу заключений. В действительности, возникнет диалог между пользователем и ПРОЛОГом, который завершится постановкой тех или иных диагнозов в зависимости от сведений, представленных пользователем.

Предположим, например, что база данных содержит следующие правила диагностики.

```
диагноз(острый_пиелонефрит,0.7) : –
        жалобы(боль),
        наблюдение(лихорадочный_озноб),
        наблюдение(кровотечение_при_мочеиспускании).
диагноз(острый_пиелонефрит,0.8) : –
        возраст(A), A < 9,
        диагноз(острый_пиелонефрит,0.7),
        исследование(положит._анализ_культуры_мочи).

диагноз(острый_пиелонефрит,0.9) : –
        жалобы(изменение_частоты_мочеиспускания),
        жалобы(боль),
        диагноз(острый_пиелонефрит,0.8),
        наблюдение(боли_в_почках),
        наблюдение(миктоурит),
        наблюдение(дизурия).
```

Обратимся с запросом

```
? поставить_диагноз.
```

Завяжется следующий диалог между пользователем и экспертной системой (ответы пользователя взяты в кавычки, а строки, содержащие комментарий, отмечены значком %).

```
? Пол пациента (м/ж)    «м»
? Возраст пациента      «11»
? Фамилия пациента      «имярек»
? Имя пациента          «имярек»
%       ПРОЛОГ вводит в базу данных следующие дизъюнкты
%       пол(м).
```

```
%       возраст(11).
%       фамилия(имярек).
%       имя(имярек).
  ? Показания пациента   боль   (да/нет)?   «да»
%       ПРОЛОГ заносит в базу данных дизъюнкт
%       жалоба(боль).
  ? Обнаружено при обследовании   пиурия   (да/нет)?   «да»
%       ПРОЛОГ заносит в базу данных дизъюнкты
%       наблюдение(пиурия).
%       диагноз(острый_пиелонефрит, 0.7).
%       и пытается найти другое подтверждение
%       для предиката диагноз.
%       Поскольку во втором программном утверждении
%       с заголовком диагноз
%       упоминается возраст(A),
%       а в нашем случае A = 11, то предикат A < 9
%       будет невыполним, и ПРОЛОГ перейдет
%       к третьему программному утверждению,
%       утверждению, описывающему предикат диагноз.
  ? Показания пациента
    изменение_частоты_мочеиспускания   (да/нет)?   «нет»
%       ПРОЛОГ отвергает последнее правило,
%       прекращает, учитывая оператор отсечения, поиски ответа,
%       на первоначальный запрос, и выдает окончательный диаг-
%       ноз
  D = острый_пиелонефрит
  P = 0.7
```

§ 3.8. Эволюция логического программирования

3.8.1. *Редакции ПРОЛОГа*

ПРОЛОГ в первоначальной редакции ПРОЛОГ I был спроектирован как вспомогательный искусственный язык для обработки естественно-языковых конструкций на основе логики [Colm90, Col90a]. И хотя этот вариант языка оказался очень эффективным средством для представления и решения сложных задач, его способности выполнять арифметические вычисления были весьма ограничены.

ПРОЛОГ II, та редакция ПРОЛОГа, которая рассматривается в нашей книге, обладает достаточной гибкостью и выразительностью для проведения арифметических вычислений, благодаря использованию аппарата бесконечных деревьев и предиката «\ =».

Последняя редакция ПРОЛОГа, ПРОЛОГ III [Colm90, Col90a] обладает еще более развитыми возможностями, включающими встроенный

аппарат обработки бесконечных деревьев, булеву алгебру, предикаты

$$<,\ =,\ <,\ >,\ =>,\ \text{и}\ \backslash=,$$

а также сложение, вычитание и умножение функций и констант.

ПРОЛОГ III относится к числу так называемых языков логического программирования с ограничениями (constraint logic programming) [Cohe90]. Логическое программирование с ограничениями сформировалось в результате попыток обогатить язык хорновских дизъюнктов переменными, принимающими значения из различных областей, как то множество деревьев, действительных чисел, булева алгебра и т. п.[1])

3.8.2. *Версии ПРОЛОГа*

ПРОЛОГ в сущности предлагает лишь аппарат для выполнения логических программ. Разнообразные версии ПРОЛОГа, имея в своей основе один и тот же аппарат логического вывода, различаются, по большей части, в стиле общения с пользователем.

МИКРО-ПРОЛОГ [ClMc84], ТУРБО-ПРОЛОГ, АРИТИ-ПРОЛОГ, Δ-ПРОЛОГ [Xant90] предназначены для малых компьютеров. Система Δ-ПРОЛОГ является греческой версией и допускает использование греческих букв.

IC-ПРОЛОГ, mu-ПРОЛОГ и nu-ПРОЛОГ имеют более сложные правила выбора предикатов для унификации [Lloy87]. Nu-ПРОЛОГ, обладающий автоматическим препроцессором, допускает гораздо более чуткое управление программой по сравнению с другими версиями ПРОЛОГа и считается языком наиболее близким к языку «идеального» логического программирования.

3.8.3. *ПРОЛОГ и метапрограммирование*

В разделах 1 и 2 мы отмечали, что для языка $\mathcal{L}$ логики предикатов или логики высказываний имеется соответствующий метаязык, т. е. язык, на котором описываются свойства языка $\mathcal{L}$ и его дизъюнктов, см. Замечание 1.5.2. Подобной возможностью располагает и ПРОЛОГ. Метапрограммы в ПРОЛОГе воспринимают программы ПРОЛОГа как термы и анализируют их свойства и взаимоотношения. Например, в [Kowa90] предикату

`demo`(T, P)

придается следующий смысл

«программа T применяется для доказательства дизъюнкта P».

[1]) Помимо типизации переменных в логическом программировании с ограничениями широко используются нелогические средства разрешения встроенных предикатов, налагающих *ограничения* на допустимые значения переменных. — *Прим. перев.*

За счет способности ПРОЛОГа к метапрограммированию пользователь, имея вначале простые программы и применяя по мере необходимости средства метаязыка, может создать ПРОЛОГ-программу, управляющую целой группой ПРОЛОГ-программ, которые, в свою очередь, обращаются к другим ПРОЛОГ-программам, и т. д.

Метапрограммирование применяется с целью автоматизации разработки ПРОЛОГ-программ и автоматической проверки их свойств.

3.8.4. *Пролог и параллелизм*

Поиски путей повышения быстродействия и эффективности вычислительных машин привели к созданию параллельных компьютеров. Классический компьютер обрабатывает каждую команду или оператор программы последовательно по очереди, в результате чего машина способна исполнять только по одной команде или оператору на каждом шаге вычисления. В параллельных компьютерах на одном шаге вычисления привлекаются несколько независимых друг от друга команд или операторов, которые исполняются параллельно на многопроцессорном вычислительном устройстве.

Хотя применяя параллельные алгоритмы и параллельные компьютеры можно получить простые и естественные решения сложных задач, при этом приходится сталкиваться с новыми проблемами, связанными с синхронизацией обмена сообщениями между независимыми процессами и управлением всей программой. Эти трудности можно преодолеть обратившись к специальным языкам программирования, обладающим способностью выражать параллелизм вычислений (например, к языку ОККАМ) и восполняющим разрыв между новой многопроцессорной аппаратной технологией и классическим последовательным программным обеспечением.

В самом ПРОЛОГе нет средств для использования параллелизма вычислений. Более того, выполнение ПРОЛОГ-программы, как мы уже отмечали ранее, представляет собой последовательный процесс. Однако, в последние годы были разработаны и выдвинуты на первый план языки параллельного логического программирования. В основу этих языков положен принцип параллельной интерпретации логических программ, который обеспечивается специальными средствами синхронизации процессов и управления всей программой.

Реляционный язык, предложенный Кларком и Грегори в 1981 г. был первым языком логического программирования, который допускал возможность недетерминированного выбора независимых программных утверждений. Утверждения этого языка могут быть представлены в следующем виде

$$A : - G_1, \ldots, G_k / B_1, \ldots, B_r. \qquad (*)$$

$$A : - G_1, \ldots, G_k / S_1 || S_2 || \ldots || S_r. \qquad (**)$$

где A, $G_1, \ldots, G_k$, $B_1, \ldots, B_r$ — предикаты, а $S_1, \ldots, S_r$ — конъюнкции предикатов. Здесь A является заголовком обоих дизъюнктов, последовательность $B_1, \ldots, B_r$ — телом дизъюнкта (*), $S_1, \ldots, S_r$ — телом дизъюнкта (**), а $G_1, \ldots, G_k$ играют роль условий (предохранителей), которые должны выполняться в совокупности для активизации дизъюнктов (*) и (**). Различие между этими двумя дизъюнктами имеет процедурный характер: конъюнкции $S_1, \ldots, S_r$ обрабатываются независимыми процессорами, и унификация без отката выполняется параллельно. Реляционный язык обладает весьма ограниченными возможностями, и логические программы, выполненные на этом языке, могут применяться только для проверки выполнимости простых отношений между фактами.

Конкурентный ПРОЛОГ, предложенный Шапиро в 1983 г., использует только дизъюнкты типа (*), но зато в нем имеются переменные двух различных типов: *переменные для чтения* и обычные переменные. Переменные для чтения не допускают подстановки в них термов при унификации. Унификация откладывается до тех пор, пока эти переменные не примут конкретные значения. Возникающая при этом задержка решает также и проблему синхронизации.

ПАРЛОГ (язык ПАРаллельного ЛОГического программирования), разработанный Кларком и Грегори в 1984 г., является улучшенной модификацией Реляционного Языка. В 1985 г. Уида предложил новый язык *GHC* (*G*uarded *H*orn *C*lauses, т. е. охраняемые хорновские дизъюнкты), в котором удалось объединить лучшие черты ПАРЛОГа и Конкурентного ПРОЛОГа.

Область языков параллельного логического программирования постоянно развивается. Дополнительные сведения об этих языках можно узнать из [Shap87]

§ 3.9. Пролог и логика предикатов

Как мы уже отмечали, в логическом программировании приходится иметь дело с особым классом формул логики предикатов, а именно, с хорновскими дизъюнктами. Поэтому язык ПРОЛОГ без специального предикатного символа «.» (см. п. 3.3.7) представляет собой подмножество языка логики предикатов. При этом сама система программирования ПРОЛОГ, которая по существу есть ни что иное, как особая стратегия вывода заключений из начальных допущений в рамках логического программирования, предстает в двух различных ипостасях, которые можно условно назвать

(1) Чистый ПРОЛОГ

(2) Реальный ПРОЛОГ

Чистый ПРОЛОГ не содержит никаких средств управления вычислением, и поэтому отсечение !, а также предикаты `fail` и `not` в этом

языке отсутствуют. Реальный ПРОЛОГ является расширением чистого ПРОЛОГа за счет добавления отсечения, **fail** и **not**. Выделение этих двух разновидностей ПРОЛОГа весьма существенно, поскольку теоремы полноты для логики предикатов, будучи справедливыми в чистом ПРОЛОГе, перестают быть таковыми в реальном ПРОЛОГе.

В каждой ПРОЛОГ-программе данные представляются совокупностью предложений S логики предикатов. Всякий запрос вида «? P.» к программе требует построения доказательства или опровержения для $S \vdash_R P$. В *чистом* ПРОЛОГе теорема полноты (Теорема 2.10.11) и теорема Робинсона (Теорема 2.9.8) гарантируют, что алгоритм унификации всегда завершится, и при этом либо будут обнаружены *все* значения переменных, для которых $S \vdash_R P$, либо отношение $S \vdash_R P$ будет опровергнуто.

Применяя в *реальном* ПРОЛОГе оператор отсечения, мы коренным образом изменяем эти свойства. Отсекая поддеревья в дереве вычислений, мы можем воспрепятствовать программе найти те значения переменных, при которых $S \vdash_R P$. Показательным примером может служить программа из п. 3.4.4, содержащая правило (1') с оператором отсечения. Обращение к этому правилу вынуждает программу давать неправильный ответ.

И поэтому, хотя использование отсечения, **fail** и **not** придают программированию в реальном ПРОЛОГе большую гибкость, ни программист, ни пользователь не имеют гарантии того, что их программы будут вычислять правильные результаты.

Программы чистого ПРОЛОГа требуют порой больших затрат времени для своего исполнения. Необходимость проектирования быстродействующих программ заставила заняться разработкой приемов улучшения их производительности даже в реальном ПРОЛОГе. Реальный ПРОЛОГ, в котором отсутствует проверка вхождения переменной в терм при унификации (см. Алгоритм 2.9.7.), но зато используются отсечение, **fail** и **not**, является в настоящее время наиболее эффективным и популярным языком логического программирования. И хотя корректность программ при этом будет теоретически необоснованной, это не означает, что результаты их вычислений будут неверны. Это значит всего лишь, что программист должен быть очень осторожен в употреблении встроенных операций, отсечения и отрицания **not**. Обращаясь к реальному ПРОЛОГу, мы можем улучшить быстродействие и эффективность программы ценой отказа от общей теоремы полноты.

Коренное отличие логики предикатов от ПРОЛОГа, как чистого, так и реального, проявляется в том, как эти системы обращаются с данными программы S. ПРОЛОГ упорядочивает данные, нумеруя факты и правила в порядке их расположения в программе. И даже порядок расположения предикатов в теле правил сохраняется в точности таким, каким он был указан при написании программы. Поэтому правила

$$A :- B_1,\ B_2. \qquad (*)$$

и

$$A :- B_2,\ B_1. \qquad (**)$$

считаются различными, несмотря на то, что в логике предикатов соответствующие формулы эквивалентны благодаря коммутативности $\vee$. Более того, время выполнения программы может существенно зависеть от того, какому из этих двух правил будет отдано предпочтение.

Процедурный стиль обработки данных в ПРОЛОГе создает трудности в тех случаях, когда программа содержит правила вида

$$A :- A. \qquad (1)$$

или даже более общего вида

$$A :- B_1, \ldots, A, \ldots, B_k. \qquad (2)$$

Дело в том, что хорновские дизъюнкты вида

$$A \rightarrow A$$

или

$$(B_1 \wedge \ldots \wedge B_k \wedge A) \rightarrow A$$

являются тавтологиями (оба они эквивалентны формуле $A \wedge \neg A$), и поэтому правила (1) и (2) не добавляют программе по существу никакой новой информации. Однако, обращение с запросом «? A.» к программе, содержащей правила (1) или (2) порождает, ввиду рекурсивного определения A, бесконечный цикл, из которого программа сама по себе выйти не в состоянии. Большинство случаев зацикливания ПРОЛОГ-программ возникает в связи с наличием в них данных вида (1) или (2).

С другой стороны, логически общезначимый хорновский дизъюнкт вида

$$A :- \texttt{fail}.$$

эквивалентный в логике предикатов формуле

$$P \wedge \neg P \rightarrow A,$$

где P — произвольное предложение логики предикатов, а A — хорновский дизъюнкт, применяется в реальном ПРОЛОГе для выражения отрицания. Отрицание в реальном ПРОЛОГе гораздо сильнее обычного отрицания логики предикатов. В логике предикатов отрицание дизъюнкта A можно определить как

$$\neg A \leftrightarrow (A \rightarrow B \wedge \neg B)$$

или равносильно

$$\neg A \leftrightarrow (A \rightarrow \square)$$

Поэтому в случае

$$S \vdash_R A$$

как $\neg A$, так и $\mathtt{not}(A)$ в реальном ПРОЛОГе являются невыполнимыми предложениями (см. Определение 2.5.15 и п. 3.6.2). Однако, в случае

$$S \nvdash_R A$$

т. е. когда A резолютивно недоказуемо в программе S, в логике предикатов в общем случае нельзя определить, будет ли при этом обязательно справедливо $S \vdash_R \neg A$, в то время как в реальном ПРОЛОГе $\mathtt{not}(A)$ будет безусловно справедливо в виду допущения о замкнутости мира (см. пп. 3.6.2 и 3.6.3).

Реальный ПРОЛОГ, независимо от того, полон он или нет, получил широкое распространение и является наиболее эффективным языком логического программирования. А что касается контроля правильности программы, — так это всегда было и остается уделом программистов и пользователей.

§ 3.10. Упражнения

3.10.1. Запишите на языке ПРОЛОГ следующие высказывания обыденной речи:

а) Крис ненавидит учебу.

б) Кто-то любит Мэри.

в) Джордж дружит со всяким, кого он любит, если этот человек тоже любит его.

г) Мэри любит всякого, кто восхищается ею.

д) Энди опасается тех, кто крупнее его.

е) Энди ревнует ко всякому, кто нравится Мэри, если этот человек не менее высок, чем Энди.

Р е ш е н и е. а) **ненавидит(крис,учебу)**.

б) **любит**(X,**Мэри**).

в) **дружит(Джордж**,X): $-$**любит(Джордж**,X), **любит**(X,**Джордж**).

г) **любит(Мэри**,X): $-$**восхищается**(X,**Мэри**).

д) **опасается(Ник**,X): $-$**крупнее**(X,**Энди**).

е) **ревнует(Энди**,X): $-$**нравится(Мэри**,X),**not(ниже**(X,**Энди**)).

3.10.2. Напишите ПРОЛОГ-программу, соответствующую доказательству из упражнения 2.12.37.

Р е ш е н и е.

t(a, b, c, d). /* $abcd$ — трапеция с вершинами a, /b, /c, /d */

p(a, b, c, d) : $-$ **t**(a, b, c, d). /* $ab \parallel cd$, если $abcd$ — трапеция */

e(a, b, c, d; d, b) : $-$ **p**(a, b, c, d). /* $\widehat{abd} = \widehat{cdb}$, если $ab \parallel cd$ */

e(a, b, c, d; d, b). /* $\widehat{abd} = \widehat{cdb}$ */

3.10.3. Представьте в виде ПРОЛОГ-программы базу данных всех европейских стран вместе с их столицами. Сформируйте подходящие запросы, позволяющие определять столицу заданной страны или страну, имеющую столицей заданный город.

3.10.4. Для приведенной ниже базы данных определите отношения

выше, **выше**(X, Y) высокого роста, **высокий** (X)

среднего роста, **средний**(X) низкого роста, **низкий**(X)

Условимся считать, что мужчина, m, и женщина, f, имеют высокий рост, если он превышает 180 и 175 см соответственно, и низкий рост, если он не превышает 170 и 160 см.

рост(Билл, m, 180).
рост(Энди, m, 154).
рост(Джим, m, 187).
рост(Пол, m, 175).
рост(Джон, m, 166).
рост(Алекс, m, 176).
рост(Мэри, f, 173).
рост(Анна, f, 161).
рост(Джоан, f, 168).
рост(Кэтрин, f, 178).

Решение.

выше(X, Y) : – рост$(X, _, Z)$, рост$(Y, _, W)$, $Z > W$.
высокий(X) : – рост(X, m, Y), $Y => 180$.
высокий(X) : – рост(X, f, Y), $Y => 175$.
средний(X) : – рост(X, m, Y), $Y > 170$, $Y < 180$.
средний(X) : – рост(X, f, Y), $Y > 160$, $Y < 175$.
низкий(X) : – рост(X, m, Y), $Y =< 170$.
низкий(X) : – рост(X, f, Y), $Y =< 160$.

3.10.5. Рассмотрим следующие таблицы оборудования и его поставщиков на склад запасных частей для автомобилей

Поставщик	Код	Имя	Род деятельности	Город
	001	Джон	Фабрикант	Афины
	002	Энди	Посредник	Пирей
	010	Джон	Посредник	Салоники
	110	Ник	Импортер	Пирей

Товар	Артикул	Наименование	Модель	Вес
	003	Масло	30	300
	004	Шины	157/75	2000
	005	Лампы	RAAI	10
	013	Масло	60	500

Поставка	Код	Артикул	Количество
	001	005	150
	002	003	200
	010	004	030
	110	013	250

Представьте информацию, содержащуюся в таблицах, в виде базы данных ПРОЛОГа, используя только двухместные предикаты и запишите в виде хорновских дизъюнктов следующие запросы:

1) Как зовут поставщиков масла?

2) В каком городе проживает поставщик шин, и в каком городе проживает поставщик ламп?

3) Что поставляет Джон?

4) Какой поставщик масла обосновался в Патрасе и поставляет менее 400 тонн продукта?

5) Кто поставляет лампы, и кто поставляет шины?

Р е ш е н и е. Введем следующие предикаты:

имя(X, Y) /* X — код поставщика, Y — имя поставщика*/

профессия(X, Y) /* X — код поставщика, Y — род деятельности*/

город(X, Y) /* X — код поставщика, Y — местожительство*/

продукт(X, Y) /* X — артикул, Y — наименование оборудования*/

тип(X, Y) /* X — артикул, Y — тип оборудования*/

вес(X, Y) /* X — артикул, Y — вес оборудования*/

код_артикул(X, Y) /* X — код, Y — артикул*/

количество(X, Y) /* X — код, Y — количество продукта*/

Таким образом, образуется следующая база данных:

```
имя(001,джон).
```

```
имя(002,энди).
имя(010,джон).
имя(110, ник).
профессия(001,фабрикант).
профессия(002,импортер).
профессия(010,посредник).
профессия(110,импортер).
город(001,афины).
город(002,патрас).
город(010,салоники).
город(110,пирей).
продукт(003,масло).
продукт(004,шины).
продукт(005,лампы).
продукт(013,масло).
тип(003,"30").
тип(004,"157/75").
тип(005,"RAAI").
тип(013,"60").
вес(003,300).
вес(004,2000).
вес(005,10).
вес(013,500).
код_артикул(001,005).
код_артикул(002,003).
код_артикул(010,004).
код_артикул(110,013).
количество(001,150).
количество(002,200).
количество(010,030).
количество(110,250).
```

Сформируем следующие хорновские дизъюнкты для запросов

```
(1) поставщик_масла(ИмяПоставщика): –
                    продукт(КодПродукта,масло),
                    код_артикул(КодПоставщика,КодПродукта),
                    имя(КодПоставщика,ИмяПоставщика).
    ? поставщик_масла(ИмяПоставщика).

(2) город_продукта(ГородПоставщика,Продукт): –
                    продукт(КодПродукта,Продукт),
                    код_артикул(КодПоставщика,КодПродукта),
                    имя(КодПоставщика,ГородПоставщика).
    ? город_продукта(X,шины),город_продукта(Y,лампы).

(3) поставляет(ИмяПоставщика,Продукт): –
                    продукт(КодПоставщика,ИмяПоставщика),
                    код_артикул(КодПоставщика,КодПродукта),
```

```
                имя(КодПродукта,Продукт).

    ? поставляет(джон,Продукт).

(4) поставка_масла_патрас(ИмяПоставщика,ОбъемПоставки): –
                продукт(КодПродукта,масло),
                вес(КодПродукта,ВесПродукта),
                ВесПродукта < ОбъемПоставки,
                код_артикул(КодПоставщика,КодПродукта),
                город(КодПоставщика,патрас).
                имя(КодПоставщика,ИмяПоставщика).

    ? поставляет_масло_патрас(X,400).
(5) поставщик(ИмяПоставщика,Продукт): –
                продукт(КодПродукта,Продукт),
                код_артикул(КодПоставщика,КодПродукта),
                имя(КодПоставщика,ИмяПоставщика).

    ? поставщик(X,лампы), поставщик(Y,шины).
```

3.10.6. Проведите анализ вычислений заданной ПРОЛОГ-программы и постройте соответствующее дерево вычислений.

```
A(a, b).                                  /* 1 */
B(X) : – C(X, Y).                         /* 2 */
C(b, a) : – A(a, b).                      /* 3 */
C(X, Y) : – A(X, Y).                      /* 4 */
? B(X).
```

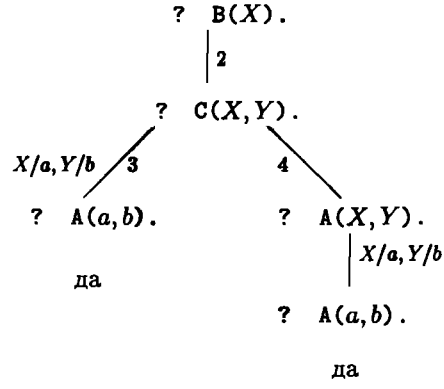
Решение. Запрос B(X) унифицируется с заголовком правила /* 2 */. Образуется цель C(X, Y). Она, в свою очередь, унифицируется с заголовком /* 3 */ при помощи подстановок X/b, Y/a. Поскольку A(a, b) унифицируется с фактом /* 1 */, вычисление успешно завершается, и ПРОЛОГ выдает ответ

$X = b$
$Y = a$

Затем ПРОЛОГ освобождает переменные X и Y от ранее полученных значений, совершает откат, пытаясь выполнить целевое утверждение C(X, Y) иным способом. Цель C(X, Y) унифицируется с заголовком правила /* 4 */. При этом образуется новая цель A(X, Y), которая унифицируется с фактом /* 1 */ при помощи подстановок X/a, Y/b. После освобождения переменных и очередного отката ПРОЛОГ не имеет возможности добиться выполнения основной цели или промежуточных целей каким-либо иным способом, и поэтому вычисление ограничивается еще одним ответом

$X = a$
$Y = b$

Дерево в пространстве вычислений имеет вид:



3.10.7. Полиция обнаружила тело несчастной женщины по имени Сюзанна, голова которой была разбита тупым орудием. Участниками этой печальной драмы являются Алан, тридцатипятилетний мясник и вор, а также Джон, двадцатипятилетний футбольный игрок, который испытывает сентиментальную привязанность к Сюзанне и Барбаре одновременно. Барбара, двадцати двух лет, парикмахер, замужем за Бертом, пятидесятилетним хромым столяром. В ходе расследования в доме Джона был обнаружен револьвер. Для полицейских ревность и запланированное ограбление служат подходящими возможными мотивами совершения преступления. Помогите им отыскать убийцу.

Р е ш е н и е. /* Формулы, выражающие предположения и факты, полученные в ходе расследования */

```
личность(джон,25,мужчина,футболист).
личность(алан,35,мужчина,мясник).
личность(барбара,22,женщина,парикмахер).
личность(берт,50,мужчина,столяр).
личность(алан,35,мужчина,карманник).
знакомы(барбара,джон).
знакомы(барбара,берт).
знакомы(сюзанна,джон).
убита(сюзанна,дубинкой).
мотив(деньги).
мотив(ревность).
имеет(берт,деревянный_костыль).
имеет(джон,пистолет).
```

/* соображения здравого смысла */

```
действуют_одинаково(деревянный_костыль,дубинка).
действуют_одинаково(брусок,дубинка).
действуют_одинаково(ножницы,нож).
действуют_одинаково(бутсы,дубинка).
```

```
возможно_имеет(X,бутсы): −личность(X,_,_,футболист).
возможно_имеет(X,ножницы): −личность(X,_,_,парикмахер).
возможно_имеет(X,Предмет): −имеет(X,Предмет).

  /* предположения о преступнике — */
  /* подозреваемые двух категорий: */
  /* имевшие возможность совершить убийство */
  /* и имевшие причину совершить убийство */

подозреваемый_по_возможности(X,Y): −убита(Y,Орудие),
                    действуют_одинаково(Орудие,Предмет),
                    возможно_имеет(X,Предмет).
подозреваемый_по_мотивам(X,Y): −мотив(ревность),
                    личность(X,_,мужчина,_),
                    знакомы(X,Y).
подозреваемый_по_мотивам(X,Y): −мотив(ревность),
                    личность(X,_,женщина,_),
                    знакомы(X,Субъект),
                    знакомы(Y,Субъект),
                    личность(Субъект,_,мужчина,_).
подозреваемый_по_мотивам(X,Y): −мотив(деньги),
                    личность(X,_,_,карманник).

  /* соображения здравого смысла */

наиболее_подозрительный(X,Y): −
                    подозреваемый_по_мотивам(X,Y),
                    подозреваемый_по_возможности(X,Y).
```

В силу своего определения предикат **наиболее_подозрительный** не имеет вероятностной интерпретации и зависит исключительно от двух предикатов

подозреваемый_по_мотивам и **подозреваемый_по_возможности**.

Проверьте, построив соответствующие полные вычисления, что на поставленные запросы построенная ПРОЛОГ-программа даст следующие ответы:

```
? подозреваемый_по_возможности(X,сюзанна).
      X=берт
      X=джон

? подозреваемый_по_мотивам(X,сюзанна).
      X=джон
      X=барбара
      X=аллен

? наиболее_подозрительный(X,сюзанна).
      X=джон
```

3.10.8. Не очень-то приятно вычислять расстояния **по карте** при подготовке к путешествию. ПРОЛОГ-программа здесь **может** очень пригодиться.

Р е ш е н и е. Рассмотрим в качестве примера следующую программу

```
/* данные из путеводителя */
дорога(каламата,триполи,90).
дорога(триполи,аргос,60).
дорога(аргос,коринф,49).
дорога(коринф,афины,83).
/* правила вычисления расстояний[1]) */
путь(Город1,Город2,Расст): -дорога(Город1,Город2,Расст).
путь(Город1,Город2,Расст): -дорога(Город1,X,Расст1),
                           путь(X,Город2,Расст2),
                           Расст = Расст1 + Расст2.
```

Каково расстояние между Каламатой и Коринфом?

```
? путь(каламата,коринф,X).
X=199
```

3.10.9. Приведите рекурсивное определение предиката **факториал**(X, Y), который подразумевает $Y = X!$, основываясь на соотношениях

а) $1! = 1$

б) $n! = (n-1)! * n$

Постройте дерево вычислений для полученной программы и запроса

```
? факториал(3, X)
```

Р е ш е н и е.

Соотношение а) задает граничное условие, а соотношение б) — рекурсивный переход. Искомое определение выглядит так:

```
факториал(1,1) : - !.                          /* 1 */
факториал(N,Фактор) : - N1 = N-1,
                        факториал(N1,Фактор1),
                        Фактор = N* Фактор1.   /* 2 */
```

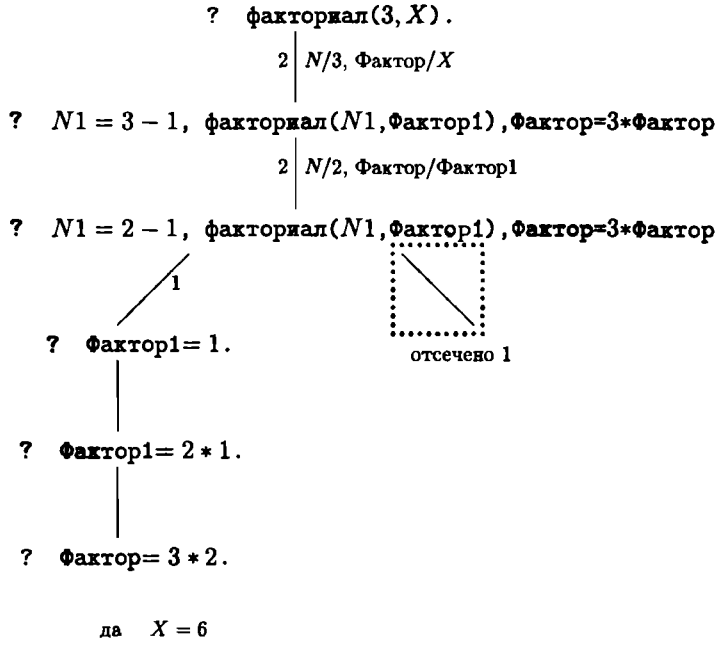
Отсечение в формуле /* 1 */ используется для предотвращения отката при вычислении 1!.

Пространство состояний запроса

```
? факториал(3, X)
```

[1]) Следует иметь в виду, что предложенная программа хорошо работает только в том случае, когда в путеводителе отсутствуют замкнутые «кругосветные» маршруты. В противном случае дерево вычислений может оказаться бесконечным. Подумайте над тем, как нужно модифицировать эту программу, чтобы она могла вычислять кратчайшее расстояние между городами. — *Прим. перев.*

представлено следующим деревом



3.10.10. Приведите рекурсивное определение предиката $\exp(x, y, z)$, который обозначает соотношение $x^y = z$ и задает рекурсивную функцию

$$x^0 = 1$$
$$x^y = x \cdot x^{y-1}$$

3.10.11. Предположим, что нам требуется определить предикат, описывающий количество родителей некоего лица x. Согласно Библии мы должны иметь ввиду, что Адам и Ева не имели родителей. Поэтому мы пишем программу $P1$

```
/* программа P1 */
    число_родителей(адам,0):- !.                    /*/1/*/
    число_родителей(ева,0):- !.                     /*/2/*/
    число_родителей(X,2).                           /*/3/*/
```

в которой применяется отсечение во избежание отката в запросах вида

```
? число_родителей(адам,X).
```

и

```
? число_родителей(адам,0).
```

а) Что выдаст ПРОЛОГ в качестве ответа на запрос

```
? число_родителей(ева,2).                    (*)
```

и почему?

б) Что выдаст ПРОЛОГ в качестве ответа на запрос **(*)**, если вместо программы $P1$ мы воспользуемся программой $P2$

```
/* программа P2 */
    число_родителей(адам,N) : - !, N = 0.        /* 1 */
    число_родителей(ева,N) : - !, N = 0.         /* 2 */
    число_родителей(X,2).                        /* 3 */
```

и каков будет ответ на запрос

```
? число_родителей(X, Y).
```

к программе $P2$?

3.10.12. Напишите ПРОЛОГ-программу вычисления абсолютной величины числа.

3.10.13. Напишите ПРОЛОГ-программу, проверяющую путем просмотра всех элементов списка, содержится ли заданный элемент (простой элемент или список) в заданном списке L.

Решение.

```
элемент(H,[H: _]).
элемент(I,[_: T]) : - элемент(I, T).
```

3.10.14. Проверьте содержится ли список H в списке L в качестве элемента, и определите первый элемент списка L. Примените написанную программу к списку $L = [a, b, c,]$.

Решение.

```
элемент(H,[H: _]) : - !.                     /* 1 */
элемент(I,[_: T]) : - элемент(I, T).         /* 2 */
```

Единственным решением, найденным ПРОЛОГом в ответ на запрос

```
? элемент(X,[a, b, c]).
```

будет $X = a$. Наличие отсечения в дизъюнкте /* 1 */ делает невозможным обнаружение других ответов.

3.10.15. Напишите программу, выделяющую n первых элементов списка.

3.10.16. Напишите программу, вычисляющую разность двух списков.

Решение.

```
вычесть(L,[],L): - !.
вычесть([H: T],L,U): -элемент(H,L),!,вычесть(T,L,U).
вычесть(_,_,[]).
элемент(H,[H: _]).
элемент(I,[_: T]): -элемент(I,T).
```

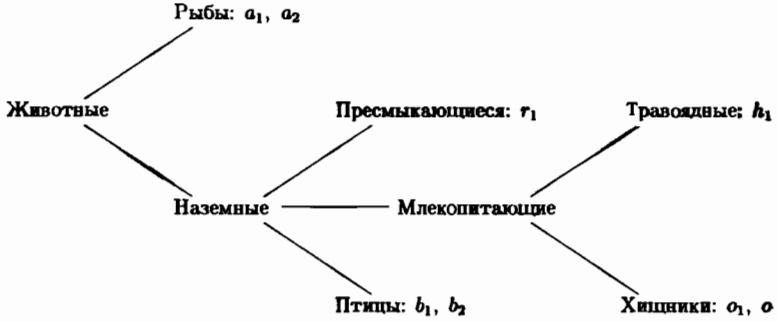
3.10.17. Определите в терминах ПРОЛОГ-программы теоретико-множественные отношения

элемент, подмножество и пересечение

Решение.

```
/* ∈ */
элемент(H, [H: _]).
элемент(I, [_: T]): −элемент(I, T).
/* ⊆ */
подмножество([H: T], I): −элемент(H, I), подмножество(T, I).
подмножество([], I).
/* ∩ */
пересечение([], X, []).
пересечение([X, T], I, [X, Z]): −элемент(X, I), !,
                пересечение(T, I, Z).
пересечение([X, T], I, Z): −пересечение(T, I, Z)[1]).
```

3.10.18. Предположим, что мы располагаем списком из восьми животных, $V = [a1, a2, r1, b1, h1, o1, o2]$, которые классифицируются следующим образом.



Заведите базу данных, описывающую приведенную классификацию, а также ПРОЛОГ-программу, способную отвечать на запросы

а) К какому классу относится животное X?

б) Какие животные относятся к классу Y?

Решение.

```
животное(C, X) : − водное(C, X), наземное(C, X).
водное(рыба,[a1, a2]).
наземное(пресмыкающееся,[r1]).
наземное(птица,[b1, b2]).
```

[1]) Данная программа верно определяет трехместное отношение пересечения пары множеств, представленных списками, при том условии, что порядок расположения общих элементов в третьем списке (переменная Z) совпадает с порядком их расположения в списке, задающем первое множество (переменная T). — *Прим. перев.*

```
наземное(C, X):- млекопитающее(C, X).
млекопитающее(травоядное,[h1]).
млекопитающее(хищник,[o1, o2]).
классификация(C, X): -животное(C, L),элемент(X, L).
```

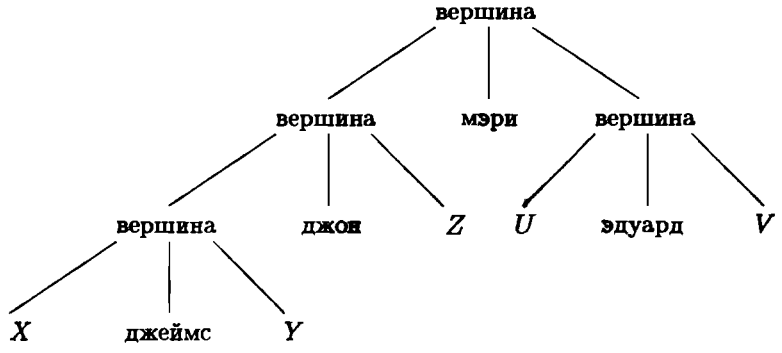
Для получения ответа на указанные вопросы следует обратиться к программе с запросом

```
? классификация(C, X).
```

3.10.19. Напишите ПРОЛОГ-программу, выдающую на экран список имен в алфавитном порядке. Имена в списке разделяются точкой, а признаком окончания списка должно служить слово `stop`.

Р е ш е н и е. Упорядочить имен по алфавиту можно при помощи дерева. Каждая вершина дерева соответствует предикату `node`(L, W, R), который интерпретируется как «L — левый непосредственный последователь текущей вершины, R — правый непосредственный последователь текущей вершины, а W — это имя, которое должно быть прочитано программой».

Каждое имя сравнивается с именем в корне дерева, а затем процедура сравнения продолжается рекурсивно применительно к левым или правым последователям соответствующей вершины. Выполнение этой процедуры прекращается как только очередная вершина оказывается переменной или как только искомое имя будет обнаружено в дереве. Например, для списка имен `джон.мэри.джеймс.эдуард.stop`. мы будем иметь следующее дерево



Искомая программа такова

```
порядок(X): -прочитать(X),имя(W, W1),
            разбор(W1, X, Y),порядок(Y).
порядок(X): - nl,write("упорядоченные слова:"),nl,
            записать(X),nl,nl,write("Выполнено."),nl.
прочитать(W): -read(W),W==stop, !, fail.
разбор(W,вершина(L, W, R),вершина(L, W, R)): - !.
разбор(W,вершина(L, W1, R),вершина(U, W1, R)): -
```

```
        ниже(W, W1), !, разбор(W, L, U).
разбор(W, вершина(L, W1, R), вершина(L, W1, U)): - !,
        разбор(W, R, U).
ниже([], _): - !, ниже(Y, T).
ниже([W: Y], [Z, T]): - X < Z.
записать(X): - var(X), !.
записать(вершина(L, W1, R)): - !, записать(L),
        имя(W, W1), write(W, W1), write("."),
        записать(R).
```

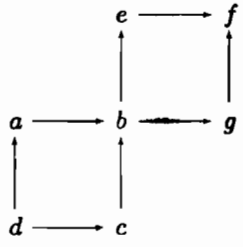
Для запуска программы необходимо обратиться к ней с запросом

```
? порядок(X).
```

где X — список имен, который должен быть упорядочен.

3.10.20. Напишите ПРОЛОГ-программу, которая по заданному списку чисел формирует дерево и возвращает древесное представление этого списка, в котором все элементы упорядочены.

3.10.21. Напишите программу, представляющую диаграмму

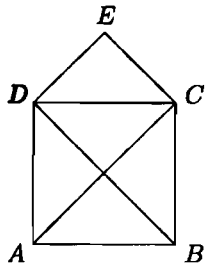


и найдите с ее помощью непосредственных последователей c и d.

Решение. Взаимосвязь вершин описывается предикатом дуга(x, y), который трактуется как «существует дуга, ведущая из x в y». Искомая программа такова.

```
дуга(a, b).
дуга(c, b).
дуга(d, a).
дуга(d, c).
дуга(b, e).
дуга(e, f).
дуга(b, g).
дуга(g, f).
? дуга(c, x).
? дуга(g, y).
```

3.10.22. Напишите программу, способную нарисовать приведенную ниже фигуру при помощи одной непрерывной линии, которую можно провести, не отрывая карандаша от бумаги.



Решение.

```
рисуй(G,[P,Q: L]) :- выбери(G,отрезок(P,Q.G1)),
                     рисуй(G1,[Q,L]).
рисуй([],[Q]).
выбери([отрезок(P,Q): G],отрезок(P,Q),G).
выбери([отрезок(P,Q): G],отрезок(Q,P),G).
выбери([E: G],F,[E,G1]): -выбери(G,F,G1).
  ? рисуй([отрезок(a,b),отрезок(a,c),отрезок(a,d),
    отрезок(b,c),отрезок(b,d),отрезок(c,d),
    отрезок(c,e),отрезок(d,e)],L),write(L).
```

3.10.23. Напишите программу, определяющую взаимное расположение точек на декартовой плоскости.

а) точка (a, b) лежит внутри круга с центром в точке (c, d) и радиусом r.

б) три точки плоскости (x_1, y_1), (x_2, y_2) и (x_3, y_3) лежат на одной прямой.

Решение. Каждая точка описывается предикатом

```
точка(x, y),
```

где x и y — координаты точки.
Соотношение, описывающее все внутренние точки круга имеет вид

$$(x-c)^2+(y-d)^2 \leqslant r^2$$

Три точки декартовой плоскости лежат на одной прямой в том и только том случае, когда

$$\frac{y_3-y_1}{x_3-x_1} = \frac{y_2-y_1}{x_2-x_1}$$

Наконец, предикат

```
внутри(точка(x, y),(c, d),r),
```

утверждает, что

```
точка(x, y),
```

располагается внутри круга с центром (c, d) и радиусом r. Встроенные предикаты и арифметические операции, необходимые для построения программы, могут быть найдены в § 3.5.4.

3.10.24. Определите сочетания элементов из заданного списка.

3.10.25. Напишите алгоритм, проверяющий совпадение заданной точки с одной из вершин заданного дерева, учитывая при этом, что каждая вершина дерева имеет не более двух непосредственных последователей (Определение 2.8.9). В том случае, если заданная точка не содержится в дереве, алгоритм должен присоединить ее к дереву в качестве новой вершины.

3.10.26. Джордж, Тим, Джон и Билл — поклонники футбола, а Ник и Джим болеют за команды A и B соответственно. Нику симпатичен всякий, кто болеет за команду A, в то время как Джорджу нравятся любители футбола, не являющиеся сторонниками команды B. Сформулируйте эти условия в терминах базы данных ПРОЛОГа и найдите ответ на следующие вопросы:

а) Кто нравится Нику?

б) Симпатичен ли Билл Джорджу?

с) Кто нравится Джорджу?

Решение.

```
/* Заданные условия */
    болеет_за(ник, a).
    болеет_за(джим, b).
    любит(джордж, футбол).
    любит(тим, футбол).
    любит(джон, футбол).
    любит(билл, футбол).
    любит(X, футбол) :- болеет_за(X, a).
    любит(X, футбол) :- болеет_за(X, b).
    нравится(ник, X) :- болеет_за(X, a).
    нравится(джордж, X) :- любит(X, футбол),
                           not(болеет_за(X, b)).
/* Запросы и ответы */
    ? нравится(ник, X).
    X = джим
    ? нравится(джордж, билл).
    да
    ? нравится(джордж, X).
    X = джордж
    X = тим
    X = джон
    X = билл
```

3.10.27. Предположим, что имеется некоторая ПРОЛОГ-программа Q. Исследуйте запросы

```
(1)    ? not(p).
(2)    ? not(not(p)).
```

(3) `? not(not(not(`p`)))`.

. . .

(2n) ? $\underbrace{\texttt{not(...(not(}p\texttt{))...)}}_{2n}$.

(2n + 1) ? $\underbrace{\texttt{not(...(not(}p\texttt{))...)}}_{2n+1}$.

где p — предикат не содержащий свободных переменных.

Решение. Допустим, что p успешно выполняется в программе Q. Тогда цель `not(`p`)` терпит неудачу, и поэтому ПРОЛОГ ответит «нет» на запрос (1), «да» на запрос (2), «нет» на запрос (3), ..., «да» на запрос (2n), и «нет» на запрос (2n + 1) по определению оператора `not`.

Если же p терпит неудачу, то `not(`p`)`, напротив, успешен, и ПРОЛОГ ПРОЛОГ ответит «да» на запрос (1), «нет» на запрос (2), «да» на запрос (3), ..., «нет» на запрос (2n), и «да» на запрос (2n + 1).

В общем случае, предикат $\texttt{not}^{2n}(p)$ принимает то же значение, что и p, тогда как предикат $\texttt{not}^{2n+1}(p)$ принимает то же значение, что и `not(`p`)`, в точности также, как ведет себя логическое отрицание.

Различие между `not` и отрицанием логики предикатов подробно рассмотрено в п. 3.6.1 и § 3.9.

3.10.28. Постройте простую систему лексического анализа предложений

«Ребенок спит»

и

«Ребенок ест яблоко».

Решение.

```
предложение(X, Y) : – подлежащее(X, U), сказуемое(U, Y).
подлежащее(X, Y) : – существительное(X, Y).
сказуемое(X, Y) : – непереходный_глагол(X, Y).
сказуемое(X, Y) : – переходный_глагол(X, U),
          дополнение(U, Y).
существительное([ребенок: Y], Y).
существительное([яблоко: Y], Y).
переходный_глагол([спит: Y], Y)
непереходный_глагол([спит: Y], Y)
переходный_глагол([ест: Y], Y)
```

3.10.29. Постройте простую систему лексического анализа предложения

«Мэри съела торт»

3.10.30. Представлено следующее естественно-языковое предложение

P: Джордж занят покупками или он находится дома

Постройте соответствующий дизъюнкт для P, а также следствия, которые можно вывести из базы данных, содержащей это утверждение.

Решение. Предположим, что A и B обозначают предложения

A: Джордж занят покупками.

B: Джордж находится дома.

Тогда в рамках логики предикатов P примет вид

$$P: \quad A \vee B$$

который эквивалентен любой из следующих форм

$$\neg A \rightarrow B$$
$$\neg B \rightarrow A$$

Дизъюнкты, соответствующие этим предложениям, таковы.

B:- not(A). /* 1 */

A:- not(B). /* 2 */

Если база данных ПРОЛОГ-программы состоит из дизъюнктов /* 1 */ и /* 2 */, то запросы

? A.

? B.

не найдут ответа, поскольку вычисление зациклится, и это легко заметить, рассмотрев пространство состояний программы.

Один из путей получения решения заключается в том, чтобы образовать две разные программы, состоящие из дизъюнктов /* 1 */ и /* 2 */ соответственно. В этом случае пространство состояний каждой из программ для тех же запросов будет иметь вид

Программа 1

B:- not(A). /* 1 */
? B.
? not(A).

Программа 2

A:- not(B). /* 2 */
? A.
not(B).

На оба запроса ПРОЛОГ, применяя правило NF выдаст положительный ответ «да». Таким образом в программе 1 выводимы «not (A)» и «B», а в программе 2 выводимы «not (B)» и «A». Обратите

внимание на то, что из одного и того же предложения P выводимы два различных и взаимно противоречивых множества следствий. Проблемы такого рода возникают всякий раз, когда мы имеем дело с предложениями, подобными P, которые не могут быть представлены хорновскими дизъюнктами, и для которых нет возможности определить, является одна или несколько составных частей этого предложения тождественно истинной или тождественно ложной. Так, например, в нашем случае невозможно установить истинностные значения A или B. Как было показано в Определении 1.9.7, составные хорновские дизъюнкты должны содержать хотя бы одну отрицательную литеру. Это условие не соблюдается для дизъюнкта $A \vee B$, где A и B — предложения, не содержащие отрицаний.

БИБЛИОГРАФИЯ

1. РЕКОМЕНДАЦИИ ДЛЯ ДАЛЬНЕЙШЕГО ЧТЕНИЯ

1.1. Логика высказываний.

Читателю, интересующемуся историей предмета и его развитием, рекомендуем обратиться к работам [Heij67, Boch62] и [NeSh93], содержащих обширный список литературы.

Об аксиоматизации и правилах вывода в логике высказываний подробно написано в [Schm60, Chur56, HiAc28, Hami78, Raut79, Mend64]. Эти же вопросы на более высоком уровне изложения представлены в [Klee52] в разделе, посвященном логике высказываний.

О методе семантических таблиц см. [Smul68, NeSh93].

О булевой и других алгебрах логики см. [RaSi70, Rasi74].

О модальной логике см. [Chel90, HuCr68, NeSh93, Raut79, Schm60] Для более глубокого изучения связи модальной логики и логики предикатов рекомендуем обратиться к [Bent83].

Более подробные сведения об интуиционнстской логике содержатся в работах [Brou75, Dumm77, Fitt69, NeSh93, Raut79].

О методе резолюций см. [ChLe73, Dela87] в разделах, касающихся логикн высказываний.

1.2. Логика предикатов.

Для ознакомления с историей возникновения и развития этой области математической логики рекомендуем обратиться к работам [Heij67, Boye68, Boch62] и [NeSh93], где также содержится обширный список литературы.

Об аксиоматических методах в логики предикатов можно узнать из работ [Chur56, Hami78, Mend64, HiAc28]. Этот же материал на более высоком уровне изложения представлен в [Klee52].

О методе семантических таблиц см. [Smul68, NeSh93].

Теореме Эрбрана посвящены работы [Herb30] в [Heij67], а также [ChLe73, Dela87, NeSh93, NiMa95].

О проблеме разрешимости см. [Acke54, Chur56, Klee52].

О методе резолюций см. [ChLe73, Dela87, NiMa95, NeSh93], а также стандартный учебник по логическому программированию [Lloy87].

Наиболее значительные статьи по логике, изданные за период с 1879 по 1931 год, содержаться в [Heij67]

Великолепное изложение теоремы Гёделя представлено в [Klee52].

1.3. Логическое программирование.

С основами логического программирования можно ознакомиться в работах [ChLe73, Dela87, Kowa79, Kowa90, Lloy87, NeSh93, NiMa95].

О ПРОЛОГе см. [Brat90, ClMc84, ClMe94, Dela87, NiMa95, StSh86].

Семантике отрицания в логическом программировании посвящена серия работ [Kowa90, Lloy87, NeSh93, NiMa95, Shep88, Shep92]

О метапрограммировании см. [Kowa90] и [HiLl94], где представлен новый язык программирования ГЁДЕЛЬ.

О немонотонных логиках см. [MaTr93].

2. СПИСОК ЛИТЕРАТУРЫ

[Acke54] Ackermann, W., *Solvable Cases of the Decision Problem*, North-Holland Pub. Co., 1954.

[Bent93] van Benthem, J.F.A.K., *Modal Logic and Classical Logic*, Bibliopolis, Napoli, 1983

[Besn89] Besnard, P., *An Introduction to Default Logic*, Springer-Verlag, 1989.

[Beth68] Beth E.W., *The Foundation of Mathematics; a study in the Philosophy of Science*, 3rd edn., North-Holland Pub Co. 1968.

[Boch62] Bochenski, J. M., *Formale Logik*, (на немецком), 2nd edn., Verlag Karl Alber, Freiburg-München, 1962. Translated as: *A History of Formal Logic*, Thomas, I., tr., Chelsea Pub Co., 1970

[Boye68] Boyer, C. B., *A History of Mathematics*, Princeton University Press, 1985

[Brat90] Bratko, I., *PROLOG Programming for Artifical Intelligence*, 2nd edn., Addison-Wesley Pup. Co., 1990.

[Brou75] Brouwer, L. E. J., *Collected Works*, Heyting, A., ed., North-Holland Pub. Co , 1975

[BuSh84] Buchanan, B. G., Shortliffe, E. H., eds., *Rule-based Expert Systems: the MYSIN experiments of the Standfort Heuristic Programming Project*, Addison-Wesley Pup. Co., 1984.

[CCPe85] Coelho, H., Cotta, J. C., Pereira, L. M., *How to Solve it with PROLOG*, 4th edn., Ministério do Equipamento Social, Laboratório Nacional de Engenharia Civil, Lisbon, 1985

[Chel80] Chellas, B. F., *Modal Logic, an Introduction*, Cambridge University Press, 1980

[ChLe73] Chang, C-L , Lee, C-T., *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, 1973.

[Chur36] Church, A., *A Note on the Entscheidungsproblem*, Journal of Symbolic Logic, **1**, 1936, pp.40-41, and correction, ibid., pp. 101-102.

[Chur56] Church, A., *Introduction to Mathematical Logic, Volume I*, Princeton University Press, 1956.

[Clar78] Clark, K. L., *Negation as Failure*, in [GaMi78], pp 293-322.

[ClMc84] Clark, K. L., McCabe, F. G., *MICRO-PROLOG: Programming in Logic*, Prentice-Hall International, 1984

[ClMe94] Clocksin, W. F , Mellish, C. S., *Programming in PROLOG*, 4th edn., Springer-Verlag, 1989.

[Cohe90] Cohen, J., *Constraint Logic Programming Languages*, Communications of the ACM, **33**, 7, July 1990, pp. 52-68.

[Colm87] Colmerauer, A., *An Introduction to PROLOG III*, in [Espr87], Part I, North-Holland, 1987, pp. 611-629.

[Colm90] Colmerauer, A , *An Introduction to PROLOG III*, in [Lloy90], pp. 37-79

[Col90a] Colmerauer, A., *An Introduction to PROLOG III*, Communications of the ACM, **33**, 7, July 1990, pp. 69-90.

[Curr63] Curry, H. B., *Foundations of Mathematical Logic*, McGraw-Hill, 1963, Dover Pub., 1977.

[Dela87] Delahaye, J.-P., *Formal Methods in Artificial Intelligence*, Howlett, J., tr., North Oxford Academic Publishers Ltd., London, 1987, Wiley, New York, 1987.

[DiSc90] Dijkstra, E. W., Scholten, C. S., *Predicate Calculus and Program Semantics*, Springer-Verlag, 1990.

[Damm77] Dummet, M. A. E., *Elements of Intuitionism*, Clarendon Press, Oxford, 1977.

[Espr87] ESPRIT '87: *Achievements and Impacts*, Proceedings of the 4th Annual ESPRIT Conference, Brussels, September 28-29, 1987.

[Fitt69] Fitting, M. C., *Intuitionistic Logic, Model Theory and Forcing*, North-Holland Pub. Co., 1969.

[Fitt90] Fitting, M. C., *First-Order Logic and Automated Theorem Proving*, Springer-Verlag, 1990.

[GaMi78] Gallaire, H., Minker, J., eds., *Logic and Date Bases*, Proceedings of the Symposium on Logic and Data Bases held at the Centre d'Études et de Recherches de L'École National Supérieure de L'Aéronautique et de L'Espace de Toulouse (C.E.R.T.), Toulouse, November 16-18, 1977, Plenum Press, New York, 1978.

[Hami78] Hamilton, A. G., *Logic for Mathematicians*, Cambridge University Press, 1978.

[Heij67] van Heijernoort, J., *From Frege to Gödel. A Source Book in Mathematical Logic, 1879-1931*, Harvard University Press, 1967.

[Herb30] Herbrand, J., *Recherches sur la théorie de la démonstration*, thesis at the University of Paris. Chapter 5 translated as: *Investigations in Proof Theory: the Properties of True Propositions*, Dreben, B , van Heijenoort., tr., in [Heij67], pp. 525-581.

[HiAc28] Hilbert, D , Ackermann, W., *Grundzüge der theoretischen Logic*, Springer-Verlag, Berlin, 1928. The 2^{ed} edn. translated as: *Principles of Mathimatical Logic*, Hammond, L. M., Leckie, G. G., Steinhardt, F., tr., Luce, R. E., ed., Chelsea Pub. Co., 1950

[HiLl94] Hill, P., Lloyd, J. W , *The GÖDEL Programming Language*, MIT Press, 1994.

[HuCr68] Hughes, G. E., Cresswell, M. J., *An Introduction to Modal Logic*, Methuen and Co. Ltd, 1968, Routledge, 1989.

[Klee52] Kleene, S. C., *Introduction to Metamathematics*, corrected reprint of the 1952 edn., North Holland Pub. Co., 1971.

[Kowa79] Kowalski, R. A., *Logic for problem solving*, North-Holland Pub. Co , 1976, reprinted, 1986

[Kowa90] Kowalski, R. A., *Problems and Promises of Computational Logic*, in [Lloy90], pp. 1-36.

[Lloy87] Lloyd, J. W., *Foundations of Logic Programming*, 2nd edn, Springer-Verlag, 1987.

[Lloy90] Lloyd, J. W., ed., *Computational Logic: Symposium Proceedings, Brussels, November 13-14, 1990*, ESPRIT Basic Research Series, Springer-Verlag, 1990.

[MaTr93] Marek, V. W., Truszczyński, M., *Nonmonotonic Logic: Context Dependent Reasoning*, Springer-Verlag, 1993.

[Mend64] Mendelson, E., *Introduction to Mathematical Logic*, Van Nostrand Company, Inc , 1964

[Meta85] Metakides, G., *Mathematical Logic*, (на греческом), University of Patras, Greece, 1985.

[Meta92] Metakides, G., *From Logic to Logic Programming*, (на греческом), Kardamitsa Pub., Athens, Greece, 1992.

[Mink88] Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Pub., Los Altos, CA, 1988.

[Mitc86] Mitcie, D., ed., *Expert Systems in the Micro-Electronic Age*, Edinburg University Press, 1979, reprinted 1986.

[Mosc92] Moschovakis, V. N., ed., *Logic from Computer Science: Proceedings of a Workshop held November 13-17, 1989*, MSRI Publications 21, Springer-Verlag, 1992.

[NeSh93] Nerode, A., Shore, R. A., *Logic for Applications*, Springer-Verlag, 1993.

[Nils71] Nilsson, N. J., *Problem-solving Methods in Artificial Intelligence*, McGraw-Hill, 1971.

[NiMa95] Nilsson, U., Maluszynski, J., *Logic, Programming and PROLOG*, 2nd edn.,John Wiley and Sons, 1995.

[OlRu87] Olson, J. R., Rueter, H. H., *Extracting Expertise from Expert: Methods for Knowledge Aquisition*, Technical Report, University of Michigan, Cognitive Science and Mashine Intelligence Laboratory, N 13, 1987.

[Quin86] Quinlan, J. R., *Discovering Rules by Induction from Large Collections of Examples*, in [Mitc86], pp. 168-201.

[Rasi74] Rasiowa, H , *An Algebraic Approach to Non-Classical Logic*, North-Holland / American Elsevier Pub. Co., 1974.

[RaSi70] Rasiowa, H., Sikorski, R., *The Mathematics of Metamathematics*, 3rd edn , PWN-Polish Scientific Publishers, 1970.

[Raut79] Rautenberg, W., *Klassische und nichtklassische Aussagenlogic*, Vieweg, 1979.

[Reit78] Reiter, R., *On Closed World Data Bases*, in Logic and Databases, Plenum Press, New York, 1978.

[Robi65] Robinson, J. A., *A Mashine-Oriented Logic Based on the Resolution Principle*, Journal of the Association for Computing Machinery, **12**, 1, 1965, pp. 23-41.

[Schm60] Schmidt, H. A., *Mathematische Gesetze der Logic I*, Springer-Verlag, 1960.

[Schw71] Schwabhäuser, W., *Modelltheorie, Bd. I* Hochultaschenbuecher, Band 813, 1971.

[Shap87] *Concurrent PROLOG· Collected Papers*, Shapiro, E., ed, vols. 1 and 2, MIT Press, 1987.

[Shep88] Shepherdson, J., *Negation in Logic Programming*, in [Mink88], pp. 19-88.

[Shep92] Shepherdson, J., *Logic for Negation as Failure*, in [Mosc92].

[Smul68] Smullyan, R. M., *First Order Logic*, Springer-Verlag, 1968, Dover Publ., 1995.

[StSh86] Sterling, L., Shapiro, E., *The Art of PROLOG: Advanced Programming Techniques*, MIT Press, 1986.

[Thay88] Thayse, A., ed., *From Standard Logic to Logic Programming: Introduced a Logic Based Approach to Artifical Intelligence*, John Wiley }] Sons, 1988.

[Turi37] Turing, A. M., *On computable numbers with an application to the Entscheidungsproblem*, Proc. London Math. Soc., **42**, 1937, pp. 230-265. A correction, ibid, **43**, 1937, pp. 544-546.

[Turn84] Turner, R., *Logic for Artifical Intelligence*, Ellis Horwood Series in Artifical Intelligence, Halsted Press, 1984

[Watt90] Watt, D. A., *Programming Language Concepts and Paradigms*, Prentis Hall, 1990.

[WeKu84] Weiss, S. M., Kulikowski, C. A., *A Practical Guide to Designing Expert Systems*, Chapman and Hall Ltd., London, 1984.

[WiBe89] Winston, P. H., Horn, B. K. P., *LISP*, 3rd edn., Addison-Wesley, Reading, MA, 1981.

[Xant90] Xanthakis, S., *PROLOG, Programming Techniques*, (на греческом), New Technology Editions, Athens, 1990.

[Zeno76] Diogenes Laertius, *Lives of Eminent Philosophers, Diogenes Laertius, Volume II*, Book VII, Zeno, para. 76, Hicks, R. D., tr., Loeb Classical Library, pp. 182, 184 (Greek), pp. 183, 185 (English), Heinemann, London, and Harvard University Press, 1925.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Автоматическое доказательство теорем (automatic theorem proving) 38, 122
аксиоматическая система вывода (axiomatic proofs) 44
аксиоматическая система логики высказываний (axiomatic system of propositional logic) 44, 47
———— (of prdicate logic) предикатов 99
алгебра булева (Boolean Algebra) 19
алгоритм построения замкнутых систематических таблиц (construction of a complete systematic tableau) 130–131
—— КНФ (algorithm constraction of a CNF) 49, 50
—— ПНФ (construction of a PNF) 118–119
—— СНФ (construction of a SNF) 120
—— семантического дерева 134–135
—— эрбрановского универсума (construction of the Herbrand Universe) 122–123
— унификации (unification) 143–144, 145, 206–209
— Эрбрана (Herbrand) 190
аппарат вывода (inference mechanism) 182, 210, 246
аристотелев мир (Aristotelean world) 22
АРИТИ-ПРОЛОГ (ARITY-PROLOG) 252
ассоциативность (associativity) 49
атом (atom) 15, 94

База данных (database) 193, 196, 246
БЕЙСИК (BASIC) 186
Бет 38
Брауэр (Brouwer) 26

Варианты множеств формул (variants of sets of formulae) 98
вершина заключительная (final) 132
— семантической таблицы (of a semantic tableau) 40
—— обычная (unused) 41
—— особая (used) 41
—— подходящая (suitable) 66
ветвь дерева (branch of a tree) 133
— семантического дерева (of a semantic tree) 134
—— полная (complete) 135
—— противоречивая (contradictory) 134
Ветвь семантической таблицы (branch of a semantic tableau) 41
———— противоречивая (contradictory) 41
взаимодействие с экраном (screen interaction) 227
— с клавиатурой (keyboard interaction) 226
вхождение переменной свободное (occurence, free, of variable) 96
—— связанное (bound) 95
вывод 45
— в аксиоматической системе (axiomatic) логики предикатов 103
— по Бету (Beth proof) 42
—— резолютивный (resolution) 56
———— в логике предикатов 147
———— высказываний 44
выражение (expression) 15
— правильно построенное (well-formed) 15
высказывание (proposition) 15
— атомарное (atomic) 15
— выводимое (provable) 45
—— по Бету (Beth-provable) 42
— выполнимое (satisfiable) 21
— дедуктивно выводимое по Бету (Beth-deduction) 63
— доказуемое (provable) 45
—— по Бету (Beth-provable) 42
— логически истинное (logically true) 21
—— ложное (logically false) 21
— невыполнимое (not verifiable) 21
— опровержимое по Бету (Beth-refutable) 42
— подтверждаемое (verifiable) 21
— составное (compound) 15
— , таблица истинности (truth table) 24
— , теоретико-множественное представление (set-theoretic representation) 51
высказывания логически эквивалентные (logically equivalent) 22

Генцен (Gentzen) 38, 47
Гилмор (Gilmore) 190
Гильберт (Hilbert) 25
Гёдель 154
Гераклит 91

Данные (data) 187, 192
— добавление (addition) 225
— синтаксис (syntax) 198
— список (list) 203
— удаление (deletion) 226
доказательство 45
дедуктивный вывод (deduction) 63
дерево (tree) 132
— семантическое (semantic) 134
— фамильное 217
детерминированное свойство ПРОЛОГа (deterministic nature of PROLOG) 212
Диагностическая система (diagnostic system) 247
дизъюнкт (clause) 51
— в логике предикатов 104
— пустой (empty) 51
— резолютивно выводимый (provable by resolution) 56
— унитарный (unit) 53
— хорновский (Horn) 52, 105, 182
дизъюнктивная нормальная форма, ДНФ (disjunctive Normal Form) 35
дизъюнкция (disjunction) 15
дистрибутивность (distributivity) 49, 101, 118
доказательство (proof) 42
— по Бету (Beth) 42
допустимый переход (allowed transition) 212
допущение замкнутости мира (closed world assumption) 189, 233
дуга дерева (arc of a tree) 132

Заголовок дизъюнкта (head of a clause) 52
— правила (of a rule) 193
закон Пирса (Pierce's Law) 41
— двойного отрицания (Law of Double Negation) 28
— де Моргана (of De Morgan) 28
— исключенного третьего (Excluded Middle Law) 28
— контрапозиции (Contrapositive Law) 28
— силлогистики второй (Second Syllogistic Law) 28
— силлогистики первый (First Syllogistic Law) 28
— транспозиции (Transportation Law) 28
замена эквивалентная (substitution of equivalences) 101
запрос (query) 193
— к программе (of a program) 192, 196
— конъюнктивный (conjunctive) 198
Запрос неуспешный (fail) 197
— обработки данных (data management) 198
— проверки данных (data verification) 198
— составной (complex) 198
запятая (comma) 94
ЗСТ 128, 131

Импликация (implication) 15
индукция (induction)
— , базис индукции (first step) 15
— , индуктивный переход (inductive step) 15
— , общая схема индукции для высказываний (general scheme of induction for proposition) 16
— полная (complete) 16
— , принцип математической индукции (principle of mathematical induction) 16
— , схема для семантических таблиц (scheme for semantic tableaux) 58
интерпретация (interpretation) 31
— в логике предикатов 109–112
— логики предикатов 92
— эрбрановская (Herbrand) 124
интуиционистская логика (intuitionistic logic) 26
исключенного третьего принцип (principle of the Excluded Middle) 22
искусственный интеллект (artificial intelegence) 245
истинностное означивание (truth valuation) 19
—— согласованное с ветвью (agrees with branch) 59
истинностные значения (truth values) 18
истинность предложения (truth of a sentence) 111
— универсальная (universal) 93
— частичная (partial) 93
исчисление систематических таблиц (systematic tableau proofs) 126
λ-исчисление 191

Кант (Kant) 26
квантор (quantifier) 93, 94
— всеобщности (universal) 93
— двойственный (dual) 94
— существования (existential) 93
Кенига лемма (Koenig's lemma) 66
КОБОЛ (COBOL) 186
Ковальский (Kowalski) 190
Колмероэ (Colmerauer) 190
команда (command) 186
— императивная (imperative) 186

Комбинация отсечения и неудачи (failure-cut combination) 242
комментарии к ПРОЛОГ программе (comments to PROLOG program) 210
коммутативность (commutativity) 49, 101
композиция подстановок (composition of substitution) 97
конечная степень ветвления семантической таблицы (finite degree of a semantic tableau) 66
конкатенация (concatenation) 224
константа (constant) логики предикатов 92, 94
конъюнктивная нормальная форма (Conjunctive Normal Form) 36
конъюнкция (conjunction) 15
корень дерева (origin of a tree) 132

ЛИСП (LISP) 203
Литерал (literal) 49
— в логике предикатов 104
логика высказываний (logic propositional) 14
— интуиционистская (intuitionistic) 26
— модальная (modal) 22
— немонотонная (nonmonotonic) 243
логическая длина высказывания (logical length of a proposition) 15
логическое следствие (consequence) 28
Лукашевич (Lukasiewicz) 48

Матрица (matrix) формулы 117
местность (degree, arity) предиката 94
метавысказывание (metaproposition) 29
— логики высказываний (of propositional logic) 29
метаязык (metalanguage) 29
— логики высказываний (of propositional logic) 29
МИКРО-ПРОЛОГ (MICRO-PROLOG) 252
множество высказываний выполнимое (set of satisfiable propositions) 31
—— невыполнимое (non-satisfiable) 31
—— неподтверждаемое (non-verifiable) 31
—— непротиворечивое (consistent) 31
—— подтверждаемое (verifiable) 31
—— противоречивое (inconsistent) 31
— дизъюнктов (of clauses) 51
—— пустое 51
—— опровергаемое семантическим деревом (refutable by semantic tree) 135
Множество подстановочное (substitution set) 96
— полное 34
— предложений выполнимое (of sentences satisfiable) 114
—— невыполнимое (non-satisfiable) 114
—— непротиворечивое (consistent) 114
—— противоречивое (inconsistent) 114
— рассогласования (desagreement) 144
— эрбрановское (Herbrand) 123
модальные операторы (modal operators) 22
модель предложения (model of sentence) 112

Наиболее общий унификатор (НОУ) 144
Негативные знания (negative knowledge) 234
Неудача (failure) 189
Неуспешное завершение (fail) 197
Нормализация переменных (normalization of variables) 147

Область интерпретации (universe of an interpretation) 109
общий унификатор в ПРОЛОГе (general unifier in PROLOG) 207
означивание (valuation) 18
— булево (Boolean) 19
— истинностное (truth) 19
— , расширение до истинностного означивания (extention) 20
ОККАМ (OCCAM) 252
оператор (operator)
— , ассоциативность (associativity) 230
— инфиксный (infix) 230
— модальный (modal) 22
— позиции (position) 230
— постфиксный (postfix) 230
— префиксный (prefix) 230
— , приоритет (priority) 230
определение рекурсивное (recursive definition) 214
опровержение по Бету (refutation Beth) 42
откат (backtracking) 186, 210
отношение рекурсивное (recursive relation) 214
отождествление (matching) 206
отрицание безопасное (negation safe) 238
— в ПРОЛОГе (in PROLOG) 233

Отрицание как неудача (by failure) 235
— небезопасное (unsafe) 238
отсечение (cut) 214

ПАРЛОГ (PARLOG) 254
ПАСКАЛЬ (PASCAL) 181
ПВ 145
Переименование (renaming substitution) 99
Переменная (variable) 92, 93
— безымянная (anonymouse) 199
— для чтения (read only) 254
— свободная (free) 96
— свободная для терма в формуле (free for a term in a formula) 99
— связанная (bound) 96
ПНФ 117
подстановка (substitution) 96
— пустая (empty) 97
подтерм (subterm) 95
подформула (subformula) 17, 95
подцель (subgoal) в логике предикатов 105
— дизъюнкта (of a clause) 52
позитивные знания (positive knowledge) 234
поиск пути (finding a path) 210, 212
полное множество (adequate set) 34
Помпея (Pompei) 182
порядок плотный (dense order) 115
последователь (descendant) 65
— непосредственный (immediate) 65
последователь (next node) 132
последовательность высказываний выполнимая (sequence of satisfiable propositions) 65
правило Modus Ponens 44, 47, 99
— введение нового (entering a new) 225
— обобщения (generalization) 99
— программы (of a program) 192, 195
— резолюции (of resolution) 53
предикат (predicate) 92
— встроенный (built-in) 225
— древесное представление (tree structure) 201
— ПРОЛОГа (of PROLOG) 200
— обобщенный (generalized) 204
— полное определение в программе (complete definition in a program) 237
— размещения данных (data management) 225
— составной (compound) 200
предложение (sentence) 96
— выводимое по Бету (Beth-provable) 131
—— из множества предложений 131
— предложение выполнимое (satisfiable) 114
— непротиворечивое (consistent) 114
— опровержимое по Бету (Beth-refutable) 131
— универсальное (universal) 119
предшественник (previous node) 132
префикс (prefix) формулы 117
проблема разрешимости (problem's decision) 154
проверка вхождений (ПВ) (occur check) 145
программа (program) 106
— , взаимодействие (interaction with) 226
— , декларативная интерпретация (declarative interpretation) 193, 209
— , замыкание (completion) 2166 236
— , зацикливание (loops) 216
— логическая интерпретация (logical interpretation) 209
— нормальная (normal) 238
— обновление (updating) 225
— обобщенная (general) 238
— процедурная интерпретация (procedural interpretation) 206, 209
— расслоенная (stratified) 238
— стартифицированная 240
— , структура (structure) 192
— , структура управления (flow) 188
программирование (programming) логическое с ограничениями (constraint logic) 252
— объектно-ориентированное (object-oriented) 191
— функциональное (functional) 191
ПРОЛОГ (PROLOG) 48, 181
— , алфавит (alphabet) 198
— , аппарат вычислений (operational mechanism) 206
— , арифметика (arithmetic) 228
— , атомы (atoms) 199
— , версии (dialects) 242
— конкурентный (concurrent) 254
— , константы (constants) 199
— , контроль типов (type checking) 229
— , метапрограммирование (metaprogramming) 252
— , неравенство (inequality) 229
— , общий унификатор (general unifier) 207
— , объекты (objects) 198
— , оператор (operator) 229
— , отрицание (negation) 233
— , переменные (variables) 199
— , предикат (predicate) 200
— , равенство (equality) 228

ПРОЛОГ, реальный (real) 254
— , резолютивное доказательство (resolution proof) 191
— , символы специальные (special characters) 198
— , список (list) 203
— , унификация (unification) 207
— , функции (functions) 198
— , целевое утверждение (goal) 192, 196
— , число (number) 198
— , чистый (pure) 255
ПРОЛОГ I (PROLOG I) 251
ПРОЛОГ II (PROLOG II) 251
ПРОЛОГ III (PROLOG III) 251
IC-ПРОЛОГ (IC-PROLOG) 252
mu-ПРОЛОГ (mu-PROLOG) 252
nu-ПРОЛОГ (nu PROLOG) 252
Δ-ПРОЛОГ (nu-PROLOG) 252
Протагор (Protagoras) 11
противоречие (contradiction) 21
процедура разрешающая (procedure decision) 154
путь решения (solution path) 212

Расслоенность (stratification) 239
расширение (extension) 113
— неэлементарное (non-elementary) 113
—— интерпретации 113
——— языка 113
— элементарное (elementary) 113
——— интерпретации 113
——— языка 113
резольвента (resolvent) 54
резолюция (resolution) 53
— линейная (linear) 190
рекурсивный базис (bound of recursion) 221
рефлексивность (reflexivity) 102
Робинсон (Robinson) 190

СНФ 120
связки логические (logical connectives) 15, 94
связывание переменных (binding of variables) 206
семантика (semantic) 15
— Крипке (Kripke) 22
— логики высказываний (of propositional logic) 15, 18
семантическая таблица (semantic tableau) 38
—— с конечной степенью ветвления (of a finite degree) 66
— замкнутая 41, 128, 131
— незамкнутая 41
— противоречивая 41, 131
символ (symbol) логики предикатов 93
— предикатный (predicate) 94, 204
— функциональный (functions) 94
симметричность (symmetry) 102
синтаксический анализ (syntactic analysis) 201
системы баз знаний (knowledge base systems) 246
скобки (parentheses) 94
следствие (consequence) в логике предикатов 117
состояние заключительное (final state) 212
— начальное (initial) 212
— промежуточное (intermediate) 212
— пространство (space) 212
список (list)
— , внутреннее представление (internal representation) 205
— , древесная структура (tree structure) 205
— , заголовок (head) 204
— непустой (non-empty) 204
— , обработка (management) 222
— пустой (empty) 204
— , тело (body) 204
— , унификация (unification) 222
— , хвост (tail) 204
степень (degree, arity) предиката 94
стратегия поиска в глубину с возвратом (depth-first strategy) 212

ТУРБО-ПРОЛОГ (TURBO-PROLOG) 252
таблица семантическая (semantic tableau) 38, 41
—— атомарная (atomic) 38
—— замкнутая (complete) 41
—— незамкнутая (incomplete) 41
—— противоречивая (contradictory) 41
—— замкнутая (complete) 128, 131
—— противотечивая (contradictory) 131
таблица истинности (truth table) 24
— сокращенная (short) 27
тавтология (tautology) 21, 100
тело дизъюнкта (body of a clause) 52
— правила (of a rule) 193
теорема (theorem) 47
— Эрбрана (Herbrand) 138
— дедукции (deduction) 46, 104
—— в логике предикатов 104
— компактности (of compactness) 66
—— аксиоматической системы вывода (of axiomatic proofs) 66
—— дедуктивного вывода (of deductions) 66

Теорема компактности исчисления Бета 151
— корректности (of soundness) 61
—— аксиоматического исчисления 154
—— аксиоматической системы вывода (of axiomatic proofs) 66
—— дедуктивного вывода (of deductions) 64
—— исчисления Бета 151
—— исчисления резолюций 151
—— отрицания как неудачи (not as failure) 238
—— резолютивного вывода (of resolution) 66
—— семантических таблиц (of semantic tableaux) 61
— полноты (of completeness) 62
—— аксиоматического исчисления 154
—— аксиоматической системы вывода (of axiomatic proofs) 66
—— дедуктивного вывода (of deductions) 64
—— исчисления Бета 151
—— исчисления резолюций 151
—— резолютивного вывода (of resolution) 66
—— семантических таблиц (of semantic tableaux) 62
теоретико-множественное представление (set-theoretic representation) 51
теория интерпретации (theory of interpretation) 112
— типов 191
терм (term) 94
— логнки предикатов 94
— основной (ground) 96
— атомарный (atomic term) 200
транзитивность (transitivity) 102

Универсум эрбрановский (Herbrand universe) 122
унификатор (unifier) 144
наиболее общий (most general) 144
— общий (general) 143
унификация в ПРОЛОГе (unification in PROLOG) 207
— в логике предикатов 142
условия граничные (boundary conditions) 221
— краевые (marginal) 220

ФОРТРАН (FORTRAN) 186
факт (fact) 53
— в логике предикатов 106
— введение нового (entering a new) 225
— программы (of a program) 192, 194
форма нормальная (normal form) дизъюнктивная (disjunctive) 35
—— конъюнктивная (conjinctive) 36
—— предваренная (prenex) 117
—— сколемовская (Skolem) 120
формула (formula) 94
— замкнутая (closed) 96
— выводимая (provable) 103
—— из множества формул (from a set of formulae) 103
— помеченная 38
— логики предикатов 92, 94
— логически истинная (logically true) 114
— общезначимая (logically true) 114
— помеченная (signed) 38
— элементарная (atomic) 94
Фреге (Frege) 48
функтор (functor) 205
функция сколемовская (Skolem function) 120

Ханойские башни (towers of Hanoi) 232
хвост дизъюнкта (tail of a clause) 52
Хинтикка (Hintikka) 38, 59
Хинтикки лемма (Hintikka's lemma) 59
Хорновский дизъюнкт (Horn clause) 52
— охраняемый (guarded) 254

Целевое утверждение (goal) 193
— в ПРОЛОГе (in PROLOG) 193, 196
— нормальное (normal) 235
цель (goal) в логике предикатов 105
— дизъюнкта (of a clause) 52
— неуспешная (fail) 53, 153
— программы (of a program) 52
—— положительная (to be definite) 52
— успешная (succeed) 53, 153
— неудачная (failed goal) 53

Число (number) 198
— в ПРОЛОГе (in PROLOG) 200

Эвристики (heuristics) экспертной системы (expert system) 246
экспертная система (expert system) 245
— , автоматические системы накопления знаний (automatic knowledge acquisition systems) 247
— , аппарат вывода (inference mechanism) 246
— , база данных (database) 246
— , высокоуровневые языки символьной обработки (high level symbolic processing languages) 247
— инженер базы знаний (data base engineer) 247
— , интерфейс (interface) 246

— оболочка (shell) 246
— обработка знаний (knowledge management) 247
— представление знаний (knowledge representation) 247
Эпикур (Epicurus) 181
Эрбран (Herbrand) 190
Язык арифметики (of arithmetic) 94
— дискриптивный (descriptive) 187
— императивный (imperative) 186
— логики высказываний (of propositional logic) 14
— логики предикатов 93
— реляционный (relational) 253

СОДЕРЖАНИЕ

Предисловие к русскому изданию (*В. А. Садовничий*) 5
Введение 8

Глава 1. **Логика высказываний** 11
§ 1.1. Введение 11
§ 1.2. Язык логики высказываний 14
§ 1.3. Понятие семантики в логике высказываний 18
§ 1.4. Таблицы истинности 24
§ 1.5. Логические следствия и интерпретации 28
§ 1.6. Полнота множеств логических связок и нормальные формы 34
§ 1.7. Семантические таблицы 38
§ 1.8. Аксиоматическая система вывода 44
§ 1.9. Метод резолюций 48
§ 1.10. Корректность и полнота метода семантических таблиц 58
§ 1.11. Дедуктивный вывод из гипотез 63
§ 1.12. Корректность и полнота аксиоматической системы вывода 67
§ 1.13. Корректность и полнота метода резолюций 67
§ 1.14. Упражнения 69

Глава 2. **Логика предикатов** 91
§ 2.1. Введение 91
§ 2.2. Язык логики предикатов 93
§ 2.3. Аксиоматическое основание логики предикатов . . 99
§ 2.4. Система обозначений логического программирования 104
§ 2.5. Интерпретация в логике предикатов 107
§ 2.6. Нормальные формы в логике предикатов 117
§ 2.7. Эрбрановские интерпретации 122
§ 2.8. Метод систематических таблиц 126
§ 2.9. Унификация и резолюция в логике предикатов . . . 142
§ 2.10. Корректность и полнота исчислений логики предикатов 150
§ 2.11. Проблема разрешимости логики предикатов 154
§ 2.12. Упражнения 156

Глава 3. **Логическое программирование: парадигма пролога** 181
§ 3.1. Пролог и логическое программирование 181
3.1.1. Введение 181
3.1.2. Логика и программирование 186

3.1.3. Логическое программирование 189
3.1.4. История развития 190
§ 3.2. Структура программы 192
3.2.1. Элементы программы 192
3.2.2. Факты 104
3.2.3. Правила 195
3.2.4. Запросы 196
§ 3.3. Синтаксис данных 198
3.3.1. Объекты ПРОЛОГа 198
3.3.2. Алфавит ПРОЛОГа 198
3.3.3. Переменные 199
3.3.4. Константы 199
3.3.5. Предикаты 200
3.3.6. Представление предикатов деревьями . . . 202
3.3.7. Списки 203
§ 3.4. Аппарат вычислений 206
3.4.1. Процедура унификации в ПРОЛОГе 206
3.4.2. Вывод и процедура отката 209
3.4.3. Поиск в глубину с возвратом 212
3.4.4. Управление откатом: отсечение 214
3.4.5. Рекурсивные определения в ПРОЛОГе . . . 217
3.4.6. Обработка списков 222
3.4.6.1. Предикат `member` 223
3.4.6.2. Предикат `append` 224
§ 3.5. Встроенные предикаты 225
3.5.1. Предикаты размещения данных 225
3.5.2. Предикаты взаимодействия 226
3.5.3. Равенство в ПРОЛОГе 228
3.5.4. Арифметика в ПРОЛОГе 228
3.5.5. Контроль типов 229
3.5.6. Операторы 229
3.5.7. Ханойские башни 232
§ 3.6. Отрицание в ПРОЛОГе 233
3.6.1. Допущение замкнутости мира и отрицание как неудача 233
3.6.2. Нормальные цели 234
3.6.3. Замыкания программ 236
3.6.4. Нормальные программы и стратификация 238
3.6.5. Предикат *fail* 241
3.6.6. Предикат *not* 242
3.6.7. Немонотонные логики 243
§ 3.7. Экспертные системы 245
3.7.1. Искусственный интеллект 245
3.7.2. Экспертные системы и обработка знаний 245

3.7.3. Экспертная система для диагностики почечных заболеваний 247
§ 3.8. Эволюция логического программирования 251
3.8.1. Редакции ПРОЛОГа 251
3.8.2. Версии ПРОЛОГа 252
3.8.3. ПРОЛОГ и метапрограммирование 252
3.8.4. ПРОЛОГ и параллелизм 253
§ 3.9. ПРОЛОГ и логика предикатов 254
§ 3.10. Упражнения 257

Список литературы 275
Предметный указатель 279