

# ЛОГИКА И КОМПЬЮТЕР

$$\forall_B: \frac{A(\exists x \neg A(x))}{\forall x A(x)}$$

АКАДЕМИЯ НАУК СССР

СЕРИЯ "КИБЕРНЕТИКА —  
НЕОГРАНИЧЕННЫЕ ВОЗМОЖНОСТИ  
И ВОЗМОЖНЫЕ ОГРАНИЧЕНИЯ"

---

Основана в 1963 году

# ЛОГИКА И КОМПЬЮТЕР

МОДЕЛИРОВАНИЕ РАССУЖДЕНИЙ  
И ПРОВЕРКА ПРАВИЛЬНОСТИ  
ПРОГРАММ



МОСКВА "НАУКА"  
1990



Scan AAW

ББК 32.81

Л 69

УДК 510.6:681.3

Авторы:

Н.А. АЛЕШИНА (разд. 5.3), А.М. АНИСОВ (разд. 3.1–3.4),  
П.И. БЫСТРОВ (гл. 1), И.А. ГЕРАСИМОВА (разд. 5.1),  
В.С. МЕСЬКОВ (разд. 5.3), Н.Н. НЕПЕЙВОДА (гл. 2),  
В.А. СМЕРНОВ (гл. 1, 4), В.Н. СТЕБЛЕЦОВА (разд. 3.5),  
В.И. ШАЛАК (разд. 5.2)

Редакционная коллегия:

академик И.М. МАКАРОВ (председатель),  
академик С.В. ЕМЕЛЬЯНОВ (заместитель председателя),  
академик Н.Н. ШЕРЕМЕТЬЕВСКИЙ (заместитель председателя),  
кандидат философских наук С.Н. ГОНШОРЕК (ученый секретарь),  
академик В.Г. АФАНАСЬЕВ,  
академик О.М. БЕЛОЦЕРКОВСКИЙ,  
доктор философских наук Б.В. БИРЮКОВ,  
академик Е.П. ВЕЛИХОВ,  
академик А.А. ВОРОНОВ,  
академик Ю.В. ГУЛЯЕВ,  
академик Н.Н. ЕВТИХИЕВ,  
член-корреспондент АН СССР С.П. КУРДЮМОВ,  
академик В.А. МЕЛЬНИКОВ,  
академик В.С. МИХАЛЕВИЧ,  
академик Н.Н. МОИСЕЕВ,  
член СП СССР В.Д. ПЕКЕЛИС,  
доктор технических наук Д.А. ПОСПЕЛОВ,  
академик А.А. САМАРСКИЙ,  
доктор технических наук И.С. УКОЛОВ,  
академик К.В. ФРОЛОВ,  
доктор физико-математических наук В.В. ШЕННИКОВ

Автор предисловия академик И.М. МАКАРОВ

Научный редактор доктор философских наук Е.Д. СМЕРНОВА

Рецензент член-корреспондент Ю.И. ЖУРАВЛЕВ

Л69

**Логика и компьютер. Моделирование рассуждений и проверка правильности программ / Н.А. Алешина, А.М. Анисов, П.И. Быстров и др. — М.: Наука, 1990. — 240 с., ил. — (Серия "Кибернетика — неограниченные возможности и возможные ограничения").**

ISBN 5-02-007156-0

Логика является одним из важнейших источников развития современной информатики и вычислительной техники. Первые в отечественной литературе для широкого круга читателей детально анализируются логические предпосылки информатики, показывается ключевая роль логики в программировании, в разработке экспертных систем, в исследовании искусственного интеллекта, во внедрении современных информационных технологий.

Для читателей, интересующихся перспективами современной компьютерной революции, в особенности для тех, кто изучает и преподаёт информатику.

Л 1402000000—242  
054(02)—90 48—90 НП

ББК 32.81

ISBN 5—02—007156—0

© Издательство "Наука", 1990

## ПРЕДИСЛОВИЕ

Информатика как новое направление науки и техники оформилось буквально у нас на глазах. Само название этого направления (информатика, "вычислительная наука") закрепилось в различных языках не более 30 лет назад. Однако корни информатики весьма глубоки и разнообразны. Для того чтобы познакомиться с ними, необходимо обратиться к истории и современному развитию целого комплекса фундаментальных и технических наук (от математики и физики до криптологии и техники связи). К числу важнейших источников и постоянных факторов развития информатики относится и логика.

До недавнего времени сведения о логике во многих учебниках информатики занимали очень мало места. Конечно, даже начинающий программист знал, что точное определение понятия алгоритма впервые было найдено в рамках формальных систем математической логики. Однако конкретная программистская практика не требовала серьезного углубления в логику. Общего представления о вкладе логики в основы информатики казалось достаточно.

Между тем в последние годы на стыке логики и программирования возникли новые направления информатики (см.: "Логическое программирование". М.: Мир, 1988). Одним из наиболее известных проектов создания компьютеров пятого поколения предполагается использование логики в качестве основной формальной системы программирования (см.: Дж. Симонс. "ЭВМ пятого поколения: компьютеры 90-х годов". М.: Финансы и статистика, 1985). Специалисты, работаю-



щие в различных областях информатики, проявляют все большее внимание и интерес к логике. Глубокая идейная связь, столь характерная для современного развития логики и информатики, находит свое выражение во многих новых результатах, имеющих первостепенное *практическое* значение (см.: "Язык Пролог в пятом поколении ЭВМ". М.: Мир, 1988).

Логика, одна из древнейших областей научного знания, является существенным общекультурным фактором, и в мире современной науки ее роль важна и многопланова. Естественно, с течением времени изменяется ориентация логических исследований, совершенствуются и обогащаются логические методы, возникают новые тенденции, отвечающие требованиям научно-технического прогресса.

Долгое время, например, логическая наука развивалась как средство решения задач обоснования математики. Начиная с 60-х годов нынешнего столетия, интенсивно начали развиваться разделы логики, ориентированные на глубокую теоретико-познавательную проблематику. Были созданы модальные, временные, доказуемостные логики, логики с не всюду определенными функциями и предикатами, разработана изящная, нашедшая широкое применение логическая семантика возможных миров. Первоначально эти исследования были связаны с философской проблематикой, в частности, временная логика первоначально замышлялась как вспомогательное средство для анализа античных философских текстов. В связи с бурным развитием информатики, программирования и исследованиями по искусственному интеллекту особо значимыми становятся новые аспекты логической науки - она приобретает важное прикладное значение.

Широкое применение компьютеров потребовало соответствующего теоретического обеспечения: разработки специальных языков для баз данных и для

представления знаний, углубленного логического анализа естественных языков. Эти проблемы и стремление реализовать на компьютерах широкий класс интеллектуальных процедур (идеи искусственного интеллекта) поставили перед логической наукой новые задачи. Оказалось, что результаты, которые получены в области формальной логики, и задачи, решаемые ее средствами и методами, находят разнообразные и эффективные приложения во многих областях компьютерной науки. Хорошей иллюстрацией этому является программирование.

Языки программирования, особенно высокого уровня, основываются на тех или иных уточнениях понятия алгоритма (машины Тьюринга и Поста,  $\lambda$ -исчисления, рекурсивные функции и др.). Это позволило начать систематическое логическое исследование семантики таких языков. Например, интуиционистская логика в форме теории типов Мартин-Лефа является достаточно богатой и полной теорией процедур конструирования, спецификации и верификации программ.

Временная логика, первоначально ориентированная на анализ традиционных философских проблем, нашла применение при спецификации и верификации параллельных и конкурирующих программ. Определенные модификации темпоральной логики успешно используются для спецификации программных циклов. Модальная логика и идеи семантики возможных миров лежат в основе динамических логик, ориентированных непосредственно на верификацию и синтез программ.

Еще один из плодотворных путей контакта логики с программированием – область доказуемости программирования. Общая идея здесь, как говорится, лежит на поверхности: если можно конструктивно, интуиционистски доказать, что существуют объекты, удовлетворяющие некоторому данному условию, то,

построив доказательство, можно построить по нему и программу вычисления соответствующей функции. Сходство между структурами логических доказательств и программ позволяет реализовать идею логического программирования. Примером такой реализации является широко известный ныне язык Пролог.

Установление и исследование связи между программами и доказательствами вызывает особый интерес к логическому исследованию процедур поиска доказательства и, следовательно, к проблеме автоматического доказательства теорем. Сейчас это уже достаточно зрелая область прикладных логических исследований, хотя в основном она пока ограничена лишь классической логикой первого порядка и несколькими системами с модальными операторами.

Проблемы искусственного интеллекта послужили непосредственной причиной логического анализа так называемых немонотонных рассуждений различного типа. В случае обычных или монотонных рассуждений мы полагаем, что добавление новой информации не отменяет, не делает неверными следствия, полученные ранее. Однако в практике рассуждений мы нередко принимаем допущения о некоторой полноте исходной информации. Добавление новой информации к исходной отменяет это допущение, и то, что ранее принималось как следствие, может не быть таковым при дополнительной информации. Это лишь один из многих типов немонотонных рассуждений, названных Дж. Маккарти очерчиванием. Логический анализ — новая область логических исследований, появившаяся в результате разработки теоретических проблем искусственного интеллекта, имеющая, однако, определенную связь с интенсивно исследуемой релевантной логикой.

В качестве очень интересных с прикладной точки зрения ветвей современной логики следует указать

эпистемическую логику, которая применяется при анализе распределенных, многопроцессорных систем, и паранепротиворечивую логику. Особенность применения последней состоит в следующем. Казалось бы, формальная логика изгоняет противоречие. Однако если мы имеем противоречивые исходные данные, то отсюда не обязательно следует все, что угодно (как в классической логике). Исследование таких экзотических паранепротиворечивых систем, в которых могут приниматься некоторые противоречия, оказалось важным, в частности, для работы с массивами данных и знаний в условиях неполной или противоречивой информации.

К числу идей, важных для приложений, относится и учет ограниченности ресурсов, относительный характер знания, возможность коррекции знания. Приложение логики к компьютерной науке, информатике и проблемам искусственного интеллекта требуют учета финитности объектов, ограниченности ресурсов, динамического характера знания. Это достаточно трудные проблемы. находящие сейчас частичные решения средствами логики.

В предлагаемой книге мы, естественно, могли остановиться только на небольшом круге вопросов. Первая глава – вводная. В ней излагаются основные понятия логики предикатов первого порядка, дается теоретико-множественная семантика. Изложены различные типы формализации логического исследования. Следует обратить внимание на не совсем стандартное изложение техники натурального вывода. При этом сознательно не дается никаких доказательств основных методологических результатов, они только формулируются. Необходимые доказательства можно найти в стандартных учебниках.

Вторая глава знакомит читателя с ролью логики в программировании. Наряду с известными результатами

в ней содержатся и оригинальные идеи и постановки проблемы.

В третьей главе изложены основные результаты, относящиеся к верификации программ. Если первые четыре параграфа содержат достаточно популярное введение в эту область, то пятый носит более специальный и в определенной степени оригинальный характер. Он посвящен приложению временных логик к анализу свойств программ, особенно программ с параллельными процессами.

Четвертая глава посвящена проблемам поиска доказательств. Стандартные методы поиска доказательств, в частности метод резолюций, изложены очень кратко. Оригинальными являются процедуры поиска натурального вывода в режиме диалога, реализованные на персональном компьютере.

В последней, пятой, главе речь идет о приложениях логики к проблемам искусственного интеллекта. Она касается эпистемических логик и распределенных систем, логик с ограниченными ресурсами, правдоподобных рассуждений.

Книга будет полезна широкому кругу читателей, интересующихся проблемами взаимосвязи логики и информатики, логики и компьютерных наук, поможет уяснить роль логических исследований в современном мире, охваченном интенсивным процессом компьютеризации.

*Академик И.М.Макаров*

## Глава 1. КЛАССИЧЕСКАЯ ПЕРВОПОРЯДКОВАЯ ЛОГИКА

### 1.1. Что такое правильное рассуждение? Посылки и заключения

Возникновение науки было таким фактом истории человечества, которому невозможно найти что-либо равное по значимости как для самих людей, так и для окружающей их природы. Именно выделение, формирование и развитие научного знания стало принципиально новой ступенью в процессе освоения человеком объективной реальности. Вообще говоря, это бесспорно и на сегодняшний день даже тривиально. Однако нетривиальным является вопрос о том, что именно позволяет нам говорить, что с такого-то времени донаучное познание уступило свой приоритет научному. Какие качественные изменения методов и форм мышления и познания можно считать собственно "рождением" науки? В.И.Вернадский датирует возникновение науки открытием Александрийского музея. И дело здесь, конечно, не в том, чтобы установить, какие именно существенные, глубинные изменения в интеллектуальной деятельности древних знаменовали собой зарождение науки. Результатом таких изменений было, в частности, и открытие знаменитого музея в Александрии. Однако этому предшествовало исключительно важное дело, начатое греческими мыслителями и успешно выполненное Аристотелем. Аристотель впервые выделил, кодифицировал и систематизировал общезначимые (в некотором смысле универсальные для человеческого мышления) способы рассуждений. Именно идея возможности выявления и кодификации общезначимых способов рассуждений является необходимой предпосылкой формирования науч-

ного мышления, возникновения науки как таковой. Применение таких способов с необходимостью включено в основы той специфической деятельности человека, которую можно назвать научной.

Любопытно отметить, что после падения античной цивилизации первое, что было восстановлено из античной науки, — это логика Аристотеля. Известно, что сначала христианство отрицательно относилось ко всей античной науке, и признание ее началось именно с того, что были признаны значимыми в рамках нового мировоззрения первые семь глав аристотелевских "Аналитик". Если обратиться к эпохе Возрождения, к истокам науки нового времени, нетрудно установить, что и в этом случае первыми восстанавливались и использовались именно разработанные в античности логические методы. С этого начиналась философия Декарта и других мыслителей, начиналась вся новая наука. Таким образом, вопрос об общезначимых, правильных способах рассуждений является исключительно важным, одним из центральных вопросов, связанных с возникновением, развитием и функционированием научного знания. В первую очередь этим вопросом определяются задачи логики.

Что же такое общезначимое правильное рассуждение? Начнем с примера. Существует предание, что Александрийскую библиотеку сжег калиф Омар. Но прежде чем сжечь ее, он обосновал свое деяние с помощью следующего рассуждения: "Если ваши книги согласны с Кораном, то они излишни. Если они не согласны с Кораном, то они вредны. Но вредные или излишние книги следует уничтожить. Значит, ваши книги следует уничтожить". Рассуждение есть переход от некоторых предложений, утверждений, называемых посылками, к утверждению, которое называется заключением. В приведенном примере первые три предложения являются посылками, четвертое — заключением.

ем. Допустим, нас не интересует ни то, что в действительности не Омар сжег библиотеку, ни то, что он вообще так не рассуждал, ни истинность или ложность самого по себе заключения. Но зададимся вопросом, правильно ли независимо от всего этого само рассуждение? Как будет видно из дальнейшего, такое рассуждение с логической точки зрения совершенно правильно. Другое дело, что не истинны посылки, которые в нем используются, но в данном случае нас интересует только правильность самого рассуждения, независимо от истинности посылок и заключения.

Возьмем другой пример. Предположим, некто строит следующее рассуждение: "Дикари раскрашивают свое тело. Некоторые современные женщины раскрашивают свое тело. Следовательно, некоторые современные женщины - дикари". Правильно ли это рассуждение? Нетрудно установить, что данное рассуждение неправильно, несмотря на то, что используемые посылки и сделанное из них заключение можно признать истинными.

Итак, первая задача логики - выявить, какие способы рассуждения правильные, а какие нет. Здесь необходимо отметить, что, анализируя рассуждения, следует выделять и отличать два их типа. Есть рассуждения достоверные (их называют также дедуктивными) и рассуждения правдоподобные, или вероятностные. Заключение, полученное с помощью дедуктивного рассуждения, достоверно, оно носит необходимый характер, в то время как результаты правдоподобных рассуждений являются лишь гипотетическими, верными с некоторой степенью вероятности. Критерием правильности дедуктивных рассуждений служит следующий общий принцип: правильное рассуждение всегда гарантирует истинность заключения при условии, что истинны посылки. И задача логики состоит именно в том, чтобы по возможности наиболее



полно описать и исследовать те способы рассуждений, которые являются правильными.

## 1.2. Формальный характер рассуждений.

### Предложения и высказывания

Характерной чертой дедуктивных рассуждений является их формальный характер<sup>1</sup>. Это означает, что правильность рассуждения зависит от структуры, логической формы посылок и заключения. Поэтому необходимо выяснить, что мы будем понимать под логической формой составных частей рассуждения. Прежде всего нужно подчеркнуть, что посылки и заключения являются предложениями. Однако не всякое предложение может выступать в качестве посылки или заключения.

Обычно в лингвистике выделяют повествовательные, повелительные и вопросительные предложения. Мы, в первую очередь, будем выделять тип повествовательных предложений, в которых достаточно определенно формулируется некоторая информация о предметах, явлениях, событиях, положениях дел и т.д. Такие предложения будем называть высказываниями. Высказывание – это предложение, в котором содержится (дается, сообщается) какая-то информация о предметах, явлениях или положениях дел и которое может быть оценено как истинное или ложное. Например, предложение "Саша К. изучает логику и любит

---

<sup>1</sup> Следует отметить, что здесь слово "формальный" нельзя понимать в противопоставлении с терминами "содержательный", "имеющий смысл", "реальный" и т.п. Такое понимание отнюдь не безобидно, зачастую оно приводит к неправильному восприятию, некорректному изложению формальной логики, скепсису и необоснованной критике по отношению к ее методам и возможностям. Понятие "формальный" в данном случае не более "вредно", чем само понятие формы, без обращения к которому выделение способов общезначимых рассуждений вообще невозможно.

активный отдых" является высказыванием, а предложение "Любите животных!" – нет. Первое истинно, если Саша К. (о котором идет речь) действительно занимается изучением логики, одновременно являясь любителем активного отдыха, и ложно в противном случае. Об истинности или ложности второго предложения говорить бессмысленно.

Можно выделить другой тип предложений – предложения, в которых формулируются задачи, проблемы. К такому типу относятся, например, следующие предложения: "Постройте треугольник с основанием  $a$  и высотой  $b$ "; "Докажите средствами исчисления высказываний, что если  $p$  имплицитует  $q$ , то  $\neg q$  имплицитует  $\neg p$ ". Это предложения повелительного типа, в которых выражается требование получить некоторый результат.

Мы будем отличать также предложения, в которых формулируются четкие предписания совершить определенные действия в определенном порядке. Например: "Разделить 18 на 3, результат умножить на 4 и к последнему результату прибавить результат вычитания 12 из 19". Такого рода предложения описывают некоторую программу действий. К более подробному анализу задач и программ мы обратимся ниже. Пока необходимо лишь четко зафиксировать отличие задач и программ от высказываний.

Выше было отмечено, что отличительная черта высказываний состоит в том, что они могут быть истинными или ложными. Логика действительно опирается на теоретико-познавательное понятие истинности, которое в классической логике трактуется следующим образом: высказывание называется истинным тогда и только тогда, когда оно соответствует описываемому положению дел; в противном случае оно называется ложным. В логической семантике понятие истинности находит свое уточнение и существенным образом ис-

пользуется для обоснования правильности определенных способов рассуждений.

Рассуждения могут проводиться на различных языках. Предложения и высказывания, описывающие некоторые положения дел, могут быть сформулированы в естественных языках. Однако следует отметить, что, рассуждая на естественном языке в математике и вообще в науке, мы всегда применяем некоторые стандартные способы записи и преобразования информации. Впервые это обстоятельство было использовано и нашло четкое выражение в логике Аристотеля. Он рассматривал в качестве стандартных форм, в которых выражается информация, четыре вида категорических утверждений<sup>2</sup>:

"Все  $S$  суть  $P$ ",

"Некоторые  $S$  суть  $P$ ",

"Все  $S$  не суть  $P$ ",

"Некоторые  $S$  не есть  $P$ ".

Например: "Все сороки – птицы" – общеутвердительное высказывание; "Некоторые люди – хитрецы" – частноутвердительное высказывание; "Все лошади не являются хищниками" – общеотрицательное высказывание; "Некоторые мужчины не знают логики" – частноотрицательное высказывание. Аристотелем была разработана теория рассуждений (умозаключений), в которой посылки и заключения формулируются в виде перечисленных категорических утверждений. Она получила название ассерторической силлогистики. Что-

---

<sup>2</sup> Важно обратить внимание на различие между грамматической и логической формами предложения. Иногда предложение может иметь, в частности, вид категорического высказывания, но на самом деле обладать совершенно другой логической структурой. Например, в высказывании "Все участники прогулки разбежались в разные стороны" слово "все" употребляется в смысле, отличном от того стандартного смысла, в котором оно используется при формулировке аристотелевских категорических утверждений.

бы проследить правильность рассуждения в естественном языке средствами такой теории, необходимо все суждения представить в стандартной форме — в форме простых категорических высказываний.

По мере развития логики были учтены и использованы для стандартизации рассуждений другие формы

По мере развития логики были учтены и использованы для стандартизации рассуждений другие формы высказываний, и правильные рассуждения основывались уже на иной структуре посылок и заключений. Так, стоики включили в рассмотрение умозаключения, базирующиеся на структуре сложных высказываний. Под сложными высказываниями имеются в виду такие высказывания, которые образуются из других высказываний с помощью логических союзов "и", "или", "если ..., то ..." и др. Практически стоики систематизировали те способы рассуждений, которые ныне описываются логикой высказываний. В дальнейшем логика стала исследовать способы рассуждений, основанные на более сложной структуре высказываний. Прежде всего во внимание был принят факт, что высказывание далеко не всегда является или простым категорическим, состоящим из субъекта *S*, предиката *P* и соединяющей их связки, или сложным, состоящим из простых высказываний, соединенных логическими союзами. Например, в высказываниях "Петр старше Андрея", "12 больше, чем 7" выражается информация о том, что между Петром и Андреем, между числами 7 и 12 существуют некоторые отношения. Учет того, что в высказываниях может содержаться информация об отношениях между объектами, свойствами, положениями дел, позволяет выделить более общий тип высказываний, нежели высказывания о присущности или неприсущности свойств объектам. В конце XIX века была построена строгая теория рассуждений, включающих и простые, и сложные высказывания, в которых

выражена информация о свойствах и отношениях. Одновременно, что является весьма существенным, в логику были введены кванторы общности  $\forall$  ("все...", "всякий...") и существования  $\exists$  ("некоторые...", "некоторый..."), применимые не только в категорических высказываниях, но и в высказываниях об отношениях.

Разумеется, процесс логической стандартизации рассуждений, проводимых на естественном языке, на этом не завершился. Обогащение выразительных возможностей логики продолжалось: были приняты во внимание высказывания, выражающие необходимость, возможность, случайность (так называемые модальные суждения), утверждения, включающие факторы времени, изменения, знания, и многие другие виды высказываний.

Кардинальный сдвиг в анализе стандартных форм рассуждений произошел тогда, когда для построения логической теории был применен метод построения формальных систем с помощью специальных символических языков. Одной из хорошо разработанных и широко используемых таких систем или способов стандартизации высказываний является первопорядковая логика предикатов. В исчислении предикатов в наиболее завершенном на сегодняшний день виде содержится анализ тех правильных форм рассуждений, которые основаны на структуре различных по сложности высказываний о свойствах и отношениях.

### 1.3. Язык первопорядковой логики предикатов

Здесь мы дадим одно из возможных описаний языка первопорядковой логики (функционального исчисления первого порядка). Язык первопорядковой логики предикатов состоит из знаковых выражений нескольких типов: индивидуальных, функциональных, предикатных, логических и технических.

*Индивидуальные имена или индивидуальные константы.* Это простейшие по структуре и своему назначению знаки. Индивидуальные имена (константы) есть выражения, которые обозначают объекты, индивиды. В качестве индивидуальных констант будем использовать малые буквы начала латинского алфавита (возможно, с индексами)  $a, b, c, \dots, a_1, b_1, c_1, \dots$

Следует четко различать сам объект и его имя. Если объектом является выражение языка, то в качестве его имени используется само это выражение, взятое в кавычки. Например, корректными будут фразы: «"a" есть индивидуальная константа», «Слово "мама" состоит из четырех букв». Однако возможно автономное употребление имен, когда в качестве имени используется само данное выражение. В этом случае можно сказать: "a есть индивидуальная константа". Здесь имеется в виду, что речь идет о самой букве, а не об объекте, который она именует. Ниже мы будем широко применять автономное употребление имен, если контекст четко позволяет установить, идет ли речь об имени или объекте, который оно обозначает<sup>3</sup>.

*Индивидуальные переменные* – это выражения, которые сами по себе не обозначают объекты, но каждому из которых сопоставляется некоторая область объектов. В данном случае говорят, что переменная пробегает по области (множеству) объектов. В любом выражении, содержащем индивидуальную переменную, эту переменную можно замещать именами объектов из некоторой рассматриваемой области. Для ясности здесь нужно сделать следующую оговорку. Иногда в литературе пишут, что вместо всякого вхождения индивидуальной переменной, пробегающей множество объектов, под-

---

<sup>3</sup> В дальнейшем мы будем автономно употреблять не только индивидуальные константы, но также индивидуальные переменные, предикаты и другие знаки используемого языка.

ставляются индивиды, объекты из данной области. Строго говоря, это неверно. Если, например, переменная пробегает множество геометрических фигур или сторонников перестройки, то, естественно, вместо нее мы подставляем не сами эти фигуры как объекты и не прогрессивных деятелей как конкретных индивидов. Вхождение переменной мы замещаем выражениями, именуемыми объектами. В качестве индивидуальных переменных мы будем использовать малые буквы конца латинского алфавита (возможно, с индексами)  $x, y, z, u, w, \dots, x_1, y_1, z_1, u_1, w_1, \dots$

*Функциональные константы* – это выражения, которые именуют функции, сопоставляющие  $k$ -кам индивидов некоторый индивид. Такие функции иногда называют предметными функциями. Например, если область объектов составляют числа, то предметной функцией на этой области будет сложение – функция, сопоставляющая паре чисел некоторое число. Функциональной константой называют знак, который обозначает саму функцию. В данном примере это  $+$ . Предметные функции могут быть одно-, двух- и  $k$ -местными. Можно рассматривать также нульместные функции. Это есть не что иное, как индивидуальные константы. Другими словами, индивидуальная константа есть нульместная функция, которая сопоставляет пустой последовательности объектов некоторый объект, т.е. именует этот объект. В качестве функциональных констант будем использовать символы  $f_1, f_2, f_3, \dots$ . Для обозначения местности предметных функций применяются верхние индексы:  $f_1, f_2, \dots$  – одноместные,  $f_1^2, f_2^2, \dots$  – двухместные,  $f_1^k, f_2^k, \dots$  –  $k$ -местные. Будем говорить, что  $k$ -местные функциональные константы имеют ранг  $k$ .

*Предикатные константы* – это выражения, обознача-

чающие свойства и отношения. Свойства и отношения называют предикатами. Предикаты могут быть одно-, двух- и вообще  $n$ -местными. Как обычно, одноместные отношения мы называем свойствами, например: красивый, высокий, холодный и т.п. В предложениях "Лондон севернее Конотоп", "Мать дает сыну касторку" слово "севернее" обозначает двухместное отношение, а слово "дает" — трехместное. В качестве предикатных констант ниже будут применяться буквы  $P, Q, R, \dots$ . Обычно из контекста ясно, какова местность предиката, но в случае необходимости можно ставить верхние индексы:  $P^2, Q^3, R^4, \dots$ ;  $n$ -местные предикатные константы будем считать константами ранга  $n$ .

Перечисленные типы знаков, исключая индивидуальные переменные, обычно называют нелогическими или дескриптивными знаками языка. К логическим знакам относятся индивидуальные переменные, логические связки и кванторы.

Логические связки, или пропозициональные константы служат для образования из одних формальных выражений языка, атомарных или сложных, других, более сложных выражений этого языка. Мы будем использовать обычные логические связки:  $\&$  (конъюнкция),  $\vee$  (дизъюнкция),  $\supset$  (импликация) и  $\neg$  (отрицание).

Кванторы — это знаки, которые в сочетании с переменными дают возможность выразить утверждения о том, что для всех или некоторых индивидов выполняется определенное условие. Знак квантора всеобщности  $\forall$  (перевернутое  $A$ ) возник как сокращение английского слова *All* ("все"), а знак квантора существования  $\exists$  (перевернутое  $E$ ) — как сокращение английского слова *Exists* ("существует").

И, наконец, последний тип знаков — технические знаки:  $), ($  — скобки правая, левая и запятая, с по-



мощью которых выражение однозначно расчленяется на составляющие части (подвыражения).

Выражения языка исчисления предикатов суть последовательности перечисленных знаков, которые строятся согласно определенным правилам. Будем говорить, что знак  $z$  входит, или имеет вхождение в выражение, если это выражение имеет вид  $XzY$ , где  $X$  и  $Y$  – последовательности знаков, возможно, пустые. Точно так же можно говорить о вхождении одного выражения в другое. Следует подчеркнуть, что знак или последовательность знаков может иметь не одно, а несколько вхождений в одно и то же выражение. Рассмотрим, например, выражение:

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

В нем имеется по два вхождения каждого из логических знаков  $\forall$  и  $\supset$ , два вхождения знака  $P$  и три вхождения одного и того же знака  $x$ . Если имеется в виду какое-то конкретное вхождение, то будем говорить о *фиксированном вхождении* данного знака или выражения в некоторое выражение.

Отметим, что наряду с функциональными и предикатными константами в языке исчисления предикатов могут использоваться функциональные и предикатные переменные соответствующих рангов, а также пропозициональные переменные. В логических языках первого порядка используются только кванторы вместе с индивидуальными переменными. Квантификация по функциональным, предикатным и пропозициональным переменным приводит к исчислению предикатов второго порядка. То, что в данном случае мы не вводим в язык функциональные, предикатные или пропозициональные переменные, не нарушает общности рассмотрения.

Теперь, используя введенные в употребление знаки, можно дать строгое систематическое описание языка исчисления предикатов первого порядка. При

этом для чисто технического удобства мы будем использовать два вида индивидуальных переменных:

$u, v, w, u_1, v_1, w_1, \dots$  — свободные индивидуальные переменные,

$x, y, z, x_1, y_1, z_1, \dots$  — связанные индивидуальные переменные.

С помощью индуктивного определения введем понятие квазитерма:

1. Индивидуальные константы и индивидуальные переменные суть квазитермы.

2. Если  $f$  есть  $k$ -местная функциональная константа и  $t_1, \dots, t_k$  — квазитермы, не обязательно различные, то  $f(t_1, \dots, t_k)$  есть квазитермы.

3. Никакое другое выражение, кроме указанных в пп. 1 и 2, не есть квазитерм.

Последний пункт определения носит косвенный характер, и в дальнейшем в формулировках индуктивных определений мы будем его опускать.

Квазитерм, не содержащий вхождений связанных индивидуальных переменных, есть *терм*.

Теперь индуктивным определением введем понятие *квазиформулы*:

1. Если  $P$  есть  $k$ -местная предикатная константа и  $t_1, \dots, t_k$  — квазитермы, то  $P(t_1, \dots, t_k)$  есть квазиформула.

2. Если  $A$  и  $B$  суть квазиформулы, то  $(A \& B)$ ,  $(A \vee B)$ ,  $(A \supset B)$  и  $\neg A$  суть квазиформулы.

3. Если  $A$  есть квазиформула и  $x$  — связанная индивидуальная переменная, то  $\forall x A$  и  $\exists x A$  суть квазиформулы.

Будем говорить, что вхождение некоторой переменной в квазиформулу  $B$  находится в области действия квантора по  $x$ , если это есть вхождение в подформулу формулы  $B$  вида  $\forall x A$  или  $\exists x A$ .

Вхождение связанной переменной  $x$  в квазиформулу  $A$  будем называть *свободным*, если оно не находится

в области действия квантора по  $x$ , или *связанным* в противном случае.

Под *подстановкой* вместо свободной переменной  $w$  квазитерма  $t$  в выражение  $A$  будем понимать замещение каждого вхождения свободной переменной  $w$  в  $A$  на квазитерм  $t$ . Операцию подстановки будем обозначать  $F_t^w A$ .

*Формулой* называется квазиформула, не содержащая свободных вхождений связанных переменных. Формула без вхождений свободных переменных называется замкнутой формулой, или предложением<sup>4</sup>.

Необходимо также ввести понятие *правильной подстановки* вместо свободной переменной квазитерма  $t$  в квазиформулу  $A$ . Подстановка  $F_t^w A$  правильна, если и только если ни одно вхождение переменной  $w$  в квазиформулу  $A$  не находится в области действия кванторов по связанным переменным, входящим в квазитерм  $t$ .

**Пример.** Пусть  $t$  есть квазитерм вида  $f(u, y)$ , квазиформула  $A$  имеет вид  $\forall z(Q(w, z) \& P(y))$ , а квазиформула  $B$  имеет вид  $\forall z \exists y(Q(w, z) \supset P(y))$ . В этом случае подстановка  $F_t^w A$  правильна, а подстановка  $F_t^w B$  неправильна.

Очевидно, что подстановка терма всегда правильна. Так, если в приведенном примере  $t$  есть терм, скажем, вида  $f(u, v)$ , то подстановки  $F_t^w A$  и  $F_t^w B$  правильны.

Отметим, что каждую квазиформулу можно рассматривать как результат правильной подстановки связанных переменных вместо свободных. Поэтому всякую формулу вида  $\forall x A$  или  $\exists x A$  можно представить в виде  $F_x^w A$ , где  $A$  есть формула и  $F_x^w A$  — правильная подстановка.

<sup>4</sup> Здесь термин "предложение" используется не в грамматическом, а в некотором техническом смысле, т.е. как синоним термина "замкнутая формула".

Описание языка первопорядкового исчисления предикатов на этом заканчивается. В данном описании (а также в дальнейшем) большие буквы начала латинского алфавита  $A, B, C, \dots$  используются в качестве синтаксических переменных, пробегающих по формулам (квазиформулам), а буквы  $t_1, t_2, \dots, t_R$  — в качестве синтаксических переменных, пробегающих по термам (квазистермам). Принимаются также обычные соглашения об экономии скобок: опускаются внешние скобки формул, опускаются также скобки исходя из того, что логические знаки упорядочены по силе следующим образом. В первую группу входят  $\forall, \exists, \neg$ , во вторую —  $\wedge$  и  $\vee$ , в третью —  $\supset$ .

#### 1.4. Интерпретация языка логики предикатов

Формальные выражения логического языка имеют смысл только в том случае, если имеется какая-то интерпретация входящих в них знаков, т.е. построена некоторая семантика данного языка. Один из способов построения семантики состоит в том, что выделяется некоторое непустое множество нелингвистических объектов, называемое областью интерпретации, и определяется функция, сопоставляющая выражениям языка объекты, множества объектов или отношения между объектами области интерпретации. На этой основе определяются важные понятия выполнимости, истинности и общезначимости формул рассматриваемого языка.

Дадим строгое описание семантики прикладного языка первопорядковой логики предикатов, не выходя за рамки теоретико-множественных понятий. Пусть  $U$  есть некоторое непустое множество объектов. Интерпретация  $I$  на множество  $U$  есть функция, сопоставляющая:

индивидуальной константе  $a$  — элемент множества  $U$ , т.е.  $I(a) \in U$ ;

$k$ -местной функциональной константе  $f_i$  - функцию с областью определения  $U^k$  и областью  $U$ , т.е.

$$I(f_i) = U^k \rightarrow U;$$

$n$ -местной предикатной константе - подмножество  $n$ -ок, т.е. некоторый элемент из множества всех подмножеств  $U^n$ , или  $I(P_i^n) \in \mathcal{U}(U^k)$ .

Здесь  $U^k$ ,  $U^n$  означает декартово произведение  $U$  самого на себя  $k$  и  $n$  раз соответственно, а  $\mathcal{U}(U^k)$  - множество всех подмножеств  $U^k$ .

Возможной реализацией  $\mathcal{M}$  прикладного языка назовем пару вида  $\langle U, I(\mathcal{A}) \rangle$ , где  $\mathcal{A}$  - словарь этого языка, т.е. множество нелогических знаков, входящих в данный язык.

Пусть  $\{t\}$  есть список всех свободных переменных, входящих в терм  $t$ , а  $g$  - функция, приписывающая всякой индивидуальной переменной из  $t$  значение из  $U$ . Индуктивным определением введем понятие значения термина  $t$  в возможной реализации  $\mathcal{M}$  при приписывании  $g$ , обозначив это понятие  $V(t, g)$ :

1. Если  $t$  есть  $a_i$ , то  $V(t, g) = I(a_i)$ .

2. Если  $t$  есть  $w_i$ , то  $V(t, g) = g(w_i)$ .

3. Если  $t$  есть  $f^k(t_1, \dots, t_k)$ , то

$$V(f^k(t_1, \dots, t_k)) = (I(f^k))(V(t_1, g), \dots, V(t_k, g)).$$

Основным понятием описываемой семантики является понятие выполнимости. Пусть имеется некоторый набор значений  $\varphi(\mathcal{A})$ , приписанный функцией  $\varphi$  всем свободным переменным, входящим в формулу  $A$ . В этом случае будем говорить " $\varphi(\mathcal{A})$  выполняет формулу  $A$  в возможной реализации  $\mathcal{M}$ " (более кратко: " $\varphi$  выполняет  $A$  в  $\mathcal{M}$ "), т.е.  $\models_{\mathcal{M}}^{\varphi} A$ , или " $\varphi$  не выполняет  $A$  в  $\mathcal{M}$ ", т.е.  $\not\models_{\mathcal{M}}^{\varphi} A$ . Определим понятие " $\varphi$  выполняет  $A$  в  $\mathcal{M}$ " индукцией по построению формулы:

1.  $\models_{\mathcal{M}}^{\varphi} P(t_1, \dots, t_k)$ , если и только если

$$V(t_1, \varphi), \dots, V(t_k, \varphi) \in I(P).$$

2.  $\models_{\mathcal{M}}^{\varphi} \neg A$ , если и только если  $\not\models_{\mathcal{M}}^{\varphi} A$ .

3.  $\models_M^{\varphi} A \& B$ , если и только если  $\models_M^{\varphi} A$  и  $\models_M^{\varphi} B$ .

4.  $\models_M^{\varphi} A \vee B$ , если и только если  $\models_M^{\varphi} A$  или  $\models_M^{\varphi} B$ .

5.  $\models_M^{\varphi} A \supset B$ , если и только если  $\not\models_M^{\varphi} A$  или  $\models_M^{\varphi} B$ .

6.  $\models_M^{\varphi} \forall x F_x^w A$ , если и только если для любой функции  $\varphi'$ , возможно, отличающейся от  $\varphi$  только приписыванием значений некоторой переменной  $w$ , имеет место  $\models_M^{\varphi'} A$ .

7.  $\models_M^{\varphi} \exists x F_x^w A$ , если и только если для некоторой функции  $\varphi'$ , возможно, отличающейся от  $\varphi$  приписыванием значений переменной  $w$ , имеет место  $\models_M^{\varphi'} A$ .

Формула  $A$  называется *общезначимой* в  $\mathcal{M}$ , если любая функция выполняет  $A$  в  $\mathcal{M}$ . Символически:  $\models_{\mathcal{M}} A$ , если  $\forall \varphi \models_M^{\varphi} A$ .

В том случае, если  $A$  – предложение, т.е. формула, не содержащая вхождений свободных переменных, будем говорить, что  $A$  *истинна* (в возможной реализации  $\mathcal{M}$ ) тогда и только тогда, когда  $\models_{\mathcal{M}} A$ , т.е. любая функция выполняет  $A$  в  $\mathcal{M}$ . Некоторую возможную реализацию  $\mathcal{M}$  называют *моделью* формулы  $A$  в том и только в том случае, когда  $A$  общезначима в  $\mathcal{M}$ . Выражение " $\mathcal{M}$  является моделью формулы  $A$ " будем записывать:  $\mathcal{M} \models A$ . Понятие модели некоторого предложения естественным образом обобщается до понятия модели множества предложений. Возможная реализация  $\mathcal{M}$  называется моделью множества предложений  $S$  тогда и только тогда, когда для всякого  $A \in S$  имеет место  $\models_{\mathcal{M}} A$ .

Будем говорить, что формула  $A$  *общезначима*, если и только если в каждой возможной реализации  $\mathcal{M}$  и для любой функции приписывания  $\varphi$  имеет место  $\models_M^{\varphi} A$ . Утверждение об общезначимости  $A$  будем символически записывать  $\models A$ .

Наряду с понятием общезначимости (логической истинности) в логике играет важную роль другое,

тесно связанное с ним семантическое понятие *логического следования*.

Формула  $B$  *логически следует* из формулы  $A$  тогда и только тогда, когда во всякой возможной реализации  $\mathcal{M}$  и для всякой функции приписывания  $\varphi$ , если  $\models_{\mathcal{M}}^{\varphi} A$ , то  $\models_{\mathcal{M}}^{\varphi} B$ . Символически отношение логического следования будем записывать как  $A \models B$ .

Формула  $A$  логически следует из множества формул  $\Gamma$  тогда и только тогда, когда для всякой возможной реализации  $\mathcal{M}$  и любой функции приписывания  $\varphi$ , если  $\varphi$  выполняет в  $\mathcal{M}$  каждую формулу из  $\Gamma$ , то  $\varphi$  выполняет в  $\mathcal{M}$  формулу  $A$ . Символически:  $\Gamma \models A$ .

Между двумя множествами формул  $\Gamma$  и  $\Theta$  имеет место отношение логического следования  $\Gamma \models \Theta$  тогда и только тогда, когда для всякой возможной реализации  $\mathcal{M}$  и всякой функции приписывания  $\varphi$ , если  $\varphi$  выполняет в  $\mathcal{M}$  каждую формулу из  $\Gamma$ , то  $\varphi$  выполняет в  $\mathcal{M}$  по крайней мере одну из формул из  $\Theta$ .

Формула  $A$  общезначима в том и только в том случае, когда  $A$  логически следует из пустого множества формул.

От семантического понятия логического следования необходимо отличать два других, более слабых. Так, для случая отношений между двумя формулами мы имеем следующие определения этих понятий.

Формула  $B$  *следует* из формулы  $A$  тогда и только тогда, когда для всякой возможной реализации  $\mathcal{M}$ , если любая функция приписывания  $\varphi$  выполняет  $A$  в  $\mathcal{M}$ , то любая функция приписывания  $\varphi$  выполняет и  $B$  в  $\mathcal{M}$ .

Формула  $B$  *следует относительно общезначимости* из формулы  $A$  тогда и только тогда, когда для всякой возможной реализации  $\mathcal{M}$  и любой функции приписывания  $\varphi$  последняя выполняет  $B$  в  $\mathcal{M}$ , если для всякой возможной реализации  $\mathcal{M}$  и любой функции приписывания  $\varphi$  она выполняет  $A$  в  $\mathcal{M}$ .

Отметим, что если из  $A$  логически следует  $B$ , то

формула  $A \supset B$  будет общезначимой. Однако из  $A$  может следовать  $B$ , но при этом формула  $A \supset B$  общезначимой не будет. Например, из  $Ax$  следует  $\forall xAx$ , но формула  $Ax \supset \forall xAx$  не является общезначимой и из  $Ax$  логически не следует  $\forall xAx$ .

Легко видеть, что для предложений отношения следования и логического следования совпадают. Исходя из этого можно говорить, что предложение  $A$  логически следует из множества предложений  $\Gamma$  тогда и только тогда, когда всякая модель множества  $\Gamma$  является моделью  $A$ .

### 1.5. Способы формализации логического следования и общезначимости

Мы можем теперь точно ответить на вопрос, что есть правильное рассуждение в рамках описанного выше первопорядкового языка. Рассуждение правильно тогда и только тогда, когда между посылками и заключением этого рассуждения имеет место отношение логического следования. Теперь встает вопрос, можно ли формализовать это отношение.

Проблема формализации семантического понятия логического следования сводится к вопросу о том, что если  $A$  логически следует из множества формул  $\Gamma$ , то в этом исчислении можно построить вывод  $A$  из посылок  $\Gamma$ , и наоборот, если существует вывод  $A$  из посылок  $\Gamma$ , то из  $\Gamma$  логически следует  $A$ . Для языка первопорядковой логики предикатов этот вопрос решается положительно. В логике разработано несколько способов формализации отношения логического следования. По существу все они сводятся к трем основным: аксиоматическому (гильбертовскому), натуральному выводу и секвенциальному (генценовскому). Следует отметить, что все три типа исчислений могут быть представлены в многочисленных ва-



риантах. Начнем с формулировки аксиоматического исчисления гильбертовского типа.

*Аксиоматическое исчисление предикатов (НС)*, или классическое исчисление гильбертовского типа, включает список формул, принимаемых в качестве аксиом, правила вывода и определение понятия вывода. Логическими аксиомами будут формулы вида:

1.  $A \supset (B \supset A)$ .
2.  $(A \supset (B \supset C)) \supset ((A \supset B)(A \supset C))$ .
3.  $A \& B \supset A$ .
4.  $A \& B \supset B$ .
5.  $A \supset (B \supset (A \& B))$ .
6.  $A \supset A \vee B$ .
7.  $B \supset A \vee B$ .
8.  $(A \supset B) \supset ((C \supset B)(A \vee C \supset B))$ .
9.  $(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$ .
10.  $\neg \neg A \supset A$ .
11.  $\forall x F_x^w A \supset F_t^w A$ .
12.  $\forall x F_x^w (B \supset A) \supset (B \supset \forall x F_x^w A)$ .
13.  $F_t^w A \supset \exists x F_x^w A$ .
14.  $\forall x F_x^w (A \supset B) \supset (\exists x F_x^w A \supset B)$ .

В формулах 12 и 14  $B$  не содержит вхождений переменной  $w$ . Правила вывода:

$$\frac{A, A \supset B}{B} - \text{modus ponens}; \quad \frac{A}{\forall x F_x^w A} - \text{обобщение}.$$

В последних четырех аксиомах и правиле обобщения подстановки везде правильны. Нужно отметить, что выражения 1 - 14, строго говоря, являются не аксиомами, а схемами аксиом, в которых  $A$ ,  $B$ ,  $C$  представляют любые правильно построенные формулы.

Схемы аксиом можно рассматривать как правила вывода из пустого множества посылок<sup>5</sup>.

Будем говорить, что формула, стоящая в правиле вывода под чертой, непосредственно выводима из формул, стоящих над чертой.

Вывод из множества посылок  $\Gamma$  есть последовательность формул, такая, что каждая формула, входящая в эту последовательность, является или аксиомой, или одной из посылок множества  $\Gamma$ , или непосредственно выводима из формул, предшествующих ей в данной последовательности.

Формула  $A$  выводима из множества посылок  $\Gamma$ , если и только если существует вывод из посылок  $\Gamma$  и  $A$  есть последняя формула этого вывода.

Вывод из пустого множества посылок будем называть *доказательством*, а его последнюю формулу — *доказуемой формулой*.

Определение вывода в исчислении  $HC$  эффективно в том смысле, что относительно любой последовательности формул мы можем однозначно установить, является она выводом или нет, проведя анализ вывода. Анализ вывода есть явное указание на то, на каком основании каждая формула входит в данную последовательность: как аксиома определенного вида, в качестве посылки или как формула, непосредственно выводимая из предшествующих формул по определенному правилу вывода. Для удобства вывод в исчислении будем записывать в виде столбца формул: слева пишем номера формул, входящих в вывод, справа — ана-

<sup>5</sup> В случае классической логики вместо схем аксиом 12 и 14 можно применять два правила вывода, которые получили название правил Бернайса:

$$\frac{B \supset Aw}{B \supset \forall x F_x^w Ax}, \quad \frac{Aw \supset B}{\exists x F_x^w Aw \supset B},$$

где  $B$  не содержит вхождений свободной переменной  $w$ .

лиз данного вывода. Например:

1. $\forall x(Px \supset Qx)$	посылка
2. $\forall x(Qx \supset Sx)$	посылка
3. $\forall x(Px \supset Qx) \supset (Pw \supset Qw)$	аксиома 11
4. $Pw \supset Qw$	т.р.; 1, 3
5. $\forall x(Qx \supset Sx) \supset (Qw \supset Sw)$	аксиома 11
6. $Qw \supset Sw$	т.р.; 2, 5
7. $(Qw \supset Sw) \supset (Pw \supset (Qw \supset Sw))$	аксиома 1
8. $Pw \supset (Qw \supset Sw)$	т.р.; 6, 7
9. $(Pw \supset (Qw \supset Sw)) \supset ((Pw \supset Qw) \supset (Pw \supset Sw))$	аксиома 2
10. $(Pw \supset Qw) \supset (Pw \supset Sw)$	т.р.; 8, 9
11. $Pw \supset Sw$	т.р. 4, 10
12. $\forall x(Px \supset Sx)$	обобщ.; 11

Это есть вывод в исчислении **НС** формулы  $\forall x(Px \supset Sx)$  из посылок  $\forall x(Px \supset Qx)$ ,  $\forall x(Qx \supset Sx)$ . В записи анализа вывода строка "т.р.; 2, 5" означает, что формула, стоящая в этой строке (в данном случае в строке 6), получена по правилу *modus ponens* из формул, стоящих в строках 2 и 5.

Теперь поставим следующий существенный вопрос. Является ли исчисление **НС** с определенным выше понятием вывода из посылок искомой адекватной формализацией семантического отношения логического следования? Ответ оказывается однозначно отрицательным, поскольку можно показать, что согласно правилу обобщения мы можем перейти от  $Px$  к  $\forall xPx$ , но из первой формулы логически не следует вторая. Это же относится и к правилам Бернайса (если они включены в формулировку системы <sup>6</sup>).

Для того чтобы **НС** полностью формализовало отношение логического следования, необходимо зафиксировать определенные ограничения на применения пра-

<sup>6</sup> Однако исчисление **НС** с таким понятием вывода из посылок адекватно формализует отношения следования и следования относительно общезначимости.

вила обобщения (и правил Бернайса). В этих целях обычно используются понятие зависимости вхождения некоторой формулы в вывод от посылок и понятие варьирования в данном выводе некоторой свободной переменной.

Вхождение формулы  $B$  в вывод из посылок  $A_1, \dots, A_n$  зависит от посылки  $A_i$  ( $i=1, \dots, n$ ), если

- 1)  $B$  совпадает с  $A_i$ ,
- 2)  $B$  непосредственно выводима из  $C$  и формула  $C$  зависит от  $A_i$  или она выводима из формул  $C$  и  $D$ , которые зависят от  $A_i$ .

Свободная переменная  $w$  варьируется в данном выводе из посылок  $A_1, \dots, A_n$  относительно посылки  $A_i$  ( $i=1, \dots, n$ ), если в  $A_i$  имеются вхождения  $w$ , и в этом выводе применяется правило обобщения (правило Бернайса) по переменной  $w$  к формуле, которая зависит от посылки  $A_i$ .

Если можно построить вывод формулы  $B$  из множества посылок  $\Gamma$ , такой, что в нем ни одна переменная не варьируется относительно какой-нибудь посылки, будем писать  $\Gamma \vdash B$ . Если же в выводе имеет место варьирование переменных  $w_1, \dots, w_k$ , то пишется  $\Gamma \vdash^{w_1 \dots w_k} B$ .

Так, в приведенном выше примере вывода формулы  $\forall x(Px \supset Sx)$  из посылок  $\forall x(Px \supset Qx)$ ,  $\forall x(Qx \supset Sx)$  ни одна переменная не варьируется, и мы можем записать

$$\forall x(Px \supset Qx), \forall x(Qx \supset Sx) \vdash \forall x(Px \supset Sx).$$

Но, например, в выводе формулы  $\forall x(Px \supset Sx)$  из посылок  $\forall x(Qx \supset Sx)$  и  $Pw \supset Qw$  переменная  $w$  будет варьироваться, и в этом случае мы запишем

$$\forall x(Qx \supset Sx), Pw \supset Qw \vdash^w \forall x(Px \supset Sx).$$

В том случае, когда все посылки являются замкнутыми формулами, варьирования переменных не будет и вывод можно строить без всяких ограничений на применение правила обобщения и правил Бернайса.

Логическое исчисление сформулировано. Теперь возникает проблема: является ли данная формализация адекватной? Она сводится к двум вопросам — о семантической непротиворечивости (корректности) и о полноте этого исчисления.

Ответ на первый вопрос дает теорема о семантической непротиворечивости исчисления предикатов. Нетрудно показать, что если в исчислении **НС** формула  $A$  выводима из множества посылок  $\Gamma$  (без варьирования переменных в выводе  $A$ ), то  $A$  логически следует (в семантическом смысле) из  $\Gamma$ . Это означает, что данная формализация корректна, т.е. в рамках исчисления мы не построим выводов, представляющих рассуждения, которые были бы неприемлемы с семантической точки зрения. Доказательство теоремы о непротиворечивости **НС** достаточно простое, и оно излагается в учебниках по формальной логике.

Более трудным является вопрос о полноте логического исчисления. Все способы рассуждений, приемлемые с семантической точки зрения могут быть формализованы в исчислении **НС**? Ответ на этот вопрос был получен К.Геделем в 1930 г. в работе "Полнота аксиом логического функционального исчисления" [31]. В дальнейшем доказательство Геделя было упрощено и усовершенствовано, а также предложены другие способы доказательств полноты первопорядкового исчисления предикатов. Среди известных можно назвать метод Л.Хенкина [35], улучшенный Г.Хазенбегером [34], метод модельных множеств Я.Хинтикки [36,38], алгебраические [49] и топологические [26] методы доказательства. Доказательство полноты **НС** сложнее доказательства теоремы о корректности, и оно излагается в любом более или менее полном курсе символической логики.

Однако неверно думать, что утверждение о полноте **НС** означает, что все правильные рассуждения мо-

гут быть формализованы. Уже в том случае, если мы перейдем от языка логики предикатов первого порядка к более богатым логическим языкам, в частности к второпорядковому языку, в котором применяются кванторы не только по индивидуальным, но и по предикатным или функциональным переменным, проблема адекватной формализации рассматривается по-другому. Можно построить для второпорядкового языка точную семантику и точным образом определить семантические понятия общезначимости и логического следования. Но при этом оказывается, что данные понятия в принципе не могут быть формализованы, т.е. нельзя построить такое обладающее свойством полноты непротиворечивое исчисление, в котором все правила вывода были бы финитны<sup>7</sup>, а число логических аксиом конечным и рекурсивным. Строгое доказательство того, что семантическое понятие логического следования для второпорядкового языка принципиально не может быть формализовано в исчислении, было дано также К.Геделем в 1931 г. в его знаменитой работе "О формальной неразрешимости предложений *Principia Mathematica* и родственных систем" [32].

Часто упомянутый результат К.Геделя формулируют как теорему о неполноте формализованной арифметики. Но при этом надо иметь в виду, что неполнота формализованной арифметики возникает не из-за слабости (неполноты) используемых собственно арифметических аксиом, а является следствием неполноты логических средств, применяемых в исчислении. Пояснить это можно следующим образом. Хорошо известно, что аксиомы арифметики Пеано можно сформулировать на языке второпорядковой логики. Тогда схема математической индукции будет выражима в

---

<sup>7</sup> Логическое правило называется финитным, если число посылок этого правила конечно. Очевидно, что правила исчисления **НС** финитны.

форме одного предложения с кванторами по всем свойствам: "Для всякого свойства  $P$  из того, что  $0$  обладает свойством  $P$ , и из того, что если  $x$  обладает свойством  $P$ , то  $x+1$  тоже обладает свойством  $P$ , следует, что всякий  $x$  обладает свойством  $P$ ". Здесь  $P$  является предикатной переменной и квантификация проводится по предикатным переменным, пробегающим по всем свойствам, а не только по тем, которые выразимы в первопорядковом языке.

Хорошо известно, что сформулированная таким образом система аксиом арифметики Пеано является категорической, т.е. все модели данной второпорядковой системы изоморфны между собой. Но семантическое понятие логического следования в этом случае оказывается "слишком богатым" и неформализуемым ни в каком стандартном языке, в котором все правила вывода финитны и эффективны, и понятие аксиомы тоже является эффективным. Таким образом, неполнота формализации обусловлена не какими-либо недостатками математических постулатов, а глубиной и сложностью семантических понятий логического следования и общезначимости для языков, более богатых, чем первопорядковый.

Возвращаясь к аксиоматическому исчислению  $HC$ , отметим, что интерес представляют не только такие его общие свойства, как неполнота и непротиворечивость. Важно также знать, какими свойствами обладают выводы в этом исчислении, насколько удобны они для моделирования содержательных логических рассуждений, что можно извлечь для анализа содержательных рассуждений в различных областях науки, применяя технику этих выводов, насколько они "естественны", "ясны", "удобны" и т.д. Выявление указанных или каких-то других свойств выводов особенно существенно в том случае, когда дело касается внелогических приложений формального исчисления.

Разумеется, речь идет не только о чисто синтаксической стороне исчисления – о свойствах выводов как синтаксических объектов, не менее важны естественность, ясность, а возможно, какие-то другие свойства семантических построений, необходимые для более эффективного кодирования информации о той области объектов, о которой ведутся рассуждения.

Нетрудно убедиться, что с некоторой "практической" точки зрения" исчисление гильбертовского типа весьма неудобно. Построение вывода в **НС** оказывается процедурой весьма неестественной и сложной, требующей определенных навыков и приложения творческих усилий. Причина заключается в том, что шаги построения вывода в аксиоматическом исчислении очень мелки. Фактически каждый шаг элементарен – это есть применение правила к формуле или паре формул. Используются практически всего два правила, и цель их применений – получить доказуемую формулу (или формулы, из которых она может быть выведена). При этом мы на каждом шаге можем иметь весьма большое множество уже полученных формул вместе с посылками и аксиомами. Но ни в определении понятия вывода, ни в формулировке исчисления не содержится указаний на то, к каким формулам и в каком порядке следует применять правила, строя вывод той или иной формулы. Некоторые "подсказки" можно, конечно, извлечь из структур посылок, аксиом и самой доказываемой формулы. но в общем случае их эффект незначителен. Выбор нужных аксиом на нужном шаге остается на уровне догадок, интуиции, навыка (если исключить, конечно, тривиальный перебор всех вариантов на каждом шаге, делающий построение достаточно сложного вывода процедурой, практически нереализуемой за время, скажем, равное времени человеческой жизни).

Однако ситуация может улучшиться, если мы най-



дем возможность строить вывод не только элементарными шагами, но применяя более крупные "логические блоки". Само по себе исчисление позволяет это сделать. Поясним, что здесь имеется в виду. Всякое исчисление содержит правила вывода, необходимые для его формулировки, они называются *основными правилами вывода*. В исчислении *НС* это правила *modus ponens* и обобщения. Укрупнение шагов (использование блоков) вывода может быть достигнуто за счет применения производных правил. *Производным правилом вывода* (некоторого исчисления) называется такое правило, что можно построить вывод заключения этого правила из его посылки с помощью основных правил (и, возможно, аксиом данного исчисления).

Прежде чем сформулировать наиболее интересные производные правила, применимые в первопорядковом исчислении предикатов, отметим еще одно методологически важное деление правил вывода – на прямые и не прямые. *Прямые правила вывода* суть правила непосредственного перехода от одних формул к другим, т.е. от посылки (посылок) к заключению. Им сопоставляются определенные шаги формального вывода. *Непрямые правила вывода* суть правила перехода от одних формальных выводов к другим. Таким правилам соответствуют метаутверждения (метатеоремы) о преобразованиях одних формальных выводов в другие. Заметим, что основные правила исчисления *НС* являются прямыми – *НС* не содержит никаких не прямых основных правил вывода. Это же имеет место в любых других исчислениях гильбертовского типа, и потому говорят, что такого типа исчисления непосредственно формализуют только прямые способы рассуждений. Но, как мы отмечали, эти исчисления допускают модификацию за счет более крупных шагов вывода в виде применений производных правил.

Начнем с формулировки прямых производных правил. Рассмотрим на примерах, на чем основана идея "возникновения" и применения прямых производных правил. Замечая, что всегда можно построить вывод

1.	$A$	посылка
2.	$B$	посылка
(I)	3. $A \supset (B \supset (A \& B))$	аксиома 5
	4. $B \supset (A \& B)$	т.р.; 1,3
	5. $A \& B$	т.р.; 2,4,

а также вывод

1.	$A \supset B$	посылка
2.	$\neg B$	посылка
3.	$(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$	аксиома 9
(II)	4. $(A \supset \neg B) \supset \neg A$	т.р.; 1,3
	5. $\neg B \supset (A \supset \neg B)$	аксиома 1
	6. $A \supset \neg B$	т.р.; 2,5
	7. $\neg A$	т.р.; 6,4

и вывод

1.	$A \supset B$	посылка
2.	$B \supset C$	посылка
(III)	3. $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$	аксиома 2
	4. $(B \supset C) \supset (A \supset (B \supset C))$	аксиома 1
	5. $A \supset (B \supset C)$	т.р.; 2,4
	6. $(A \supset B) \supset (A \supset C)$	т.р.; 5,3
	7. $A \supset C$	т.р.; 1,6,

мы оформляем результаты этих выводов в виде правил. Выводу (I) соответствует простое правило введения конъюнкции (из посылок  $A$  и  $B$  выводима их конъюнкция):

$$\frac{A, B}{A \& B}.$$

Выводу (II) соответствует известное еще античным логикам правило, получившее название *modus tollens*:

$$\frac{A \supset B, \neg B}{\neg A}.$$

Выводу (III) соответствует также известное и широко применяемое правило транзитивности импликации:

$$\frac{A \supset B, B \supset C}{A \supset C}.$$

Итак, идея обоснования производных правил и законности их применения достаточно проста. Если некоторые выводы уже построены, т.е. мы знаем, что из таких-то посылок с необходимостью следует такое-то заключение, то это знание мы оформляем в виде прямого правила перехода от данных посылок к заключению и в дальнейшем, при построении другого вывода используем их в качестве блоков, значительно сокращая этим построение. Производные прямые правила суть не что иное, как краткая форма записи выводов, которые включаются в качестве крупных шагов, фрагментов в построение более сложных выводов. Конечно, вывод, полученный с помощью производных правил, не будет выводом в строгом смысле слова — это некоторый сокращенный вывод, который мы всегда можем превратить в полный (удовлетворяющий определению вывода в НС), заменяя всякое применение прямого производного правила выводом заключения этого правила из его посылок.

Итак, обоснование производных правил состоит в указании эффективного метода, с помощью которого любой вывод, содержащий применения производных правил, можно преобразовать в вывод с той же конечной формулой, содержащей только применения основных правил. Читатель может убедиться сам в том, что нетрудно обосновать производность в НС следующих прямых правил:

из  $\neg \neg A$  выводимо  $A$ :

$$\frac{\neg \neg A}{A} \quad - \text{закон снятия двойного отрицания;}$$

из  $A$  выводимо  $\neg\neg A$ :

$\frac{A}{\neg\neg A}$  – закон введения двойного отрицания;

правила коммутативности конъюнкции и дизъюнкции

$$\frac{A \& B}{B \& A}, \quad \frac{A \vee B}{B \vee A};$$

правила де Моргана:

$$\frac{\neg(A \& B)}{\neg A \vee \neg B}, \quad \frac{\neg A \& \neg B}{\neg(A \vee B)}, \quad \frac{\neg A \vee \neg B}{\neg(A \& B)}, \quad \frac{\neg(A \vee B)}{\neg A \& \neg B};$$

правила удаления дизъюнкции посредством отрицания:

$$\frac{A \vee B, B}{A}, \quad \frac{\neg A \vee B, A}{B}, \quad \frac{A \vee B, \neg A}{B}, \quad \frac{A \vee B, \neg B}{A}.$$

Способу рассуждения, которым пользовался еще Евклид, соответствуют следующие правила:

$$\frac{A \supset \neg A}{\neg A}, \quad \frac{\neg A \supset A}{A}.$$

Приведем несколько примеров производных правил с кванторами. Это правила преобразования формул, содержащих квантор существования, в формулы, содержащие квантор всеобщности, и наоборот:

$$\frac{\exists x A x}{\neg \forall x \neg A x}, \quad \frac{\forall x A x}{\neg \exists x \neg A x};$$

правила перестановки кванторов:

$$\frac{\forall x \forall y A}{\forall y \forall x A}, \quad \frac{\exists x \exists y A}{\exists y \exists x A}, \quad \frac{\exists x \forall y A}{\forall y \exists x A};$$

правила пренесения квантора  $\forall$  через конъюнкцию и квантора  $\exists$  через дизъюнкцию:

$$\frac{\forall x (A x \& B x)}{\forall x A x \& \forall x B x}, \quad \frac{\exists x (A x \vee B x)}{\exists x A x \vee \exists x B x}.$$

Интересно производное правило, в котором выражена возможность переименования переменных в выводах **НС**:

$$\frac{\forall x F_x^w A}{\forall y F_y^w A}.$$

Многие производные правила целесообразно оформлять в виде эквивалентностей. Возможность введения и удаления знака эквивалентности выражают правила:

$$\frac{A \supset B, B \supset A}{A \equiv B}, \quad \frac{A \equiv B}{A \supset B}, \quad \frac{A \equiv B}{B \supset A}.$$

Легко убедиться, что в **НС** доказуемы, например, следующие эквивалентности:

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

Эквивалентные преобразования значительно сокращают построение вывода в аксиоматических системах, и их применения можно рассматривать как применения прямых производных правил вывода.

Наряду с прямыми производными правилами вывода можно сформулировать не прямые производные правила, которые суть не что иное, как правила преобразования одних выводов в другие. Согласно такому правилу, если можно построить некоторый вывод, который называется *исходным*, то можно построить и некоторый другой вывод, называемый *результатирующим*. Наиболее интересным примером непрямого производного правила вывода является так называемая теорема о дедукции, которая формулируется следующим образом.

Если имеется вывод формулы  $B$  из множества посылок  $A, \Gamma$  и ни одна переменная этого вывода не варьируется относительно формулы  $A$ , то можно построить вывод формулы  $A \supset B$  из множества посылок  $\Gamma$ . Символически:

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A \supset B} .$$

Дедукционная теорема обосновывает не прямое производное правило, получившее название *правила введения импликации*.

Аналогичным образом можно сформулировать другие не прямые производные правила вывода. К их числу относится широко известный *метод рассуждения путем разбора случаев*: для того чтобы построить вывод формулы  $C$  из посылок  $A \vee B, \Gamma$ , нам достаточно получить выводы этой формулы из множества посылок  $A, \Gamma$  и множества посылок  $B, \Gamma$ , т.е.

$$\frac{A, \Gamma \vdash C; B, \Gamma \vdash C}{A \vee B, \Gamma \vdash C} .$$

Еще одним интересным способом рассуждения, который может быть оформлен в виде не прямого производного правила, является так называемый *метод доказательства от противного*. Пусть нам нужно получить вывод формулы  $A$  из посылок  $\Gamma$ . Тогда данную задачу мы сводим к следующей: отрицание формулы  $A$  добавляем в качестве посылки к множеству  $\Gamma$  и пытаемся получить из посылок  $\neg A, \Gamma$  противоречие. Если таковое получено, например мы выводим и  $B$  и  $\neg B$ , то это означает, что можно построить вывод  $A$  из  $\Gamma$ . Обозначив противоречие константой  $f$ , можно записать соответствующее этому типу рассуждений правило:

$$\frac{\neg A, \Gamma \vdash f}{\Gamma \vdash A} .$$

Необходимо заметить, что способ рассуждения от противного, который выражен в этом правиле, является сугубо классическим. Существует аналогичный, но более слабый способ, который можно выразить не прямым правилом

$$\frac{A, \Gamma \vdash J}{\Gamma \vdash \neg A},$$

которое приемлемо как в классической, так и в интуиционистской логике.

Подобные способы рассуждений были известны, применялись и анализировались еще в античности. В исчислении предикатов они находят свои точные формулировки. Выделим два вида производных правил: *логические правила* и *структурные правила*. Они могут быть как прямыми, так и непрямыми. Примерами прямых производных логических правил являются приведенные выше правила введения знака &, введения и снятия двойного отрицания, а также многие другие правила, связанные с введением или удалением логических связок или эквивалентными преобразованиями формул, содержащих различные связки. Структурными называются правила, затрагивающие не логические связки, а структуру выводов в целом. Простейшим примером структурного правила является правило перестановки посылок:

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}.$$

Легко обосновать также производные структурные правила сокращения и добавления посылок:

$$\frac{A, A, \Gamma \vdash B}{A, \Gamma \vdash B}, \quad \frac{\Gamma \vdash B}{A, \Gamma \vdash B}.$$

Весьма важную роль играет структурное правило, называемое сечением:

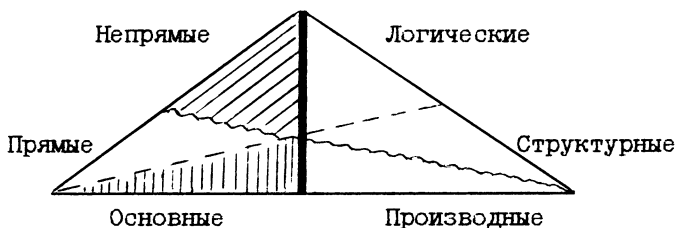
$$\frac{\Gamma \vdash A; \quad A, \Delta \vdash B}{\Gamma, \Delta \vdash B}.$$

Более подробно речь о нем пойдет в дальнейшем.

Деление на логические и структурные относится не только к производным правилам. Например, в сек-

венциальной формулировке исчисления предикатов, которую мы приведем ниже, как структурные, так и логические правила являются основными.

Итак, можно разделить правила вывода, во-первых, на основные (для данной системы) и производные (в данной системе), во-вторых, на прямые и не прямые и, наконец, на логические и структурные. Сведем данную классификацию в следующую условную схему (для исчисления **НС**):



где жирная черта делит все множество правил на основные и производные, волнистая – на прямые и не прямые, а штриховая – на логические и структурные. Заштрихованные части схемы обозначают, что среди основных правил **НС** нет структурных и не прямых логических правил, а есть только прямые логические правила.

В исчислениях гильбертовского типа может применяться бесконечное число всевозможных производных правил, в зависимости от того, сколь крупными блоками мы оперируем при построении вывода. Но в исчислениях такого типа невозможно сопоставить некоторые формальные объекты таким способам рассуждения, которые отражены, например, в дедукции или непрямом правиле рассуждения по случаям. Поэтому говорят, что исчисления гильбертовского типа не дают прямой формализации такого рода рассуждений, широко используемых в практике. Однако эта ситуация может быть изменена. Может быть дана формализация другого типа, включающая как прямые, так и



непрямые способы рассуждений. Это так называемые системы натурального или естественного вывода.

*Классическая система натурального вывода (NC)* является достаточно удобной и эффективной формализацией отношения логического следования, приближенной к обычным рассуждениям. Задача построения такой формализации впервые была поставлена известным польским логиком Я. Лукасевичем в 1927 г. Первые системы натурального вывода были построены А.Яськовским [39] под непосредственным влиянием идей Я.Лукасевича и независимо от него Г.Генценом в его известной работе "Исследования логических выводов" [6]. Позже были предложены различные модификации этих построений.

В основе формализации гильбертовского типа лежит понятие вывода как последовательности формул, которую можно построить, переходя от одних формул к другим согласно правилам вывода. Для того чтобы учесть не прямые рассуждения, необходимо допустить, чтобы некоторые формальные объекты более сложного вида, нежели формулы, а именно – вспомогательные выводы, рассматривались наравне с формулами как составные части формализованного рассуждения. Поэтому при построении систем натурального вывода используется понятие вывода более сложное, чем в исчислениях гильбертовского типа. Последовательность натурального вывода строится из элементов, которыми могут быть не только формулы, но и некоторые, возможно, очень сложные вспомогательные выводы, причем последние, в свою очередь, могут состоять и из формул, и из других вспомогательных выводов. Чтобы зафиксировать структуру натурального вывода, можно использовать различные средства. С этой целью, например, в первых системах Яськовского вспомогательные выводы заключались в прямоугольную рамку. Приведем простой пример.

Допустим, мы хотим построить вывод формулы  $A \supset C$  из посылок  $A \supset B$  и  $B \supset C$ . Запишем его в следующем виде:

1.	$A \supset B$	посылка
2.	$B \supset C$	посылка
3.		посылка
4.	$A \supset B$	перенос пос.1
5.	$B$	т.р.; 3,4
6.	$B \supset C$	перенос пос.2
7.	$C$	т.р.; 5, 6
8.	$A \supset C$	

В данном случае мы имеем не обычную (в смысле гильбертовских систем), а субординатную последовательность, построенную следующим образом. На первых двух шагах построения пишем посылки 1 и 2. Затем вводим дополнительную посылку  $A$  и строим вспомогательный вывод. Перенесем из основного вывода посылку 1 и по правилу *modus ponens* получаем  $B$ , так же переносим посылку 2, получая по *modus ponens* формулу  $C$ . И, наконец, завершая все построение, пишем формулу  $A \supset C$  в качестве последней формулы основного вывода. В этом случае натуральный вывод состоит из четырех элементов: трех формул и одного вспомогательного вывода, заключенного в рамку. Отметим, что формулы, входящие во вспомогательный вывод, не входят в основной вывод, но при этом, естественно, входят в конструкцию натурального вывода как целого. Вхождения вспомогательных выводов можно отмечать также уголками, звездочками и другими значками, фиксирующими начало и конец вывода. Ниже для указания вспомогательных выводов вместо прямоугольника мы будем использовать вертикальную черту слева. Заметим также, что в приведенном примере построения натурального вывода проводился перенос посылок основного вывода во вспомогательный – это так называемое правило переноса,

или реитерации. В целях упрощения можно не применять такое правило, просто разрешая без переноса использовать посылки основного вывода.

Очевидно, что в натуральных системах вывод есть более сложный формальный объект, нежели вывод в аксиоматических. И принципиальное отличие этих двух типов формализаций рассуждений состоит не в том, имеются или нет аксиомы, а в самом понятии вывода. Опишем систему натурального вывода **NC** более строго. Она включает прямые и не прямые правила вывода, которые удобно разделить на правила введения и правила удаления логических знаков.

Прямые правила вывода:

правила введения:

$$\&_B \frac{A, B}{A \& B}, \quad \vee_B \frac{A}{A \vee B}, \quad \vee_B \frac{A}{A \vee B}, \quad \forall_B \frac{A_w}{\forall x F_x^w A},$$

$$\exists_B \frac{F_t^w A}{\exists x F_x^w A};$$

правила удаления:

$$\supset_y \frac{A \supset B, A}{B}, \quad \&_y \frac{A \& B}{A}, \quad \&_y \frac{A \& B}{B}, \quad \neg \neg_y \frac{\neg \neg A}{A},$$

$$\forall_y \frac{\forall x F_x^w A}{F_t^w A}.$$

При этом переменная  $w$  все время варьируется, если правило  $\forall_B$  применяется к формуле  $A$ , содержащей  $w$  и зависящей от посылки, в которую входит переменная  $w$ .

К непрямым правилам вывода относятся правила введения импликации  $\supset_B$ , введения отрицания  $\neg_B$ , удаления дизъюнкции  $\vee_y$  и удаления квантора существования  $\exists_y$ . Поскольку в формулировках не прямых пра-

вил используется понятие вывода, естественно включить эти формулировки непосредственно в определение вывода.

Вывод в системе **НС** есть субординантная последовательность формул (каждый член которой является или формулой, или вспомогательным выводом), построенная в соответствии со следующими пунктами:

1. Формула  $A$  есть вывод из множества посылок  $\{A\}$ .  
 2. Если  $\alpha$  есть вывод из множества посылок  $\Gamma$  и  $A$  — формула, то  $\frac{A}{\alpha}$  есть вывод из множества посылок  $A, \Gamma$ .

3. Если  $\alpha$  есть вывод из множества посылок  $\Gamma$ , формула  $A$  входит в  $\alpha$  и  $B$  получена из  $A$  по одному из прямых однопосылочных правил вывода, кроме  $\forall_B$ , тогда  $\frac{\alpha}{B}$  есть вывод из множества посылок  $\Gamma$ .

4. Если  $\alpha$  есть вывод из множества посылок  $\Gamma$ , формулы  $A$  и  $B$  входят в  $\alpha$  и формула  $C$  получена из формул  $A$  и  $B$  по одному из прямых двухпосылочных правил вывода, то  $\frac{\alpha}{C}$  есть вывод из множества посылок  $\Gamma$ .

5. Если  $\alpha$  есть вывод из множества посылок  $\Gamma$ , формула  $Aw$  входит в  $\alpha$  и ни одна формула из множества  $\Gamma$  не содержит вхождений переменной  $w$ , то  $\frac{\alpha}{\forall x F_x^w Aw}$  есть вывод из множества посылок  $\Gamma$ .

6. Если  $\alpha$  есть вывод из множества посылок  $\Gamma$  и  $C_1, \dots, C_n$  есть вывод из посылок  $A, C_1, \dots, C_n$ , а формулы  $\beta$

$C_1, \dots, C_n$  входят в  $\alpha$  и  $B$  есть последняя формула  $\beta$  то  $\frac{\alpha}{\beta \begin{smallmatrix} A \supset B \end{smallmatrix}}$  есть вывод из множества посылок  $\Gamma$ .

7. Если  $\alpha$  есть вывод из множества посылок  $\Gamma$ ,  
 $\begin{matrix} D_1 \\ \vdots \\ D_n \end{matrix}$  есть вывод формулы  $C$  из посылок  $A, D_1, \dots, D_n$ , а  $\begin{matrix} E_1 \\ \vdots \\ E_m \end{matrix}$   
 $\beta_1$   $\beta_2$   
 есть вывод формулы  $C$  из посылок  $B, E_1, \dots, E_m$  и  
 формулы  $D_1, \dots, D_n, E_1, \dots, E_m$  входят в  $\alpha$ , то  
 $\begin{matrix} \alpha \\ \wedge B \\ \mid \beta_1 \\ \mid \beta_2 \\ C \end{matrix}$  есть вывод из множества посылок  $\wedge B, \Gamma$ .

8. Если  $\alpha$  есть вывод из множества посылок  $\Gamma$ ,  
 $\begin{matrix} C_1 \\ \vdots \\ C_n \end{matrix}$  есть вывод формулы  $B \& \neg B$  из посылок  $A, C_1, \dots, C_n$   
 $B \& \neg B$   
 и формулы  $C_1, \dots, C_n$  входят в  $\alpha$ , то  $\begin{matrix} \alpha \\ \mid \beta \\ \neg A \end{matrix}$  есть вывод  
 из множества посылок  $\Gamma$ .

9. Если  $\alpha$  есть вывод из множества посылок  $\Gamma$ ,  
 $\begin{matrix} C_1 \\ \vdots \\ C_n \end{matrix}$  есть вывод формулы  $B$  из посылок  $Aw, C_1, \dots, C_n$  и  
 $\beta$   
 формулы  $C_1, \dots, C_n$  входят в  $\alpha$ , а переменная  $w$  не  
 входит в формулу  $B$ , то  $\begin{matrix} \alpha \\ \exists x A x \\ \mid \beta \\ B \end{matrix}$  есть вывод из множе-  
 ства посылок  $\Gamma$ .

Пункты 6–9 определения понятия вывода в **НС** являются одновременно и формулировками непрямых правил вывода, в которых допускается использовать введенные ранее посылки при построении вспомогательных выводов.

Правило удаления квантора существования может быть сформулировано как прямое правило. В этом случае получается система, более сильная по своим

выразительным средствам, чем сформулированная в данном параграфе система **NC**. О системе натурального вывода с прямым правилом удаления квантора существования речь пойдет ниже.

Нетрудно убедиться, что работать с формальными выражениями в системе натурального вывода намного проще, чем в исчислениях гильбертовского типа. Набор правил вывода в **NC** гораздо богаче, чем в аксиоматических системах, и формулировки этих правил и определение вывода позволяют выявить некоторые общие принципы, "стратегии" построения вывода формулы того или иного вида. В последующем мы покажем, как на базе **NC** можно реализовать некоторую систематическую процедуру построения вывода, исходя из структуры формулы, относительно которой требуется установить, является ли она доказуемой или нет.

Заметим, что в натуральном выводе применяются две процедуры: непосредственное выведение одних формул из других и построение вспомогательных выводов. Если рассматривать построение искомого вывода как некоторую задачу, то натуральная система позволяет действовать двояким образом: строя вспомогательные выводы, сводить исходную задачу к некоторым подзадачам, решение которых является более простым, или искать прямое решение, применяя прямые правила. Например, для того чтобы построить вывод формулы  $A \supset B$  из посылок  $\Gamma$ , мы должны решить вспомогательную задачу (подзадачу) – построить вывод формулы  $B$  из множества посылок  $A, \Gamma$ . Как правило, эта подзадача является более простой, чем исходная задача. Понятие вывода в аксиоматических системах не позволяет применять такое упрощение, что вынуждает решать задачу построения вывода напрямую, ограничиваясь применением лишь двух прямых правил вывода.

В связи с этим естественно поставить вопрос о том, нельзя ли сформулировать логическое исчисление таким образом, чтобы избежать "неудобств", связанных с применением прямых правил вывода, и иметь возможность строить искомый вывод, сводя на каждом шаге некоторую задачу (выведения формулы) к более простым подзадачам и, в конечном счете, — к некоторым элементарным тривиально решаемым подзадачам. Такая процедура действительно осуществима. Она реализуется в секвенциальных исчислениях.

Секвенциальные исчисления были предложены впервые Г.Генценом в 1934 г. в упомянутой выше работе "Исследования логических выводов" [6]. Одним из основных понятий в формулировке такого рода исчислений является понятие секвенции.

Секвенция есть выражение вида  $\Gamma \rightarrow \Theta$ , где  $\Gamma$  и  $\Theta$  суть конечные последовательности формул, возможно, пустые.

Секвенцию можно понимать как символическую запись выводимости одних формул из других — формул, стоящих справа от стрелки, из формул, стоящих слева от нее. Смысл секвенции раскрывается с помощью понятия представляющей формулы. Пусть  $\Gamma$  есть последовательность формул  $A_1, \dots, A_n$ , а  $\Theta$  — последовательность формул  $B_1, \dots, B_m$ , тогда представляющей формулой секвенции  $\Gamma \rightarrow \Theta$  будет формула  $A_1 \& \dots \& A_n \supset B_1 \vee \dots \vee B_m$  (или формула  $\neg A_1 \vee \dots \vee \neg A_n \vee B_1 \vee \dots \vee B_m$ ). Ясно, что представляющими формулами секвенций  $\rightarrow \Theta$ ,  $\Gamma \rightarrow$  и  $\rightarrow$  будут соответственно формулы  $B_1 \vee \dots \vee B_m$ ,  $\neg A_1 \vee \dots \vee \neg A_n$  и  $f$ , т.е. противоречие.

Дадим строгую формулировку классического секвенциального исчисления (GC). Основной секвенцией (которую иногда называют аксиомой) исчисления GC является секвенция  $A \rightarrow A$ . Для каждого логического знака имеются правила введения этого знака в пра-

бую часть (сукцедент) секвенции и в ее левую часть (антецедент). Это логические правила вывода:

$$\begin{aligned}
 &\rightarrow \supset \frac{A, \Gamma \rightarrow \theta, B}{\Gamma \rightarrow \theta, A \supset B}, \quad \rightarrow \supset \frac{\Gamma \rightarrow \theta, A; B, \Gamma \rightarrow \theta}{A \supset B, \Gamma \rightarrow \theta}, \\
 &\rightarrow \& \frac{\Gamma \rightarrow \theta, A; \Gamma \rightarrow \theta, B}{\Gamma \rightarrow \theta, A \& B}, \quad \& \rightarrow \frac{A, \Gamma \rightarrow \theta}{A \& B, \Gamma \rightarrow \theta}, \quad \& \rightarrow \frac{B, \Gamma \rightarrow \theta}{A \& B, \Gamma \rightarrow \theta}, \\
 &\rightarrow \dot{\vee} \frac{\Gamma \rightarrow \theta, A}{\Gamma \rightarrow \theta, A \vee B}, \quad \rightarrow \vee \frac{\Gamma \rightarrow \theta, B}{\Gamma \rightarrow \theta, A \vee B}, \quad \vee \rightarrow \frac{A, \Gamma \rightarrow \theta; B, \Gamma \rightarrow \theta}{A \vee B, \Gamma \rightarrow \theta}, \\
 &\rightarrow \neg \frac{A, \Gamma \rightarrow \theta}{\Gamma \rightarrow \theta, \neg A}, \quad \neg \rightarrow \frac{\Gamma \rightarrow \theta, A}{\neg A, \Gamma \rightarrow \theta}, \\
 &\rightarrow \forall \frac{\Gamma \rightarrow \theta, A w}{\Gamma \rightarrow \theta, \forall x A x}, \quad \forall \rightarrow \frac{A t, \Gamma \rightarrow \theta}{\forall x A x, \Gamma \rightarrow \theta}, \\
 &\rightarrow \exists \frac{\Gamma \rightarrow \theta, A t}{\Gamma \rightarrow \theta, \exists x A x}, \quad \exists \rightarrow \frac{A w, \Gamma \rightarrow \theta}{\exists x A x, \Gamma \rightarrow \theta}.
 \end{aligned}$$

На правила вывода  $\rightarrow \forall$  и  $\exists \rightarrow$  накладывается ограничение: переменная  $w$  не входит в формулы заключения соответствующего правила.

Структурные правила:

$$\begin{aligned}
 \text{ПЛ: } &\frac{\Delta, A, B, \Gamma \rightarrow \theta}{\Delta, B, A, \Gamma \rightarrow \theta}, \quad \text{ПП: } \frac{\Gamma \rightarrow \theta, A, B, \Delta}{\Gamma \rightarrow \theta, B, A, \Delta}, \\
 \text{СЛ: } &\frac{A, A, \Gamma \rightarrow \theta}{A, \Gamma \rightarrow \theta}, \quad \text{СП: } \frac{\Gamma \rightarrow \theta, A, A}{\Gamma \rightarrow \theta, A}, \\
 \text{ДЛ: } &\frac{\Gamma \rightarrow \theta}{A, \Gamma \rightarrow \theta}, \quad \text{ДП: } \frac{\Gamma \rightarrow \theta}{\Gamma \rightarrow \theta, A}, \\
 \text{сечение: } &\frac{\Gamma \rightarrow \theta, A; A, \Delta \rightarrow Z}{\Gamma, \Delta \rightarrow \theta, Z}.
 \end{aligned}$$



Здесь ПЛ, ПП – правила перестановки, СЛ, СП – сокращения, ДЛ, ДП – добавления левой и правой частей соответственно.

В правилах вывода посылки, т.е. секвенции, написанные над чертой, называются верхними секвенциями, а заключение, т.е. секвенция, написанная ниже черты, – нижней секвенцией. В каждом двухпосылочном правиле имеются две верхние секвенции – левая и правая. Явно выделенные из списков формулы верхних секвенций называются боковыми формулами данного правила, а явно выделенная формула нижней секвенции – главной формулой этого правила. Все остальные формулы, сохраняющиеся без изменений при переходе от верхних секвенций к нижней, называются параметрическими формулами. Это все формулы списков  $\Gamma$ ,  $\Delta$ ,  $\Theta$  и  $Z$ , если списки не являются пустыми. В правилах перестановки ПЛ и ПП боковые и главные формулы совпадают графически, но занимают различные места в сукцеденте или антецеденте. В правилах сокращения СЛ и СП одна из боковых формул просто вычеркивается. Правила добавления ДЛ и ДП (иногда их называют правилами уточнения или ослабления) не имеют боковой формулы, а в правиле сечения отсутствует главная формула.

Можно дать другую формулировку исчисления СС. Основными секвенциями будут секвенции вида  $\Gamma_1, A, \Gamma_2 \rightarrow \Theta_1, A, \Theta_2$ , и вместо правил  $\&\rightarrow$ ,  $\rightarrow\vee$ ,  $\rightarrow\exists$ ,  $\vee\rightarrow$  применяются следующие правила<sup>8</sup>:

$$\frac{At, \forall xAx, \Gamma \rightarrow \Theta}{\forall xAx, \Gamma \rightarrow \Theta}, \quad \frac{\Gamma \rightarrow \Theta, \exists xAx, At}{\Gamma \rightarrow \Theta, \exists xAx},$$

<sup>8</sup> Легко показать, что эти правила являются производными в сформулированном варианте исчисления СС. Основная же секвенция вида  $\Gamma_1, A, \Gamma_2 \rightarrow \Theta_1, A, \Theta_2$  выводима в СС с помощью правил добавления.

$$\frac{\Gamma \rightarrow \theta, A, B}{\Gamma \rightarrow \theta, A \vee B}, \quad \frac{A, B, \Gamma \rightarrow \theta}{A \& B, \Gamma \rightarrow \theta}.$$

В этом случае правила добавления и сокращения не понадобятся – они будут производными такого исчисления. Можно обойтись также без правил перестановки, если отождествлять все списки  $\Gamma_1, \Gamma_2, \dots$ , отличающиеся только перестановками входящих в них формул.

Сформулированное в такой новой форме исчисление секвенций обладает рядом интересных свойств. Во-первых, легко установить, что все правила являются обратимыми, т.е. правила вывода, полученные из исходных заменой верхней секвенции (верхних секвенций) на нижнюю, а нижней – на верхнюю, являются производными правилами. Во-вторых, все правила вывода, исключая сечение, обладают свойством подформульности, т.е. в верхние секвенции входят только формулы, графически совпадающие с формулами нижней секвенции, или являющиеся подформулами нижней секвенции, или являющиеся подформулами формул, входящих в нижнюю секвенцию.

Единственным правилом, играющим в исчислении особую роль, является сечение. Оно нарушает указанные, весьма выгодные во многих отношениях свойства **СС**. Однако Г.Генценом в работе [6] было показано, что и в классической, и в интуиционистской логике первого порядка сечение является допустимым правилом: если можно построить вывод некоторой секвенции с помощью сечения, то можно построить вывод этой же секвенции, не содержащий применений сечения. Теорема Г.Генцена об устранимости (допустимости) сечения является фундаментальным результатом в логике и имеет исключительно важное значение в первую очередь для обоснования логики. В частности, из этой теоремы непосредственно

следует теорема о непротиворечивости исчисления. Действительно, легко установить, что секвенция  $\rightarrow A \& \neg A$  выводима тогда и только тогда, когда выводима пустая секвенция  $\rightarrow$ , но вывод последней невозможно получить без применения сечения, откуда автоматически следует, что противоречивая формула не выводима.

Имеется целый ряд очень интересных применений секвенциальных исчислений. Выводы в таких исчислениях суть конечные деревья секвенций, вершинами которых являются основные секвенции, а корнями — конечные или доказуемые секвенции. В силу свойств обратимости и подформульности правил, выводы в таких исчислениях (с устранимым правилом сечения<sup>9</sup>) можно строить аналитически — снизу вверх, переходя от сложных формул (рассматривая их как главные формулы некоторого правила) к составляющим их подформулам (боковым формулам).

Исчисления секвенций, хотя они первоначально и не были предназначены для этого, составили основу нового направления логического исследования — поиска логических доказательств. Проиллюстрируем это на простом примере. Допустим, поставлена задача найти вывод формулы  $\neg(A \& B) \supset \neg A \vee \neg B$ . Нам нужно установить, доказуема ли секвенция  $\rightarrow \neg(A \& B) \supset \neg A \vee \neg B$ . Согласно правилу  $\rightarrow \supset$ , сводим эту задачу к более простой:

$$\frac{\neg(A \& B) \rightarrow \neg A \vee \neg B}{\rightarrow \neg(A \& B) \supset \neg A \vee \neg B} \rightarrow \supset. \quad (1)$$

<sup>9</sup> Неверно считать, что любая формализация в секвенциальной форме автоматически является непротиворечивой и сохраняет свойство подформульности выводов. Это имеет место лишь в том случае, если сечение устранимо. С другой стороны, непротиворечивое и полное исчисление может быть сформулировано так, что сечение не является в нем допустимым правилом вывода.

Верхнюю секвенцию можно было бы получить двумя способами: применив правило  $\rightarrow \vee_1$  или правило  $\neg \rightarrow$ . Выбирая первый, ставим более простую задачу - получить вывод секвенции  $\neg(A \& B) \rightarrow \neg A, \neg B$ , т.е. имеем

$$\frac{\neg(A \& B) \rightarrow \neg A, \neg B}{\neg(A \& B) \rightarrow \neg A \vee \neg B} \rightarrow \vee_1. \quad (2)$$

Верхнюю секвенцию мы получим по правилу  $\neg \rightarrow$ , если сможем построить вывод более простой секвенции  $\rightarrow A \& B, \neg A, \neg B$ , т.е.

$$\frac{\rightarrow A \& B, \neg A, \neg B}{\neg(A \& B) \rightarrow \neg A, \neg B} \neg \rightarrow. \quad (3)$$

Но у нас есть правило  $\rightarrow \&$ . Значит, на этом шаге задача получения верхней секвенции сводится к тому, чтобы доказать секвенции  $\rightarrow A, \neg A, \neg B$  и  $\rightarrow B, \neg A, \neg B$ :

$$\frac{\rightarrow A, \neg A, \neg B \rightarrow B, \neg A, \neg B}{\rightarrow A \& B, \neg A, \neg B} \rightarrow \&. \quad (4)$$

Теперь ясно, что наша задача сведена к двум очень простым задачам - вывести секвенции  $A \rightarrow A, \neg B$  и  $B \rightarrow B, \neg A$ . Очевидно, что сделать это элементарно - применить правило ДП к основным секвенциям  $A \rightarrow A$  и  $B \rightarrow B$ . В результате мы приходим к тривиальным задачам, т.е. устанавливаем, что искомая секвенция выводима. Объединяя шаги проведенного анализа (снизу вверх, начиная с первого), мы получаем дерево вывода<sup>10</sup>:

<sup>10</sup> При построении вывода в целях сокращения записи используется допущение, что правила перестановки могут не входить в число основных правил исчисления, т.е. порядок вхождения формул в антецедент и сукцедент секвенций не принимается во внимание при применениях правил. Желающие могут легко получить этот вывод, применяя правило перестановки ПП в явном виде.

$$\begin{array}{c}
\frac{A \rightarrow A}{A \rightarrow A, \neg B} \text{ ДП} \quad \frac{B \rightarrow B}{B \rightarrow B, \neg A} \text{ ДП} \\
\frac{\quad}{\rightarrow A, \neg A, \neg B} \rightarrow \neg \quad \frac{\quad}{\rightarrow A, \neg B, \neg A} \rightarrow \neg \\
\frac{\quad}{\rightarrow A \& B, \neg A, \neg B} \rightarrow \& \\
\frac{\quad}{\neg (A \& B) \rightarrow \neg A, \neg B} \neg \rightarrow \\
\frac{\quad}{\neg (A \& B) \rightarrow \neg A \vee \neg B} \rightarrow \vee_1 \\
\frac{\quad}{\rightarrow \neg (A \& B) \supset \neg A \vee \neg B} \rightarrow \supset
\end{array}$$

Поскольку представляющей формулой доказанной секвенции является формула, стоящая в ее сукцедате, первоначальная задача решена - мы установили, что вывод формулы  $\neg (A \& B) \supset \neg A \vee \neg B$  можно построить.

С помощью исчисления секвенций, как будет показано в третьей главе, можно не только решить частный случай подобной задачи, но можно доказать систематический поиск доказательств в рамках языка первопорядкового исчисления предикатов. А в частных случаях, например, исчисления высказываний, исчисления одноместных предикатов и др. эти исчисления дают простое доказательство того, что проблема разрешения разрешима, т.е. для любой правильно построенной формулы мы можем однозначно установить, доказуема она в данной логической системе или нет. Но в данном случае нам более важно то, что идея секвенциальных исчислений открывает широкие возможности для организации систематического поиска логических доказательств.

#### 1.6. Силлогистика и так называемые естественные рассуждения

Хорошо известно, что впервые формализация рассуждений была осуществлена Аристотелем. Им была разработана теория силлогистических выводов. Существует достаточно широко распространенное мнение,

что в силлогистике формализованы естественные рассуждения, которые человек проводит на обычном, естественном языке. В настоящее время в силу различных причин и, в частности, в связи с некоторыми задачами в области компьютерных наук значительно возрос интерес именно к таким естественным рассуждениям. При этом довольно часто традиционная, аристотелевская логика противопоставляется современным методам построения логических систем. В частности, высказывается мнение, что исчисление предикатов является слишком неестественным способом логического анализа содержательных рассуждений, оно "навязывает" слишком жесткие формальные структуры символического языка и в силу этого лишь очень косвенно отражает и даже искажает ход естественных рассуждений, применяемых в практических и в научных целях. Такое противопоставление традиционной и современной символической логики не имеет серьезных оснований. Можно показать, что силлогистика является фрагментом первопорядкового исчисления предикатов, хотя долгое время вопрос о соотношении этих двух способов формализации логического следования действительно оставался открытым.

Кратко опишем аристотелевскую силлогистику. Напомним, что эта теория формулировалась для описания рассуждений, в которых посылки и заключения являются категорическими утверждениями четырех видов (см. разд. 1.2). Обозначим общеутвердительные высказывания — *ASP*, общеотрицательные — *ESP*, частноутвердительные — *ISP* и частноотрицательные — *OSP*. В целях удобства построим силлогистику на базе классической пропозициональной логики. Заметим, что это никак не нарушает общности рассмотрения вопроса, поскольку силлогистика может быть построена совершенно независимо от логики высказываний.

*Система силлогистики* *S2* включает аксиомы клас-

сического исчисления высказываний и следующие специальные аксиомы:

1.  $ASM \ \& \ AMP \supset \ ASP.$
2.  $ASM \ \& \ EMP \supset \ ESP.$
3.  $ASP \supset \ ISP.$
4.  $ESP \supset \ EPS.$
5.  $ISP \equiv \neg EPS.$
6.  $OSP \equiv \neg ASP.$
7.  $ISP \supset \ ASS.$

Единственным правилом вывода является *modus ponens*.

Можно показать, что все принимаемые Аристотелем правила вывода (модусы силлогизма) проходят в системе C2.

Связь между C2 и исчислением предикатов устанавливается на основе следующей интерпретации категорических высказываний<sup>11</sup>:

- $ASP$  интерпретируется как  $\exists xSx \ \& \ \forall x(Sx \supset Px),$   
 $ESP$  интерпретируется как  $\forall x(Sx \supset \neg Px),$   
 $ISP$  интерпретируется как  $\exists x(Sx \ \& \ Px),$   
 $OSP$  интерпретируется как  $\exists xSx \supset \exists x(Sx \ \& \ \neg Px).$

Легко показать, что при данной интерпретации аксиомы 1-7 превращаются в общезначимые формулы одноместного исчисления предикатов, т.е. исчисления предикатов с единственной индивидуальной переменной. Можно также доказать, что если силлогистическая формула недоказуема в C2, то перевод этой формулы будет недоказуем в одноместном исчислении предикатов<sup>12</sup>. Более того, сформулированную систему силлогистики можно обогатить за счет введения сложных терминов, т.е. таких, которые являются объединени-

<sup>11</sup> Система C2 и ее интерпретация в терминах исчисления предикатов предложена одним из авторов книги В.А.Смирновым [21].

<sup>12</sup> Доказательство см. в работе [21, с.156-161].

ем, пересечением или отрицанием терминов, используемых в С2, и доказать, что такая расширенная силлогистика эквивалентна широко известной формальной теории - булевой алгебре классов.<sup>13</sup>

Расширим алгоритм силлогистики следующим образом:

- 1)  $S, P, M, \dots$  суть термы.
- 2) Если  $T$  есть терм, то  $T'$  тоже есть терм.
- 3) Если  $T_1$  и  $T_2$  суть термы, то  $(T_1 \cup T_2)$  и  $(T_1 \cap T_2)$  суть термы.

Из термов обычным образом строятся четыре вида силлогистических атомарных формул:  $AT_1T_2$ ,  $ET_1T_2$ ,  $IT_1T_2$  и  $OT_1T_2$ .

Система расширенной силлогистики С2 содержит аксиомы 1-7 всей системы С2, дополнительные аксиомы

8.  $ASP \supset ESP'$ ,
9.  $ASS \& ESP' \supset ASP$ ,
10.  $E(S \cap P)M \supset E(M \cap S)P$ ,
11.  $EM(S \cup P) \equiv EMS \& EMP$ ,
12.  $EM(S \cap P)' \equiv EMS' \& EMP'$

и единственное правило вывода *modus ponens*.

Булева алгебра классов в терминах операций включения  $\subseteq$ , объединения  $\cup$ , пересечения  $\cap$  и дополнения задается аксиомами

- B1.  $S \subseteq S$ ,
- B2.  $S \subseteq M \& M \subseteq P \supset S \subseteq P$ ,
- B3.  $M \subseteq S \& M \subseteq P \equiv M \subseteq S \cup P$ ,
- B4.  $S \subseteq M \& P \subseteq M \equiv S \cup P \subseteq M$ ,
- B5.  $S \cap (P \cup M) \subseteq (S \cap P) \cup (S \cap M)$ ,
- B6.  $P'' \subseteq P$ ,
- B7.  $S \subseteq P' \supset P \subseteq S'$ ,
- B8.  $P \cap P' \subseteq S$ ,
- B9.  $(S \cap P)' \subseteq S' \cup P'$ .

<sup>13</sup> Впервые эта задача была решена В.А.Бочаровым [4].



Эквивалентность этой системы и системы расширенной силлогистики С2 можно доказать, используя определение отношения включения в терминах силлогистики:  $S \subseteq P =_{Df} ESP'$  и определений силлогистических утверждений в терминах отношения включения:

$$ASP =_{Df} \neg (S \subseteq S') \& (S \subseteq P),$$

$$ESP =_{Df} S \subseteq P',$$

$$ISP =_{Df} \neg (S \subseteq P'),$$

$$OSP =_{Df} \neg (S \subseteq S') \supset \neg (S \subseteq P).$$

Таким образом, различие языков и способов формализации не является принципиальным в логическом исследовании содержательных рассуждений. Что касается собственно силлогистики, то ее аппарат и методы не утратили своего значения и по сей день. Они находят различные приложения в анализе содержательных рассуждений и в более специальных областях. В частности, силлогистические рассуждения широко применяются в различного рода диаграммах. Наиболее известны и удобны для приложений методы диаграмм Дж.Венна [12, 52] и Л.Кэррола [13].

## Глава 2. ЛОГИКА В ПРОГРАММИРОВАНИИ

### 2.1. Теория алгоритмов и логика - родители программирования

Программирование, вычислительные машины с самого начала тесно связаны с математической логикой. Идею вычислительной машины с программным управлением впервые смогли реализовать болгарский ученый С.Атанасов в 1940 г. и немецкий ученый К.Цузе в 1942 г. Они вдохновлялись воспоминанием о титанических замыслах Ч.Бэббиджа, пытавшегося реализовать подобную идею еще в первой половине XIX века, и математической теорией вычислимости, теорией алгоритмов, развитой логиками в 30-х годах.

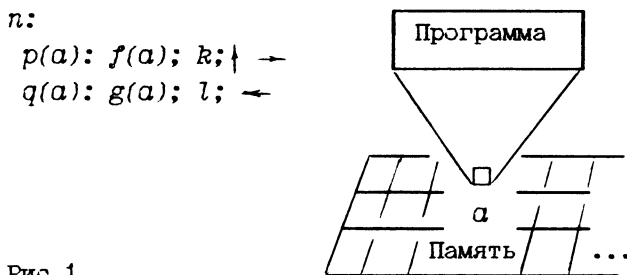


Рис. 1

В теории алгоритмов были предугаданы основные концепции, заложенные в аппаратуру и языки программирования ЭВМ. Четыре главные модели алгоритма математически эквивалентны, но на практике породили разные направления в программировании.

Первая модель – это *абстрактная вычислительная машина* (А.Тьюринг и Р.Пост, вторая половина 30-х годов). Возможное описание модели такой машины дано на рис. 1.

Вычислительная машина действует на основе конечной программы, "зашитой" в нее. Команды этой программы перенумерованы;  $n$  – номер команды, которая выполняется в данный момент. Машина обрабатывает содержимое памяти, разбитой на ячейки. В каждый момент она рассматривает содержимое одной из ячеек. Пусть это содержимое –  $a$ . Тогда машина проверяет некоторое условие  $p(a)$  (в модели Поста, где ячейки содержат натуральные числа,  $a = 0$  или  $a \neq 0$ ; в теории Тьюринга ячейка содержит символ из заранее перечисленного конечного алфавита и осуществляется просто проверка  $a = z$ ). Если это условие выполнено, то машина переходит к части команды  $n$ , соответствующей  $p(a)$ . Она производит некоторое очень простое действие  $f$  над  $a$  и переходит к одной из соседних ячеек. Следующей будет выполняться команда  $k$ . Если же  $p$  не выполнено, проверяется  $q$  и

т.д. (в определении предусмотрено, что список альтернатив  $p, q, \dots$  исчерпывающий).

Тьюринг показал, что теоретически достаточно рассматривать память как ленту, которая может неограниченно увеличиваться в одну сторону, а в каждой ячейке иметь либо 0, либо 1. Тогда единственные возможные действия — оставить содержимое ячейки тем же либо заменить его на противоположное. В книге [15] собрано множество определений абстрактных машин, и каждое из них можно свести к другому подходящей кодировке входов и выходов. Таким образом, любая функция, вычисляемая на одной машине, вычислима и на другой (конечно, если пренебречь размером требуемой памяти и временем работы машины).

Так появились важнейшие для применений логики понятия:

*вычисляемая функция* — та, для которой имеется абстрактная машина, вычисляющая ее за конечное время;

*разрешимая задача* — та, для которой имеется машина, проверяющая за конечное число шагов для любых входных данных, имеется ли решение данной задачи, и, если имеется, строящая его;

*неразрешимая задача* — задача, для которой нет такой машины.

До сих пор все вычислительные машины в некотором смысле базируются на идее Тьюринга: их память физически состоит из *битов*, каждый из которых содержит либо 0, либо 1. Более того, так называемое микропрограммное управление унаследовало от этих абстрактных машин и программу, помещенную в "постоянную память", и даже в значительной мере структуру команды.

Другим способом описания алгоритмов, идущим от Д.Гильберта и Р.Аккермана, стали *рекурсивные функ-*

ции. От них унаследовало свои основные конструкции современное структурное программирование.

Третий способ описания алгоритмов ( $\lambda$ -исчисления) базируется на идеях советского логика Р.Шейнфинкеля и американского логика Х.Б.Карри. Оказалось, что для определения всех вычислимых функций достаточно операций  $\lambda$ -абстракции и применения функции к аргументу (который может быть, в свою очередь, функцией). Идеи  $\lambda$ -исчисления активно развиваются в языке Лисп, функциональном программировании и во многих других перспективных направлениях современного программирования. С ними мы неоднократно встретимся далее.

Следующим направлением в теории алгоритмов явились нормальные алгоритмы А.А.Маркова. Они послужили основой языка Рефал и многих других языков обработки символьной информации (и в конце концов даже столь популярного сейчас Пролога).

Для успешного развития вычислительных машин громадное значение имело предложение Дж.фон Неймана поместить программу ЭВМ в основную память, чтобы иметь возможность изменять ее в ходе вычислений.

Если теория алгоритмов — в некотором смысле мать современных ЭВМ и программирования, то сама по себе математическая логика — их отец. Работы логиков первой половины XX века были посвящены, казалось бы, исключительно далекой от приложений проблеме обоснования математики. Но именно они показали и колоссальные возможности, и принципиальные ограничения формализованных языков, положили начало строгому определению таких языков и разработке техники формализации, а любая программа является формализацией.

Более того, сведение вычислимости к манипуляциям с нулями и единицами (что оказалось очень эффе-

ктивным при воплощении в "железе") буквально заставило с самого начала активно использовать в программировании булеву логику. Все помнят определения логических выражений в языках программирования и таблицы истинности. Поэтому в 50-х - 70-х годах в популярных книгах программирование представлялось как прямое приложение логики, а тем временем собственно математическая логика и программирование расходились все дальше, создавая предпосылки кризиса, который наступил в каждой из этих дисциплин к началу 60-х годов.

## 2.2. Попытки применения логики в программировании в 50-х - 70-х годах

Первые попытки применить в программировании не только булеву алгебру, но и более развитые исчисления и методы формализации предпринял Х.Б.Карри. В 1952 г. он сделал доклад "Логика программных композиций", идеи которого опередили свое время по крайней мере на 25 лет. Х.Б.Карри рассмотрел задачу программирования как составление более крупных программ из готовых кусков. Были введены две базисные системы конструкций: последовательное исполнение, разветвление и цикл, с одной стороны, и последовательное исполнение и условный переход - с другой. Он показал, какими логическими средствами можно воспользоваться для композиции программ из подпрограмм в каждом из этих случаев (для структурного программирования - естественным выводом, для переходов - матричным представлением логических взаимосвязей). Было отмечено, что для успешного применения таких методов необходима система настройки модулей на конкретное окружение, более простыми словами - привязка неполных описаний к контексту (у него даже промелькнуло сравнение параметров модуля с местоимениями естественного языка).

Эта работа была опубликована в 1954 г. и благополучно забыта, видимо, потому, что она содержала многие скептические утверждения, показывающие необоснованность глобальных притязаний тогдашней кибернетики.

При развитии программирования стала очевидной принципиальная разница между теоретическими и практическими оценками выразительной силы и возможностей языков. Конечно, теоретически на языке машин Тьюринга либо в двоичных кодах ЭВМ можно записать все, что угодно. Но для сколько-нибудь серьезной программы необходимо описание ее на концептуальном, более соответствующем специфике решаемой задачи и понятном человеку языке. Опустим пока Фортран и другие алгоритмические языки, в которых приоритет отдавался не концепциям, а требованиям реализации программы на примитивных ЭВМ примитивными средствами. Рассмотрим язык блок-схем, предназначенный не для реализации на ЭВМ, а для записи схем программ человеком.

Возьмем, например, блок-схему обеда в кафе, представленную на рис. 2.

Видно, что здесь мы имеем описание логики процесса в терминах действий и условий. Но логическая теория, соответствующая блок-схемам (простейшему, часто слишком примитивному с точки зрения программирования средству), появилась лишь в середине 80-х годов. Блок-схемы соответствовали нарождаю-

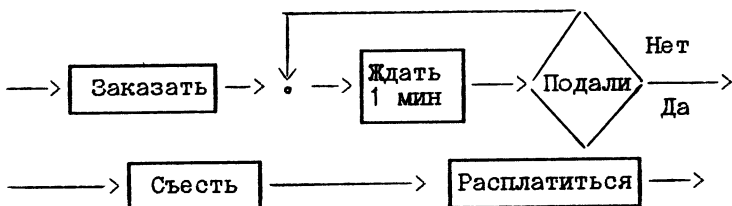


Рис.2

шемуся новому типу логики - логики, которой пользуется программист для создания сложных, многовариантных, итеративных планов действий. Компьютер не воспримет программу, написанную на не до конца формализованном языке. Он - "идеальный бюрократ", приступающий к сути дела лишь после того, как все выражено в соответствии с детальными инструкциями (естественно при этом часто теряется смысл, который первоначально вкладывался в программу). И становится ясно, что слишком часто не осознанные до конца принципиальные недостатки проявляются как множество "технических затруднений", каждое из которых по отдельности кажется преодолимим, но в совокупности они делают работу просто невыносимой.

Поэтому в 60-е годы на первое место встали потребности точного определения формальных языков достаточно сложной структуры. В математике формальные языки имели очень простую синтаксическую структуру (правда, содержащую такие особенности, воспринятые современными языками программирования, как составление выражений из вложенных друг в друга частей, понятие локализации переменной, связанной квантором, и т.п.). Все необходимые для практического использования усложнения трактовались "для простоты" как неформальные соглашения. Даже естественный вывод определялся в математической логике самыми обходными путями, часто уничтожавшими содержательную структуру вывода, лишь бы побыстрее перейти к доказательству интересных, сложных теорем. В отношении описания синтаксиса богатых формальных языков и работы с ними математическая логика получила существенную помощь от программирования.

К началу 70-х годов встали вопросы точного описания не только синтаксиса, но и семантики языков программирования.

В середине 60-х годов практически одновременно появился ряд пионерских работ в области описания условий, которым удовлетворяет программа. В.М.Глушков в 1965 г. предложил алгоритмические алгебры (впоследствии послужившие прообразом динамических логик), Ф.Энгелер в 1967 г. - использование языков с бесконечно длинными формулами, чтобы выразить бесконечное множество возможностей, возникающих при разных исполнениях программы. Но наиболее популярны стали языки *алгоритмических логик*, которые включают высказывания вида  $\{U\} S \{Z\}$ , читающиеся следующим образом: "если до исполнения оператора  $S$  было выполнено  $U$ , то после него будет  $Z$ ". Здесь  $U$  называется *предусловием*,  $Z$  - *постусловием*, либо *обещанием*  $S$ . Эти языки были изобретены практически одновременно Р.У. Флойдом (1967 г.), С.А.Р. Хоаром (1969 г.) и учеными польской логической школы (А.Сальвицкий и др., 1970 г.).

Принципиально отличный способ определения семантики программ, пригодный скорее для описания всего алгоритмического языка, а не конкретных программ, предложил в 1970 г. Д. Скотт. Он построил математическую модель  $\lambda$ -исчисления и показал, как переводить функциональное описание языка структурного программирования в  $\lambda$ -исчисление и как определить математическую модель алгоритмического языка через модель  $\lambda$ -исчисления. Работы по такой, *бенотационной*, семантике алгоритмических языков насчитываются уже тысячами. Она стала практическим инструментом построения надежных трансляторов со сложных алгоритмических языков. Так еще одна абстрактная область математической логики получила прямые приложения.

Теория алгоритмов все время успешно взаимодействовала с программированием. По этому поводу имеется немало хороших книг и популярных статей, мы



остановимся здесь лишь на некоторых, наиболее близких к собственно логике моментах.

По мере того как становилось ясно, что компьютер не только быстродействующий калькулятор, все больше места занимала машинная обработка сложно устроенной информации. Поэтому стало необходимо описывать на концептуальном уровне и сложные структуры данных, а современные языки программирования такой возможности не дают.

Средства описания и преобразования сложных структур на концептуальном уровне и их конкретизации в процессе решения задачи поставляют теория абстрактных типов данных, находящаяся на стыке теории алгоритмов логики и алгебры.

Нам необходимо описывать не только готовые программы, но и саму систему рассуждений, которой пользуется программист при построении и преобразовании программ. Логика программирования тесно связана с очень трудной (теоретически неразрешимой<sup>14</sup>) задачей автоматического синтеза программ по их условиям и постусловиям. В качестве логик программирования предлагаются различные разновидности конструктивных логик. Толчок к этому дали две работы.

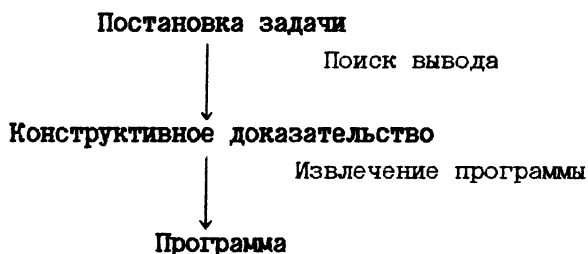
В 1968г. Х.Б.Карри показал, что в так называемой чистой импликативной интуиционистской логике доказанные теоремы однозначно соответствуют  $\lambda$ -термам, в которых каждой функции, каждому подвыражению сопоставлен тип. Чтобы пояснить разницу между  $\lambda$ -выражениями с типами и без типов, заметим, что  $\lambda$ -терм  $\lambda x.x(x)$  соответствует операции применения функции к самой себе. В  $\lambda$ -термах с типами такая возможность отсутствует. Если, например,  $x$  - пере-

---

<sup>14</sup> Следует правильно понимать, что значит "неразрешимая задача". Неразрешимость означает отсутствие общего алгоритма, единого способа решать данную задачу, но не исключает возможности найти такой способ во многих конкретных случаях.

менная типа  $a$  (обозначается  $x_a$ ), то в терме  $\lambda f.f(x)$  величина  $f$  обязана быть переменной для функции из  $a$  в некоторый тип  $b$  ( $f_{(ab)}$ ), и тогда весь этот терм имеет тип  $((ab)b)$ . Такая строгая дисциплина обеспечивает конечность вычислений  $\lambda$ -термов с типами, если все входящие в них функции исчислимы за конечное время. В интерпретации Д.Скотта существенно то, что рассматриваются  $\lambda$ -выражения без типов, а при конструировании программ естественнее как раз  $\lambda$ -термы с типами.

В том же году Э.Бишоп подметил, что, если пользоваться не классической логикой и математикой, а конструктивной, то в принципе возможна цепочка чисто формальных преобразований:



Наблюдение Бишоп стимулировало ряд работ по применению конструктивной логики к программированию.

После того как активизировались работы по логическому описанию программ и программирования, стало видно, что логический анализ позволяет находить многие скрытые недоработки в концепциях языков программирования, анализировать эти концепции. Пожалуй, до сих пор это применение строгих математических методов в качестве инструмента критики концепций (зачастую совершенно не продуманных, вставляемых ради "усиления" языков программирования) является важнейшим применением логики в данной области.

Остановимся на трех ветвях прикладной логики,

наиболее близких к задачам традиционной логики: как описывать программы с помощью логики, как описывать само программирование, как анализировать концепции логическими средствами.

### 2.3. Логика описания программ

Как известно, правильность созданной программы обычно проверяется на ряде так называемых *тестовых примеров*, на начальных данных, для которых возможно предсказать результат. Неоднократно отмечалось, что такая проверка может лишь выявить наличие ошибок, но не доказать их отсутствие. Поэтому важна задача строгого доказательства правильности программ, и именно для этой цели и начали разрабатываться программные и динамические логики.

Программа в ее традиционной интерпретации определяет последовательность действий над исходными (входными) данными, в результате которой получают выходные данные. Данные хранятся в памяти машины, и любое действие может преобразовывать их (см. абстрактную машину, рис.1).

Рассмотрим простейший алгоритмический язык  $\mathcal{L}_0$ .

Память в  $\mathcal{L}_0$  считается разделенной на *ячейки*, каждая из ячеек имеет свое имя, обозначаемое *идентификатором* - словом из латинских букв и цифр, начинающимся с буквы. И обратно, каждому идентификатору, имеющемуся в программе, сопоставлена ячейка. Традиционно, для простоты, считают, что ячейки содержат натуральные числа.

Программа на  $\mathcal{L}_0$  состоит из *операторов*. Исходный оператор - *присваивание*:

$$x := ex;$$

где  $x$  - идентификатор,  $ex$  - выражение, составленное из идентификаторов и натуральных чисел при помощи символов и арифметических действий. При присваивании вычисляется значение  $ex$  с использованием

старого значения  $x$ , а затем содержимое  $x$  заменяется на вычисленное значение. Старое значение  $x$  при этом пропадает. Например,  $x:=x+1$ ; увеличивает значение  $x$  на 1. Такая интерпретация присваивания элементарно описывается и логически.

Пусть состояние памяти, создавшееся после присваивания, удовлетворяет условию  $U(x)$  (здесь  $U$  можно понимать как формулу языка формальной арифметики); тогда то состояние, которое было до присваивания, можно описать формулой  $U(x|\varepsilon x)$ . В программной логике такое утверждение записывается:

$$\{U(x|\varepsilon x)\} x:=\varepsilon x; \{U(x)\}.$$

Например, чтобы после присваивания  $x:=x+1$ ; стало  $x < y$ , необходимо и достаточно, чтобы до него было  $x+1 < y$ :

$$\{x+1 < y\} x:=x+1; \{x < y\}.$$

Уже для присваиваний видно, как различаются логическое и "операциональное" истолкования программ: если для операционального истолкования легко объяснить, что будет после исполнения той или иной конструкции, но редко можно восстановить то, что было до ее исполнения, то при логическом — наоборот.

Далее, операторы могут соединяться в последовательность один за другим:  $S_1 S_2 \dots S_n$ . В этом случае они последовательно и выполняются. Соответственно, в логической записи:

$$\frac{\{U\} S \{B\}, \{C\} T \{D\} \quad B \rightarrow C}{\{U\} ST \{D\}}$$

Следующая конструкция  $\square_0$  — условный оператор:

$$\text{IF } A_1 \rightarrow S_1 \square A_2 \rightarrow S_2 \square \dots \square A_n \rightarrow S_n \text{ FI};$$

Здесь  $A_1, A_2, \dots, A_n$  — логические выражения, построенные из отношений  $\varepsilon x_1 = \varepsilon x_2, \varepsilon x_1 > \varepsilon x_2$  при помощи связок классической логики высказываний;  $S_i$  — по —

следовательности операторов. Проверяются формулы  $A_i$  при нынешнем содержимом памяти. Если ни одна из  $A_i$  не истинна, то фиксируется ошибка и исполнение программы аварийно завершается. Если же некоторые  $A_i$  истинны, то выбирается одна из них (каким образом, мы не интересуемся) и выполняется соответствующая последовательность  $S_i$ . Поскольку доступ к  $S_i$  возможен лишь при выполнении  $A_i$ , выражение  $A_i$  называется *охраной*  $S_i$ , а  $S_i$  — *охраняемой командой*.

Если каждая из команд  $S_i$  описана как  $\{U_i\}S_i\{Z\}$ , то весь условный оператор описывается утверждением

$$\{(A_1 \supset U_1) \& (A_2 \supset U_2) \& \dots \& (A_n \supset U_n) \& (A_1 \vee \dots \vee A_n)\} \\ \text{IF } A_1 \rightarrow S_1 \square A_2 \rightarrow S_2 \square \dots \square A_n \rightarrow S_n \text{ FI} \\ \{Z\}.$$

Содержательно такое утверждение означает, что для получения желаемого результата на выходе условного оператора необходимо, чтобы к моменту его выполнения хотя бы одна из охранных была истинна и если истинна  $A_i$ , то выполнялось бы предусловие соответствующего оператора  $S_i$ .

Заметим, что логическая формулировка дает указания, когда целесообразно применять условный оператор. А именно, если нельзя достичь нынешней цели  $Z$  сразу, но можно достичь ее в ряде частных случаев, то целесообразно попытаться объединить эти случаи. Конечно, для построения IF необходимо, чтобы разбиение исчерпывало все возможности, доступные в данный момент (истинность  $(U_1 \vee \dots \vee U_n)$ ), но этого недостаточно. Выражения  $U_i$  могут быть сложными утверждениями, содержащими кванторы, а проверка истинности таких формул не может быть проведена чисто механически, компьютером. Мы должны заменить их простыми, легко проверяемыми

формулами  $A_1, \dots, A_n$ . Порою для этого нужно про-  
 делать предварительные вычисления, определить не-  
 которые вспомогательные величины для сравнений.

Описание условного оператора логическими средс-  
 твами ставит некоторые вопросы. Во-первых, не лю-  
 бое утверждение проверяемо, а охранами могут быть  
 лишь проверяемые утверждения. Во-вторых, исполне-  
 ние условного оператора часто недетерминировано,  
 имеется неопределенность, какое из истинных  $A_i$  вы-  
 брать? Эта недетерминированность является еще од-  
 ной из характерных черт логического описания язы-  
 ка программирования, в отличие от функционального.

Циклы в значительной степени определяют специ-  
 фику современных языков программирования. Оператор  
 цикла имеет вид

$$\begin{aligned} \text{DO } A_1 \rightarrow S_1 \square A_2 \rightarrow S_2 \square \dots \square A_n \rightarrow S_n \text{ OUT } B_1 \rightarrow T_1 \square B_2 \rightarrow T_2 \square \dots \\ \dots \square B_k \rightarrow T_k \text{ OD;} \end{aligned}$$

Исполняется он следующим образом. Как и в устном  
 операторе, проверяется, истинна ли хоть одна из  
 охран  $A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_k$ . Если ни одна из  
 них не истинна – ошибка. Если же истинные имеются,  
 выбирается одна из них. Если выбрана  $A_i$ , то  
 выполняется соответствующая последовательность  $S_i$   
 и выполнение оператора цикла возобновляется. Если  
 выбрана  $B_j$ , то выполняется соответствующее  $T_j$  и  
 выполнение цикла завершается.

Итак, цикл может завершиться на первом, на  
 втором и т.д., на каком угодно шаге. Если же он  
 продолжается бесконечно, то программа *зациклилась*  
 и она некорректна при данных, поданных ей на вход.

Логическое описание цикла сначала идет по  
 аналогии с условным оператором. Пусть каждая из  $S_i$   
 описана как  $\{U_i\} S_i \{G_i\}$ , каждая из  $T_i$  описана как  
 $\{D_i\} T_i \{B_i\}$ . Тогда условия при однократном выпол-  
 нении цикла можно описать как

$$\begin{aligned}
& \{ (A_1 \supset U_1) \& (A_2 \supset U_2) \& \dots \& (A_n \supset U_n) \& \\
& \& (B_1 \supset D_1) \& (B_2 \supset D_2) \& \dots \& (B_k \supset D_k) \& \\
& \& (A_1 \vee \dots \vee A_n \vee B_1 \vee \dots \vee B_k) \} \\
& \text{IF } A_1 \rightarrow S_1 \square A_2 \rightarrow S_2 \square \dots \square A_n \rightarrow S_n \square B_1 \rightarrow T_1 \square \dots \square B_k \rightarrow T_k \text{ FI;} \\
& \{ \mathfrak{B} \}.
\end{aligned}$$

При двукратном выполнении получаем

$$\begin{aligned}
& \{ (A_1 \supset U_1) \& (A_2 \supset U_2) \& \dots \& (A_n \supset U_n) \& \\
& \& \{ (B_1 \supset D_1) \& (B_2 \supset D_2) \& \dots \& (B_k \supset D_k) \& \\
& \& \{ (A_1 \vee \dots \vee A_n \vee B_1 \vee \dots \vee B_k) \} \\
& \text{IF } A_1 \rightarrow S_1 \square A_2 \rightarrow S_2 \square \dots \square A_n \rightarrow S_n \square B_1 \rightarrow T_1 \square \dots \square B_k \rightarrow T_k \text{ FI;} \\
& \{ \mathfrak{C}_1 \vee \dots \vee \mathfrak{C}_n \} \\
& \{ (A_1 \supset U_1) \& (A_2 \supset U_2) \& \dots \& (A_n \supset U_n) \& \\
& \& (B_1 \supset D_1) \& (B_2 \supset D_2) \& \dots \& (B_k \supset D_k) \& \\
& \& (\mathfrak{C}_1 \vee \dots \vee \mathfrak{C}_n) \& (A_1 \vee \dots \vee A_n \vee B_1 \vee \dots \vee B_k) \} \\
& \text{IF } A_1 \rightarrow S_1 \square A_2 \rightarrow S_2 \square \dots \square A_n \rightarrow S_n \square B_1 \rightarrow T_1 \square \dots \square B_k \rightarrow T_k \text{ FI;} \\
& \{ \mathfrak{B} \}
\end{aligned}$$

и т. д. до бесконечности. Условие правильности  $m$  шагов цикла обозначим  $\text{DO}^m \{ \mathfrak{B} \vee \mathfrak{C}_1 \vee \dots \vee \mathfrak{C}_n \}$ . Цикл корректен, если

$$\text{DO}^1 \mathfrak{B} \vee \text{DO}^2 \mathfrak{B} \vee \dots \vee \text{DO}^m \mathfrak{B} \vee \dots$$

Итак, условие правильности цикла выражается бесконечной формулой (вот откуда взял К.Энгелер язык бесконечно длинных формул  $\mathcal{L}_{\omega_1 \omega}!$ ). Доказано, что задача, имеется ли его конечное выражение, неразрешима для языка логики предикатов и тем более для арифметики.

Вся история алгоритмической логики – это борьба с возникающей бесконечностью. Можно просто игнорировать ее, но тогда она мстит за себя неполнотой либо некорректностью построенных систем. Можно

увеличивать либо уменьшать выразительную силу логического языка, чтобы сформулировать условия на цикл в конечном виде.

Задачи, связанные с формализацией алгоритмических логик, описывающих более сложные структуры данных, оказались на том же уровне принципиальной сложности, что и для  $\mathcal{L}_0$ , но настолько тонкими, что, как было отмечено в обзоре К.Апта, две трети систем, построенных за первые десять лет развития программных логик, оказались неверны.

Заметим, что сразу же возникают аналогии программных логик с модальными и временными. Каждое состояние памяти, возникающее в ходе исполнения программы, аналогично "возможному миру" в семантике неклассических логик; пути исполнения программы определяют переходы между этими мирами. На самом деле у нашего основного утверждения  $\{U\} S \{Z\}$  имеется много интерпретаций. Мы неявно использовали самую "жесткую" из них, так называемую интерпретацию тотальной корректности  $S$ : исполнение  $S$  обязательно завершается и  $Z$  должно быть истинно в любом состоянии, которое может получиться после исполнения  $S$ . Это соответствует в модальной логике оператору необходимости  $\Box Z$ .

Другая возможная интерпретация - частичная корректность программы, когда снимается требование обязательного завершения. Первоначально в программной логике работали в терминах тотальной и частичной корректности. Все новые, во многих случаях полезные возможности логического описания программ (утверждение всегда истинно при некоторых путях исполнения, периодически истинно на них и т.п.) и требования завершенности и удобства формализмов заставили обобщить языки программных логик до языков логик алгоритмических.

Прежде всего, выражения  $\{U\} S \{Z\}$ , рассматривав-



шиеся в программной логике как "надстройка" над логическим и алгоритмическим языками, стали рассматриваться как равноправные компоненты логического языка. Они могут соединяться связками с другими формулами, на которые могут навешиваться кванторы и которые могут появиться даже внутри выражений  $\mathcal{E}$  и  $\mathcal{D}$  в  $\{\mathcal{E}\}T\{\mathcal{D}\}$ . Сразу после этого язык претерпел две модификации.

Выражение  $\{U\} S \{Z\}$  оказывается логически эквивалентно  $U \supset \{U\} S \{Z\}$ , где  $\top$  - тавтология, и достаточно рассматривать выражения вида  $\{S\}U$ , где  $S$  - программа,  $U$  - формула. А здесь сразу становится видна синтаксическая аналогия между программами и модальностями либо временными связками, дополняющая ранее отмеченную семантическую. Далее, углубляя эту аналогию, целесообразно "раздвоить" программные операторы на необходимые и возможные:  $\Box\{S\}U$  и  $\Diamond\{S\}U$ . То, что программа всегда завершается, теперь выражается записью  $\neg\Diamond\{S\}\bot$ , где  $\bot$  - противоречие; тотальная корректность -  $\neg\Diamond\{S\}\bot \ \& \ \Box\{S\}U$ . Из этих примеров видно, что язык становится намного выразительнее и богаче. Но остаются все проблемы, связанные с циклами.

Если теперь резко упростить сам логический язык, отказавшись от предикатов, переменных и кванторов, то становится незачем явно выделять переменные и в алгоритмической части. Мы просто имеем некоторый алфавит исходных операторов и средства композиции более сложных (последовательность, IF и DO). Так получается язык *динамической логики*, которые зачастую имеют полную формализацию.

Множество возникающих принципиальных и технических трудностей привело к тому, что задача верификации программ - автоматизированного доказательства их правильности - остается до сих пор нерешенной.

## 2.4. Логическое описание программирования

Мы видели, что уже для описания программы оказалось целесообразно воспользоваться аппаратом неклассической логики. Тем более это верно для задачи описания процесса программирования, когда самой программы еще нет и ее необходимо найти.

Для логиков было достаточно очевидно, что классическая логика плохо подходит для описания процесса программирования, поскольку она плохо подходит для описания процесса в математике. Еще в начале XX века было замечено, что в математике давно разошлись понятия "существовать" и "быть построенным", трактовавшиеся с античных времен как синонимы. Аксиома выбора в теории множеств, в частности, утверждает, что можно построить функцию, выбирающую по одному элементу из заданного семейства непустых множеств, не давая никакого способа определить эту функцию. Позднее, уже в 60-х годах, было строго доказано, что в некоторых случаях нет возможности явно построить множество, существование которого доказывается при помощи аксиомы выбора.

**Пример 1.** Неизмеримое множество действительных чисел строится следующим образом. Числа из отрезка  $[0,1]$  делятся на классы эквивалентности по отношению "Разность между  $x$  и  $y$  рациональна". Неизмеримое множество "строится" путем выбора по одному элементу из каждого такого класса. Доказано, что ни одно множество, которое можно определить в теории множеств, не является неизмеримым.

**Предупреждение.** Это не означает, что любое определимое множество является неизмеримым! Еще А.А.Лузиным было построено множество, для которого в 70-х годах удалось доказать недоказуемость теории множеств, утверждения о том, что оно измеримо.

А.Пуанкаре, один из величайших мыслителей среди

математиков XIX–XX веков предложил пересмотреть аксиоматику математики с тем, чтобы исключить подобные "монстры". Но выяснилось, что причина появления существующих объектов, которые не могут быть построены, гораздо глубже.

**П р и м е р 2** (парадокс Института математики). Пусть институт математики взялся выявить оптимальные параметры некоторого процесса и после глубоких исследований выдал абсолютно точный ответ: оптимальное значение 0, если не существует нечетное совершенное число, и  $\ln \sin n$ , если  $n$  – наименьшее такое число. Нужно ли кому-нибудь это исчерпывающее решение?

**П р и м е р 3** (логическая переформулировка данного парадокса). Известно, что в любой достаточно сильной математической теории имеется неразрешимое утверждение  $\Pi$ . Определим число  $n$  следующим образом:  $(\Pi \supset n=0) \ \& \ (\neg \Pi \supset n=1)$ . Пользуясь законом исключенного третьего,  $\Pi \vee \neg \Pi$ , можно доказать существование  $n$ , но нельзя построить ни одно  $n$ , про которое можно сказать, что оно удовлетворяет приведенному условию.

Итак, причиной появления "приведений", которые существуют, но построить их нельзя, является эффект сочетания классической логики с теоремой Геделя о неполноте. Закон  $\Pi \vee \neg \Pi$  означает, что мы знаем все. А этот постулат никак нельзя назвать реалистическим: мы знаем очень мало, и чем больше знаем, тем больше не знаем.

Л.Э.Я. Брауэр определил логические корни "приведений" еще до появления теоремы Геделя, в 1908 г., и начал построение *интуиционистской* математики, не принимающей закон  $\Pi \vee \neg \Pi$  как универсальный. В 1930–1932 гг. А.Гейтинг строго сформулировал логику, которой пользовались в *интуиционистской* математике (*интуиционистская логика*). Ее мате-

матическая интерпретация, данная А.Н.Колмогоровым в то же время, сохранила свое значение до сих пор.

Колмогоров рассмотрел логику как *исчисление задач*. Каждая формула – не просто утверждение об истинности какого-то предложения, но требование решить задачу, т.е. построить объект, удовлетворяющий некоторым условиям. Это – *конструктивная* интерпретация высказываний. Логические связки понимаются как средства построения формулировок более сложных задач из более простых, аксиомы – как задачи, решения которых даны, правила вывода – как способы преобразования решений одних задач в решения других. Отметим, что решение задачи – это не только сам искомый объект, но и доказательство того, что он удовлетворяет требованиям.

Например, формула  $U \ \& \ \mathfrak{Z}$  понимается в колмогоровской интерпретации как задача, состоящая в том, чтобы построить и решение  $U$ , и решение  $\mathfrak{Z}$ , правило вывода  $U \ \dots \mathfrak{Z} \ / \ U \ \& \ \mathfrak{Z}$  – как преобразование, состоящее из объекта  $a$ , решающего задачу  $U$ , и  $b$ , решающего  $\mathfrak{Z}$ , пару  $(a, b)$ , решающую  $U \ \& \ \mathfrak{Z}$ .

Объект  $a$ , решающий задачу, сопоставленную  $U$ , называется *реализацией*  $U$  (это будем обозначать  $a \in U$ ).

Центральным моментом конструктивного понимания логических формул является интерпретация импликации. Конструктивную импликацию, в отличие от классической, будем обозначать  $\Rightarrow$ . Выражение  $U \Rightarrow \mathfrak{Z}$  понимается как требование построить эффективное преобразование  $f$ , применимое ко всем реализациям  $U$  и перерабатывающее их в реализации  $\mathfrak{Z}$ .

Интерпретация Колмогорова нестрога, хотя и сформулирована в математических терминах. Он не уточнил, каковы классы объектов, являющихся реализациями (в частности, каковы классы эффективных преобразований в интерпретации  $U \Rightarrow \mathfrak{Z}$ ), на каком

языке формулируются требования к реализациям и как проводится доказательство того, что  $\alpha \vdash \Pi$ . Это было неизбежно, так как в начале 30-х годов еще не сформировалось ни понятие алгоритма, ни общее понятие исчисления. Но неточность является не столько недостатком колмогоровской формулировки, сколько ее достоинством. Эта формулировка может конкретизироваться по-разному, давая целую схему точных математических определений, и соответственно применяться не только к интуиционистской логике, для которой она была создана, но и к другим логическим системам, о которых еще и не подозревали в те времена.

Строгие математические семантики, основанные на идее Колмогорова, обычно называют *семантиками реализуемости*. Первую реализуемость построил С.К. Клини в 1940 г. для арифметики с интуиционистской логикой. В качестве уточнения понятия функционала использовалось понятие частично-рекурсивной функции, а все искомые объекты кодировались натуральными числами.

Корректность логической системы относительно реализуемости дает некоторые надежды на то, что ее можно использовать для описания процесса построения математических объектов. А именно, поскольку каждая выводимая формула реализуема, в принципе существует транслятор, преобразующий выводы в построения реализаций. Другой вопрос, насколько трудоемок сам процесс трансляции и насколько практичны полученные с его помощью решения.

В 60-х - 80-х годах появились десятки понятий реализуемости, как для систем, базирующихся на интуиционистской логике, так и для других логик. А.А. Воронков в 1985 г. вывел даже условия, при которых классическая логика может рассматриваться как конструктивная (это происходит, если все фор-

мулы для нашей модели представляют алгоритмически разрешимые отношения между значениями своих свободных переменных. Необходимым (но недостаточным) условием конструктивности классической теории является ее полнота, т.е. выводимость для каждой замкнутой формулы  $U$  либо  $U$ , либо  $\neg U$ . Таким образом, еще раз подтверждается прозрение Брауэра относительно логических корней неконструктивности<sup>15</sup>.

Стоит отметить принципиальное отличие реализуемости от других видов логических семантик (которые базируются на системах истинностных значений). Эквивалентные формулы имеют одинаковые истинностные значения, истинны в одних и тех же возможных мирах, но они обычно имеют различные реализации. Например, если  $(a,b)$  — реализация  $U \ \& \ \mathfrak{B}$ , то реализацией  $\mathfrak{B} \ \& \ U$  служит  $(b,a)$ .

Еще одну переформулировку идеи Колмогорова о логикс как исчислении задач дал Н.А. Шанин в 1956 г. Он сформулировал *алгоритм конструктивной расшифровки*, который переводил формулу языка арифметики  $U$  в  $\exists \xi \ \exists \ \mathfrak{U}$ , где  $\xi$  — новая переменная, вводимая для обозначения объекта, который должен быть построен при доказательстве  $U$ ;  $\exists \ \mathfrak{U}$  — формула классической логики с параметром  $\xi$ . Таким образом, конструктивное истолкование формулы было сведено к построению объекта и доказательству обычного, классического утверждения о построенном объекте. При этом было показано, что строящиеся объекты делятся на типы в зависимости от логической структуры  $U$ , но не от ее доказательства; данное наблю-

<sup>15</sup> Заметим, что примерами классических теорий, имеющих конструктивное истолкование, служат элементарная геометрия и алгебраическая теория вещественных чисел; одним из главных препятствий конструктивности классической теории служит явное упоминание натуральных чисел.

дение было неоднократно использовано впоследствии, частности, П. Мартин-Лёфом.

### *Конструктивность и структура выводов*

Рассмотрим простейшие примеры того, как требование конструктивности влияет на логические понятия.

Пусть задано множество пропозициональных букв  $\mathcal{P}$ . Формулы – это слова вида  $p \Rightarrow q$ , где  $p, q \in \mathcal{P}$ ,  $\Rightarrow$  интерпретируется как конструктивная импликация.

Семантику формул зададим следующим образом. Пусть имеется некоторое множество состояний системы  $\mathcal{S}$ . Каждому  $p \in \mathcal{P}$  сопоставляем некоторое подмножество  $P \subseteq \mathcal{S}$  (пропозициональные буквы интерпретируются как предикаты на множестве состояний). Задается множество  $\mathcal{F}$  бинарных отношений на  $\mathcal{S}$ . Запись  $f \in \mathcal{F}$  понимается как недетерминированные преобразования: если  $(s, t) \in f$ , то  $f$  преобразует  $s$  в  $t$ . Символ  $f$  реализует  $p \Rightarrow q$ , если для любого  $s$  из  $P$  найдется  $t$  из  $Q$ , такое, что  $f$  преобразует  $s$  в  $t$ , и для всех  $s \in P$ , если  $(s, t) \in f$ , то  $t \in Q$ <sup>16</sup>. Таким образом, преобразование  $f$  должно обеспечивать попадание из  $s$  в  $t$ .

**Пример 1.** В качестве  $\mathcal{S}$  рассмотрим множество состояний вычислительной машины и ее внешних устройств,  $f$  – действия типа "послать данные на устройство", "включить устройство". Предикаты из  $\mathcal{P}$  описывают состояния процессора либо устройств ("данные готовы", "устройство включено" и т.п.).

**Пример 2.**  $\mathcal{S}$  – множество состояний памяти, которая может содержать данные, найденные в большой внешней базе, предикаты из  $\mathcal{P}$  интерпретируются как утверждения о том, что некоторые данные найдены и списаны в память. Действия  $f$  – поиск новых данных из уже найденных.

---

<sup>16</sup> Сравните это определение с понятием тотальной коррекции программ!

Если множество  $\mathcal{F}$  незамкнуто относительно композиции функций, то вообще не может идти речи о применении конструктивной логики в качестве инструмента планирования преобразований. Но, как показывают наши примеры, в обычных случаях  $\mathcal{F}$  естественно считать подгруппой отношений (т.е. если  $f$  и  $g$  принадлежат  $\mathcal{F}$ , то и  $f.g$  — тоже). Тогда, если имеются реализации  $f \in \mathcal{U} \Rightarrow \mathcal{B}$  и  $g \in \mathcal{B} \Rightarrow \mathcal{C}$ , то  $f.g \in \mathcal{U} \Rightarrow \mathcal{C}$ , пространство реализаций диктует нам первую конструктивную систему с единственным правилом — *правилом силлогизма*:

$$\frac{\mathcal{U} \Rightarrow \mathcal{B}, \mathcal{B} \Rightarrow \mathcal{C}}{\mathcal{U} \Rightarrow \mathcal{C}}$$

Аксиом у данной конструктивной логики нет; она приспособлена лишь для вывода одних утверждений из других в конкретных прикладных теориях, описывающих заданную систему элементарных действий.

Но правило силлогизма оказывается неадекватно нашим построениям по следующей причине. Пусть, например, дано, что

$$f \in \mathcal{P} \Rightarrow q, \quad g \in \mathcal{Q} \Rightarrow r, \quad h \in \mathcal{R} \Rightarrow s.$$

Тогда формула  $\mathcal{P} \Rightarrow \mathcal{S}$  имеет два неизоморфных вывода:

$$\frac{p \Rightarrow q, \quad q \Rightarrow r}{p \Rightarrow r, \quad r \Rightarrow s} \quad \text{и} \quad \frac{q \Rightarrow r, \quad r \Rightarrow s}{p \Rightarrow q, \quad q \Rightarrow s}$$

$$\frac{p \Rightarrow r, \quad r \Rightarrow s}{p \Rightarrow s} \quad \quad \frac{p \Rightarrow q, \quad q \Rightarrow s}{p \Rightarrow s}$$

порождающих одну и ту же реализацию  $f.g.h$  (ввиду ассоциативности композиции). Поэтому мы перейдем к системе естественного вывода с двумя обычными правилами — дедукции и *modus ponens*:

$$\frac{\begin{array}{c} \mathcal{U} \\ \vdots \\ \mathcal{B} \end{array}}{\mathcal{U} \Rightarrow \mathcal{B}} \quad \text{и} \quad \frac{\mathcal{U}, \mathcal{U} \Rightarrow \mathcal{B}}{\mathcal{B}}.$$



В такой системе формула  $p \Rightarrow s$  имеет один основной вывод:

$$\begin{array}{|l} * p \quad p \Rightarrow q \\ q \quad q \Rightarrow r \\ r \quad r \Rightarrow s \\ s \end{array} \quad \hline p \Rightarrow s$$

Полученное на очень простом примере заключение о том, что для описания построений лучше подходит естественный вывод, чем, например, резолюции и системы гильбертовского типа, справедливо и для гораздо более сложных систем. Но переход к естественному выводу заставляет задать целый ряд вопросов.

1. Можно ли повторно использовать уже использованный результат вида  $U \Rightarrow B$ ?

2. Должен ли вспомогательный вывод содержать хоть одно действие или можно его допущение прямо делать его результатом?

3. Можно ли иметь во вспомогательном выводе "тупиковые" ветви, неиспользуемые формулы?

Вопросы 1 и 3 в нашей системе сводятся к допустимости двух вариантов одного и того же правила (обычно явно не формулируемого и упрятываемого глубоко в контекстные условия) — *правила разложения*:

$$\frac{U}{U \dots U} .$$

В первом случае  $U$  имеет вид  $p \Rightarrow q$ , во втором —  $p$ .

Вопрос 2 в нашей логике по существу влияет лишь на выводимость формул  $p \Rightarrow p$ . Если имеются вырожденные выводы вида

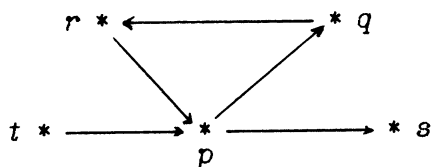
$p$  — допущение и результат,

то  $p \Rightarrow p$  выводимо всегда, в противном случае — лишь если имеется цепочка импликаций вида  $p \Rightarrow p_1$ ,  $p_1 \Rightarrow p_2$ , ...,  $p_n \Rightarrow p$ .

Конструктивный смысл формулы  $p \Rightarrow p$  состоит в том, что у нас имеется возможность сохранять нынешнее состояние мира до бесконечности.

Вопрос 1 при содержательном конструктивном переводе сводится к тому, даны ли исходные функции нам как физически реализованные блоки, которых ограниченное количество и которые должны быть в буквальном смысле изъяты со склада и вставлены в реализуемую систему. В нашей системе любое решение этого вопроса не влияет на множество выводимых импликаций  $p \Rightarrow q$ , но оно влияет на множество выводов этих импликаций, причем выводов, реализованных по-разному.

Пример 3. В теории с аксиомами  $t \Rightarrow p$ ,  $p \Rightarrow q$ ,  $q \Rightarrow r$ ,  $r \Rightarrow p$ ,  $p \Rightarrow s$ ; имеется бесконечное число выводов  $t \Rightarrow s$  при положительном ответе на вопрос 1 и всего один при отрицательном. Эти выводы соответствуют путям в графе



В нем один путь без циклов из  $t$  в  $s$  и бесконечно много путей с циклами.

Ответ на вопрос 3 также имеет содержательное конструктивное истолкование. Если мы, получив  $p$ , далее совершим действие  $p \Rightarrow q$ , а потом, убедившись, что  $q$  нам ни к чему, вернемся к  $p$  и совершим переход  $p \Rightarrow r$ , то выполненное действие является обратимым. Например, выключив внешнее устройство, мы, как правило, теряем данные, имевшиеся в его буфере, и вернуться к исходному состоянию не удастся. А вычислив лишний параметр, мы можем просто не использовать его, что ничему не повредит. Поэтому положительный ответ на вопрос 3 допустим, напри-

мер, для интерпретации примера 2 о базах данных, по не примера 1 о внешних устройствах.

Заметим, что множество выводимых импликаций не зависит от ответа на вопрос 3, от него опять-таки зависит лишь множество возможных выводов.

Заметим также, что, хотя формально вопросы 1-3 не зависимы один от другого, ответив положительно на вопрос 3, мы должны, исходя из содержательной интерпретации, ответить положительно на оба других вопроса. В самом деле, если уже использованное действие не может быть использовано повторно, то все действия в некотором смысле необратимы. Если все действия обратимы, то возможно и бездействие (ответ на вопрос 2). Такая ситуация концептуальной зависимости, когда формально независимые утверждения на самом деле содержательно взаимосвязаны, все время возникает в логическом анализе понятий (в первую очередь, понятий программирования). В последние годы появились средства формализовать и это понятие, и связанные с ним понятия концептуальной противоречивости и прагматического следствия (когда, приняв II, мы из "практических" соображений обязаны принять и III).

Если на вопросы 1-3 дан положительный ответ, то система приобретает редкое и практически важное свойство: можно исполнять действия в процессе поиска вывода. В остальных же случаях исполнение не оформленного до конца вывода может привести к столь же печальным последствиям, как и лечение больного, которому не поставлен диагноз. Заметим, что считающийся во многих работах по искусственному интеллекту эталоном логического программирования язык Пролог основан на предположении, что выполнено указанное выше свойство.

Из анализа простейших конструктивных систем мы можем сделать некоторые выводы.

1. Обычно лучше соответствует структура построений системы естественного вывода.

2. Недостаточно для проверки адекватности конструктивной системы рассматриваемому классу построений проверять множество теорем; необходимо проверять и множество возможных выводов.

3. Часто свойства рассматриваемого класса задач естественнее выражаются изменением структуры выводов, а не правил вывода либо аксиом.

4. Очень важно для приложений рассматривать не только готовые выводы, но и различные стадии их поиска — *незавершенные выводы*.

5. Пролог в значительной степени дезориентирует исследования в области логического программирования, концентрируясь на исключительном случае, когда можно исполнять незавершенный вывод.

Чтобы использовать как можно эффективнее сделанные наблюдения, оказалось целесообразно пересмотреть и само понятие вывода, перейдя к выводам в форме графов, и явно ввести в рассмотрение наряду с исчислениями завершенных выводов, исчисление незавершенных.

### *Импликативная логика и формулы как типы*

Пусть задано множество пропозиционных букв  $P$ . Формулы строятся из букв  $p \in P$  при помощи неограниченного числа применений конструктивной импликации  $\Rightarrow$ . Если рассмотреть исчисление естественного вывода, образующееся из классического оставлением лишь правил дедукции и *modus ponens*, явно говорящих об импликации, то получим систему чистой импликативной логики (PII). В ней три вопроса из предыдущего параграфа решаются однозначно: мы должны принять правило размножения и соответственно мы можем повторением  $\cup$  вывести принцип пассивности мира  $\cup \Rightarrow \cup$

Приведем пример вывода из аксиом в PII.

Т е о р и я :  $a \Rightarrow b, a \Rightarrow c, c \Rightarrow d, b \Rightarrow (d \Rightarrow e),$   
 $(a \Rightarrow e) \Rightarrow (k \Rightarrow l), m \Rightarrow k, l \Rightarrow p.$

Ц е л ь :  $m \Rightarrow p.$

В ы в о д :

*	$a$	допущение
	$a$	размножение
	$b$	$c$
	$d$	
	$e$	результат
	$a \Rightarrow e$	
	$k \Rightarrow l$	
*	$m$	допущение
	$k$	
	$l$	
	$p$	результат
	$m \Rightarrow p$	

Если каждой пропозициональной букве сопоставляется тип данных, то каждой импликации сопоставляется тип функционалов, перерабатывающих либо сами данные, либо функции, их обрабатывающие. Например,  $(a \Rightarrow e) \Rightarrow (k \Rightarrow l)$  преобразовывает функцию из типа  $a$  в тип  $e$  в функцию из типа  $k$  в тип  $l$  (в  $\lambda$ -исчислении с типами такой функции функционал имеет тип  $((ae)(kl))$  ).

Обозначив реализации аксиом через встречающиеся в них буквы (например, реализацию  $a \Rightarrow b$  через  $ab$ ), получаем следующую реализацию  $m \Rightarrow p$  как  $\lambda$ -терм с типами:

$$ae_{(ae)} = \lambda a_a (bde_{(b(de))} (ab_{(ab)} (a)_b)_{de} (ac_{(ac)} (a)_c)_{de})_e$$

$$mp = \lambda m_m (lp_{(lp)} (alkl_{((al)(kl))} (al)_{(kl)} (mk_{mk} (m)_k)_{kl})_p).$$

Здесь сразу видно, что:

- 1) структура вывода изоморфно соответствует извлекаемой из него программе;
- 2) формулы, встречающиеся в выводе, изоморфны

встречающимся в программе типам данных и функций.

Это наблюдение о типах в значительной мере переносится и на более сложные и выразительные языки и легло в основу целой теории в приложениях конструктивной математики: исчисления "формул как типов" П. Мартин-Лефа.

Наблюдение же об изоморфности вывода и извлекаемой из него программы, подобно более примитивному случаю Пролога, сыграло дезориентирующую роль, но на следующем уровне; на самом деле поддержать такой изоморфизм для достаточно выразительных языков почти невозможно.

### *Логика схем программ*

Логика схем программ, изобретенная в 80-х годах, использует наблюдения, сделанные нами при анализе простейшей импликативной логики.

Мы видели, что одним из труднейших мест в логическом описании программ является цикл. Основная сложность в том, что каждый шаг цикла должен приближать нас к его завершению, а это не просто описать в обычной логике, где, как известно, невыразимо даже понятие конечного множества. Но на этот факт можно посмотреть с другой стороны, попытавшись использовать его как преимущество. Ведь в некотором смысле нам надо обеспечить *необратимость* каждого действия, значит, надо отказаться от принципа  $\mathcal{U} \Rightarrow \mathcal{U}$ , надо отказаться от "пустых" действий, "пустых" выводов и запретить использовать выражение  $\mathcal{U}$  после того, как оно один раз использовано.

Вся эта система неформальных гипотез имеет и строгую математическую модель.

В математике известны *ординальные числа*, обобщающие понятие натурального числа:

$$1, 2, \dots, \omega, \omega+1, \dots, 2\omega, \dots, \pi\omega, \dots, \omega^2, \dots, \omega^{(\omega)}, \dots, \omega^{\omega^{\omega}}, \dots, \varepsilon_0.$$

Любая убывающая последовательность натуральных

чисел за конечное число шагов достигает 0. То же свойство остается справедливым и для ординальных чисел.

Рассмотрим некоторое множество состояний системы  $\mathcal{S}$  и построим на его основе множество ситуаций  $\mathcal{Z}$ , состоящее из пар  $w = (z, \alpha)$ ,  $z \in \mathcal{S}$ ,  $\alpha$  — ординал,  $\alpha$  интерпретируется как оценка ресурсов для действий, доступных в парах  $w \in \mathcal{Z}$ .

Для предикатов  $p \in P$  зададим в каждом  $z \in \mathcal{S}$  значения истинности. Действия  $f \in \mathcal{F}$  — отношения на  $\mathcal{Z}$ , такие, что если  $(z, \alpha), (t, \beta) \in f$ , то  $\beta < \alpha$ .

Такую интерпретацию могут иметь, например, действия, производимые в реальное время, когда каждое исполнение действия уменьшает оставшееся время. Так же могут быть проинтерпретированы действия синтаксического разбора сложной конструкции, каждое из которых, хотя, может быть, и удлиняет конструкцию, в некотором смысле упрощает ее.

Блок-схемы, построенные из таких необратимых действий и из классических формул над пропозициональными буквами  $p \in P$ , не могут заикливаться. Через конечное число шагов их исполнение заканчивается (нормально либо аварийно ввиду неприменимости очередной функции из-за недостатка ресурсов).

Имеется корректная и полная система правил (логика  $\Omega_1$ ), позволяющих при помощи некоторой блок-схемы построить доказательство возможности преобразовать  $\mathcal{U}$  в  $\mathcal{Z}$  на базе заданных функций, преобразующих некоторые  $\mathcal{U}_i$  в  $\mathcal{Z}_i$ :

$$\frac{\mathcal{U} \Rightarrow \mathcal{Z} \quad \mathcal{S} \Rightarrow \mathcal{D}}{\mathcal{U} \vee \mathcal{S} \Rightarrow \mathcal{Z} \vee \mathcal{D}} \quad - \quad \text{правило условного оператора (ПУО),}$$

$$\frac{\mathcal{U} \supset \mathcal{Z} \quad \mathcal{Z} \Rightarrow \mathcal{S} \quad \mathcal{S} \supset \mathcal{D}}{\mathcal{U} \Rightarrow \mathcal{D}} \quad - \quad \text{правило релаксации (ПР),}$$

$$\frac{u \vee \mathfrak{C} \Rightarrow \mathfrak{Z} \vee \mathfrak{C}}{u \Rightarrow \mathfrak{Z}} \quad - \text{правило заикливания (ПЗ)},$$

$$\frac{u \Rightarrow u}{\neg u} \quad - \text{правило бесконечного цикла (ПБЦ)},$$

$$\frac{\neg u}{u \Rightarrow u} \quad - \text{правило ошибки (ПОШ)}.$$

Все перечисленные правила имеют конструктивное истолкование, которое для ПУО и ПЗ видно из их названия. В ПУО  $u$ ,  $\mathfrak{Z}$  — охраны создаваемого условного оператора. В ПЗ  $\mathfrak{C}$  — инвариант создаваемого цикла,  $\mathfrak{Z}$  — условие его завершения. ПР реализации не преобразует; оно лишь говорит о том, что то же  $f$  можно использовать и в более частной ситуации, чем  $u \Rightarrow \mathfrak{Z}$ . ПБЦ утверждает, что застой не может быть вечен, ПОШ постулирует существование нигде не определенной функции *ошибка*.

Таким образом, конструктивное описание ситуации, где классические средства работают плохо, оказалось очень компактным и эффективным.

Логика  $\Omega_1$  — лишь одна из простейших логик схем программ, успешно используемых в автоматизированном планировании действий.

### Интуиционистская логика

В отношении своего синтаксиса интуиционистская логика оказалась наиболее традиционной из конструктивных логик. Она полностью сохранила язык исчисления предикатов и логические связки классической логики (вложив в них свой, конструктивный, смысл). Она по возможности сохранила и правила классической логики, постольку, поскольку они не влекут закона исключенного третьего (принципа всезнания в конструктивной интерпретации) или закона второго отрицания ( $\neg\neg u \Rightarrow u$ ). Закон двойного отрицания, как



Система естественного вывода интуиционистской логики получается из системы для классической логики заменой правил

$$\frac{u}{ur} \quad \text{и} \quad \frac{ur}{u}$$

$$\frac{u \neg u}{\mathfrak{B}} \quad u \quad \frac{* u}{\vdots} \quad \frac{\vdots}{\mathfrak{F}} \quad \frac{\vdots}{u}$$

После такой модификации происходят следующие изменения в свойствах исчисления по сравнению с классической логикой. Все логические связи становятся независимыми. Классическая логика вкладывается в интуиционистскую с помощью, например, вставки двух отрицаний перед каждой элементарной подформулой и каждой подформулой вида  $\exists x!x$  и  $(\forall \mathfrak{B})$ . Если доказуемо  $\forall \mathfrak{B}$ , то доказуемо либо  $\perp$ , либо  $\mathfrak{B}$ . Если доказуемо  $\exists x!x$ , то доказуемо  $\perp x/t$ . Последние два свойства называются *свойством дизъюнкции* и *свойством существования*. Они выполнены лишь для выводимости без допущений; в частности, они могут быть невыполнены в конкретной аксиоматической теории<sup>17</sup>. Поэтому они могут рассматриваться

92

как необходимый признак конструктивности интуиционистской теории.

Важную роль в конструктивном истолковании интуиционистской теории играют *позитивные подформулы*, которые не входят ни в одну посылку импликации и ни в одну подформулу типа  $\neg U$ . В некотором смысле только позитивные формулы дают материал, из которого строится искомый объект; все остальные подформулы обеспечивают лишь условия для построений. Можно доказать, что если в некотором месте вывода появилась элементарная формула  $P(t)$ , то  $P$  должно позитивно входить в аксиомы либо действующие допущения (за исключением того случая, когда из действующих допущений выводима  $\top$ ).

Задача построения программы выражается в интуиционистской логике с помощью *функциональной формулы*  $\forall x(U_1 \& \dots \& U_n \Rightarrow \exists y(\mathcal{B}_1 \& \dots \& \mathcal{B}_k))$ . Здесь  $x$  — входные переменные создаваемой программы;  $y$  — ее результаты;  $\mathcal{B}_i$  задают связи вход-выход;  $U_j$  амбивалентны: они могут представлять как условия на входы, так и входные значения сложных типов (структуры данных, функции-параметры). Например,  $\forall x(x \geq 0 \Rightarrow \exists y(y \geq 0 \& y^2 = x))$  интерпретируется как задача построения программы вычисления квадратного корня  $x$ ;  $\forall xy(x < y \& \forall z(x \leq z \& z \leq y \Rightarrow \exists uM(z, u)) \Rightarrow \exists v \forall zu(x \leq z \& z \leq y \& M(z, u) \Rightarrow u < v))$  может трактоваться как задача построения оператора, сопоставляющего функции, определенной на  $[x, y]$  и перерабатывающей  $z$  в  $u$ , ее верхнюю границу.

Такая амбивалентность некоторых формул в интуиционистской логике выражает еще одно свойство, неизбежно возникающее в достаточно конструктивных системах (на самом деле уже в логике программ). Доказательство является гораздо более избыточной конструкцией, чем программа, и поэтому отнюдь не все его части имеют соответствия в результирующей

программе. Многие его куски служат не для построения искомого результата, а для обоснования правильности проведенных построений. Этим кускам могут быть формально сопоставлены скрывающиеся за ними объекты и преобразования, но эти бездействующие объекты и функции по меньшей мере бесполезны при исполнении извлекаемой программы, а порой просто мешают ее исполнить. Зато при модификации программы бездействующие объекты часто выходят на первый план.

**Пример 4.** Условия корректности применения циклической операции  $\emptyset$  к  $x$  может быть следующим:

$\forall y (\text{преобразуется}(x, y) \Rightarrow y < x) \& x < x_0 \& \text{ограничено}(x)$ .

Весь вывод этого условия, включая построение  $x_0$ , будет бездействующим: в программе нужно лишь само  $x$  и  $\emptyset(x)$ . При построении же программы грубой прикидки времени исполнения синтезированного алгоритма, наоборот, нужно  $x_0$ , а не  $x$ .

Из-за отсутствия принципа всезнания выразительные возможности интуиционистской логики существенно повышаются по сравнению с классической. Семантика реализуемости для интуиционистской логики требует привлечения функционалов всех конечных типов, так что с помощью интуиционистских формул первого порядка мы можем неявно задавать утверждения высших порядков (о функциях и функционалах). Далее, пользуясь интуиционистской теорией, можно постулировать не только знание, но порою и незнание. Например, то, что значение действительного числа не может быть задано точно, постулируется в форме

$$\neg \forall xy (x > y \vee y > x \vee x = y).$$

Повышение выразительной силы и "аккуратности" выводов в интуиционистской логике по сравнению с классической имеет и обратную сторону: поиск вывода в ней существенно сложнее. В частности, метод

резолуций не может применяться потому, что формулы не могут быть разложены на дизъюнкты.

Интуиционистская логика хорошо подходит для описания задач, в которых требуется вычислить новые величины по заданным величинам и вспомогательным функциям и ни один из вычисленных ресурсов (время, память, надежность) не налагает суровых ограничений.

Для других ситуаций интуиционистскую логику приходится варьировать. Например, если ограничения по памяти существенны, можно использовать логику Ж. И. Жирара, если вычисления не найдены – симметрическую конструктивную логику, развитую, в частности, Н. Заславским, который использовал идеи Р. Нельсона и А.А. Маркова.

## 2.5. Логика как инструмент анализа понятий программирования

Прежде всего, рассмотрим простой пример: недетерминированность в условном операторе и в цикле. Ее можно устранять, потребовав, чтобы все  $A_i$  были несовместимыми (заменяя, например,  $A_i$  на  $A_i \& \neg A_1 \& \neg A_2 \& \dots \& \neg A_{i-1}$ ). Но все известные способы устранения неопределенности приводят здесь к концептуальному ухудшению как программ, так и описывающих их теорий. Ухудшение программ не обязательно означает увеличение времени, требуемого на вычисление, либо памяти для хранения промежуточных результатов. Для компьютера как раз программу лучше сделать детерминированной. Но для человека, ее составляющего, читающего и переделывающего, после внесения не оправданного решениями задач детерминизма становится гораздо труднее понимать текст, отделять существенное от несущественного, что резко повышает шансы на ошибку.

Концептуальные преимущества недетерминированности точнее всего выявляет сравнение логических формализмов, описывающих детерминированные и недетерминированные языки. Дескриптивные описания недетерминированных операторов в несколько раз короче. А в конструктивном случае различие становится просто катастрофическим, и порою нет средств построить полную конструктивную систему для детерминированного случая.

Впервые на силу логики в качестве инструмента анализа понятий программирования обратили внимание в середине 70-х годов в связи с развитием теории верификации программ. Оказалось, что для многих конструкций совместное использование которых не приводит к алгоритмической невычислимости, невозможно построить полную формальную систему, описывающую их взаимодействие.

Например, только при многочисленных безуспешных попытках описания процедур с побочным эффектом и присваиваний над массивами стало ясно, насколько неприятные и замаскированные неожиданности подстерегают нас здесь. В этом случае невозможно проследить, изменялся ли элемент и почему он изменялся.

Конструктивные логики позволили перевести логический анализ на следующий уровень. Они, как можно видеть, порою достигают исключительной выразительности очень простыми средствами. Но еще в начале 70-х годов среди математиков, развивавших конструктивный подход, считалось, что конструктивная теория всегда значительно сложнее соответствующей классической. Так действительно происходит, если мы не отобрали тщательно понятия и выразительные средства, используемые для их описания. Красота конструктивного описания является аргументом в пользу того, что в наш алгоритмический язык не вклю-

чено ничего лишнего, мешающего развитию его основных понятий.

Неформальный анализ далеко не единственная возможность, предоставляемая конструктивным описанием. В конструктивной математике возникает новый вид формул: те, которые не являются противоречиями, но отвергаются по иным причинам. Например, если в интуиционистской теории выводимо  $\neg \neg U$ , то она перестает быть конструктивной. Эта возможность была использована при логическом анализе понятия "структурного GO TO".

Как известно, еще в начале 70-х годов было выяснено, что использование переходов в алгоритмических языках является одним из факторов, мешающих корректности программ из-за невозможности проследить возникающие побочные логические связи. Знаменитый американский программист Д.Кнут исследовал случаи, когда GO TO можно тем не менее считать структурным средством. Один из важнейших таких случаев - *завершители*, которые позволяют закончить внешний блок, либо внешнюю процедуру блока, либо процедуры.

Например, если мы организовали сложный и разветвленный процесс поиска элемента в запутанной структуре базы данных и нашли его, то нужно бросить всю проделанную промежуточную работу и вернуться в тот блок, из которого поиск был запрошен.

Много исследований было посвящено структурным GO TO, они получили ясное и конструктивное обоснование в виде правила нового типа - *правила приятной неожиданности*:

если имеем последовательность вложенных незавершенных вспомогательных выводов, в первом из которых нужно доказать  $U_1$ , во втором -  $U_2$  и т.д. и в последнем мы доказали вместо  $U_n$   $U_1$ , то можно завершить все эти выводы "одним махом".

Аппарат выводов в форме графов позволил исследовать это правило и показать, что оно допустимо в классической логике и в логике схем программ, а в интуиционистской логике позволяет вывести  $\neg \vee \neg \neg$ . Таким образом, структурные GO TO оказались совместимы с циклами и обычными функциями, но несовместимы со сложными структурами данных и функционалами высших типов.

Конструктивный анализ дает не только отрицательные результаты. Например, рассмотрение логики схем программ позволило установить, что столь тщательно изгоняемые GO TO естественно получаются в том случае, если все рассматриваемые действия можно считать глобальными преобразованиями состояния системы, а операторы структурного программирования здесь лишь затемняют дело.

Циклы оказались хорошо совместимы с массивами и плохо — с рекурсивными структурами данных, а процедуры высших типов — наоборот.

Массивы и сложные структуры данных плохо совместимы с присваиваниями (в данном случае присваивание дается на целый ряд операторов, несущих различный логический смысл).

В общем, можно сделать вывод, что нет средств программирования, хороших либо плохих самих по себе; хороши либо плохи их комбинации. При этом любая логически разумная комбинация оказывается универсальной, она приспособлена лишь для определенного класса задач.

## Глава 3. О ДОКАЗАТЕЛЬСТВАХ ПРАВИЛЬНОСТИ ПРОГРАММ

### 3.1. Зачем нужно доказывать правильность программ

В настоящее время все меньше и меньше остается областей человеческой деятельности, где бы не применялись современные электронно-вычислительные машины – компьютеры. Необычайно быстрыми темпами возрастает их количество, повышается степень надежности, уменьшение размеров сопровождается увеличением быстродействия и растущими объемами памяти ЭВМ. Все эти процессы оказывают глубокое воздействие на жизнь общества. Но использование компьютеров для решения возникающих перед человеком и человечеством проблем, в свою очередь, ставит новые, никогда доселе не обсуждавшиеся вопросы, острота и важность которых не позволяет отложить поиск ответов на долгий срок.

К числу таких вопросов относится вопрос о надежности программного обеспечения ЭВМ. Как известно, универсальные вычислительные машины могут быть запрограммированы для решения самых разнородных задач – в этом заключается одна из основных их особенностей, имеющая огромную практическую ценность. Один и тот же компьютер, в зависимости от того, какая программа находится у него в памяти, способен осуществлять арифметические вычисления, доказывать теоремы и редактировать тексты, управлять ходом эксперимента и создавать проект автомобиля будущего, играть в шахматы и обучать иностранному языку. Однако успешное решение всех этих и многих других задач возможно лишь при том условии, что компьютерные программы не содержат ошибок, которые способны привести к результатам, ничего общего не имеющим с ожиданиями и целями пользователя.



Могут сказать, что требование отсутствия ошибок в программном обеспечении совершенно естественно и не нуждается в обосновании. Но как убедиться в том, что ошибки в самом деле отсутствуют? Вопрос не так прост, как может показаться на первый взгляд. Прежде всего, в практике программирования используется прием *тестирования* программ. Суть его сводится к следующему. Подлежащую проверке программу неоднократно *запускают с теми входными данными, относительно которых результат известен заранее*. Затем *сравнивают полученный машиной результат с ожидаемым*. Если во всех случаях такой проверки налицо совпадение этих результатов, появляется некоторая уверенность в том, что и последующие машинные вычисления не приведут к ошибочному итогу, т.е. что исходная программа работает правильно.

Например, пусть имеется программа поиска оптимального маршрута при поездке в метро, *минимизирующая число промежуточных станций и количество пересадок*<sup>18</sup>. Предположим, при тестировании программа действительно выдает оптимальные маршруты. Однако проверить их все при достаточно сложной схеме путей практически невозможно, поэтому удачное тестирование не дает полной гарантии того, что программа не содержит ошибок.

Вряд ли, однако, выбор неоптимального маршрута поездки в метро способен привести к чему-то иному, чем к мелким неприятностям, связанным с потерями времени и опозданиями. Но представим себе, что программа управляет ходом производственного процесса или зенитным комплексом, движением самолета или космического корабля - в этих и других подоб-

<sup>18</sup> Такие программы существуют. В книге [22], например, на с. 183-218 обсуждается программа выбора оптимального маршрута в Лондонском и Парижском метро, которую с успехом можно применить и для Московского метрополитена.

ных случаях каждая ошибка в программе может обернуться серьезными или даже катастрофическими последствиями. Здесь уже простое тестирование программ не в состоянии служить основой надежности программного обеспечения. Требуются иные процедуры и методы удостоверения правильности программ и отсутствия в них ошибок.

Еще одна сфера применения ЭВМ, где совершенно необходима уверенность в безошибочности программ — доказательство математических теорем при помощи компьютеров. В самом деле, если результатом работы программы считается *доказательство* некоторого математического суждения, малейшая неточность при составлении программы (например, просмотр одного-единственного возможного случая, способного изменить истинное значение доказываемого утверждения) может обесценить всю программу в целом. Более того, высказывается точка зрения, согласно которой вообще нельзя доказывать математические теоремы на компьютерах. Рассмотрим этот аспект проблемы подробнее.

Одной из первых попыток применить компьютер при поиске доказательств математических теорем является программа Логик-Теоретик (ЛТ) известных специалистов по информатике С.Ньюэлла, Э.Саймона, Д.Шоу. С помощью программы ЛТ заново доказано 38 из 52 теорем одного из разделов математической логики — исчисления высказываний, содержащихся в книге Б.Рассела и А.Уайтхеда "*Principia Mathematica*". В дальнейшем на ЭВМ с большим быстродействием удалось вывести все 52 теоремы. Достижения ЛТ неоспоримы, но ведь "передоказаны" уже доказанные теоремы! А существуют ли теоремы, впервые полученные при помощи ЭВМ? Да, существуют, и таких примеров довольно много. Один из самых известных — решение знаменитой проблемы четырех красок (рис.3).

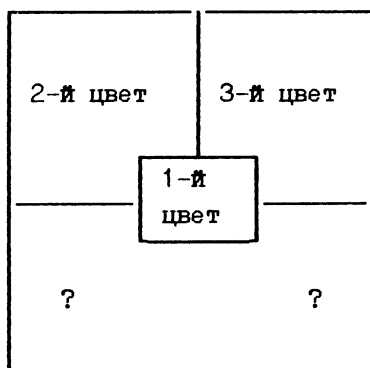


Рис. 3

Немногим более десяти лет назад было опубликовано сообщение о том, что так называемая проблема четырех красок была успешно решена К.Аппелем и В.Хакеном. Проблема заключается в следующем: допускает ли каждая карта такую раскраску, что любая страна на этой карте будет иметь один из четырех цветов и никакие две страны, имеющие общую границу, не будут окрашены одинаково. Поставлена она была в середине прошлого века, когда один из лондонских студентов обнаружил, что для раскраски графств на карте Англии достаточно четырех цветов, и предположил, что данного количества цветов хватит для раскраски любой карты. Несколько десятилетий спустя удалось доказать, что пяти красок во всяком случае достаточно. Но вот можно ли обойтись четырьмя?

Убедиться в том, что задачу о раскраске карты нельзя решить, пользуясь лишь тремя цветами, нетрудно.

Несмотря на простоту формулировки, проблема четырех красок не поддавалась усилиям математиков более 100 лет. Даже после работы Аппеля и Хакена некоторые математики отказывались признать, что проблема действительно решена. Причина возникших

разногласий была связана с существенным использованием ЭВМ в процессе решения, исключавшим возможность проверки вычислений человеком из-за непомерно большого объема работы. Вот что писал по этому поводу советский математик Н.М.Нагорный в письме, опубликованном в книге Л.А.Кудрявцева "Современная математика и ее преподавание" [11, с.103-104]:

"Мне кажется, что протекий и никак не зафиксированный процесс в ЭВМ так же мало годится быть элементом доказательства, как и процессы, протекающие в синхрофазотронах, паровозах или, например, в утюгах. То, что ЭВМ является машиной, используемой математиками, не меняет сути дела. Главная же беда ситуации заключается в том, что этот процесс не только не зафиксирован, но и никак не может быть зафиксирован в принципе. В самом деле, пусть он протекает на какой-либо машине М1. Человек не в состоянии контролировать ее работу, смена ее состояний не может фиксироваться органами его чувств. Это могла бы сделать только какая-нибудь другая машина М2. Но чем М2 лучше М1? Значит, нужна еще и М3, которая будет проверять работу М2. И т.д. Таким образом, налицо регресс в бесконечность."

Впоследствии Д.Коэн предложил другое решение проблемы четырех красок, тем не менее вновь основывающееся на применении компьютера. Хотя объем машинных вычислений, требующих проверки, удалось сократить, трудности, как отмечают в своей книге В.Г.Болтянский и В.А.Ефремович, фактически остались непреодолимыми [3, с.76]:

"Решение проблемы четырех красок, найденное Коэн-ом, изложено в книге среднего объема и формата. По его мнению, проверка этого решения может быть, при желании, выполнена вручную одним человеком в течение двух-трех лет (!) ежедневной восьмичасовой

работы. Скептики считают, что и это решение мало-приемлемо: вряд ли человек, занимавшийся с утра до вечера в течение двух лет нудным перебором вариантов, может гарантировать, что он нигде не допустил ни одной ошибки."

Пожалуй, можно согласиться со "скептиками" в том, что непосредственная проверка человеком протоколов машинных вычислений бесперспективна, поскольку очень часто оказывается за пределами человеческих возможностей. А если проверять не правильность вычислений, проводившихся компьютером при выполнении какой-либо программы, а правильность самой этой программы? В заключительном разделе мы постараемся показать, что вычисления по доказуемо правильным программам могут служить средством получения полноценных математических теорем при помощи компьютеров.

### 3.2. Понятие правильной программы: первое знакомство

Прежде чем говорить о том, каким образом можно доказывать правильность программ, следует, разумеется, определить само понятие правильной программы. С интуитивной точки зрения программа будет правильной, если в результате ее выполнения будет достигнут тот результат, с целью получения которого и была написана программа. Сам по себе факт безаварийного завершения программы еще ни о чем не говорит: вполне возможно, что программа в действительности делает совсем не то, что было задумано и для чего она была написана. Хотя, конечно, правильная программа не должна содержать в себе так называемых синтаксических ошибок.

Ошибки такого рода могут возникать по различным причинам. Например, они могут быть связаны с ошибками при написании команд. Скажем, вместо правиль-

ного PRINT (команда печати в языке Бейсик) от компьютера требуют выполнить оператор PRANT, чего он сделать не может по той простой причине, что такого оператора просто не существует в упомянутом языке программирования. Или предписывается выполнение команды печати PRINT X для переменной X, впервые появившейся в программе. Но в этом случае переменной X еще не присвоено никакого значения и печатать (на экране или на бумаге) нечего, что также приводит к появлению сообщения об ошибке. К числу синтаксических относится и ошибка, связанная со смешением типов переменных. Например, если с переменной, значением которой является строка символов (допустим, какое-то слово русского языка), производится арифметическое действие умножения и т.п. Короче говоря, о синтаксических ошибках мы говорим тогда, когда нарушены правила написания текстов программ на данном языке программирования, поэтому эти ошибки очень похожи на ошибки, возникающие в диктантах, сочинениях или других письменных работах, в которых проверяется знание русского языка. Только в роли учителя выступает компьютер, который в отличие от обычного преподавателя строг настолько, что выставляет оценку "2" при наличии хотя бы одной синтаксической ошибки. Зато компьютер гораздо более терпелив и готов до бесконечности заниматься работой над ошибками - до тех пор, пока их не будет совсем.

В дальнейшем мы будем предполагать, что обсуждаемые программы не содержат синтаксических ошибок, поэтому при обосновании их правильности внимание будет обращать только на содержательную сторону дела, связанную с вопросом о том, достигается ли при помощи данной программы данная конкретная цель. Целью можно считать поиск решения поставленной задачи, а программу рассматривать как

способ ее решения. Программа будет правильной, если она решит сформулированную задачу. Доказательство правильности программы состоит в предъявлении такой цепочки аргументов, которые убедительно свидетельствуют о том, что это действительно так, т.е. что программа на самом деле решает поставленную задачу.

Например, если в одном ящике содержится 500 яблок, а в другом 250 груш и поставлена задача определить, сколько плодов находится в обоих ящиках, то соответствующая программа на Бейсике выглядит как выполнение утверждения присваивания  $x = 500 + 250$ . Чтобы в явной форме получить значение  $x$ , добавим еще одну команду: PRINT  $x$ . В результате после выполнения программы будет напечатано число 750, являющееся решением первой задачи. Если же требуется узнать, во сколько раз яблок больше, чем груш, используем другое присваивание:  $x = 500/250$  (знак "/" обозначает операцию деления) и снова распечатаем результат. Таким образом, программа

```
10.  $x = 500 + 250$ 
```

```
20. PRINT  $x$ 
```

решает первую задачу, а программа

```
10.  $x = 500/250$ 
```

```
20. PRINT  $x$ 
```

- вторую, но не наоборот. Очевидно, доказывать, что это действительно так, совершенно излишне. Факт слишком очевидный. Рассмотрим более сложный пример:

Даны два неотрицательных целых числа  $x$  и  $y$ . Проверить, делится  $x$  на  $y$  или нет.

Если бы были даны два конкретных числа, наши действия по проверке делимости можно было бы свести к обращению ко второй из приведенных выше программ, в которой 500 и 250 заменены на соответствующие новые числа. Взглянув на экран после вы-

полнения программы, мы непосредственно убедились бы в факте делимости или его отсутствии, поскольку целые числа (есть делимость) печатаются на экране дисплея иначе, чем дробные (нет делимости). Однако подобный образ действий вызывает сомнения. Получается, что компьютер решит задачу не полностью, поскольку на заключительном этапе требуется непосредственное участие человека. Кроме того, речь в задаче идет о любых целых неотрицательных числах<sup>19</sup>. Не будем же мы каждый раз смотреть на экран. Вот если бы компьютер самостоятельно определял, будет результат деления целым числом или дробным! Но непосредственного способа сообщить машине наше требование посредством ввода соответствующей команды, вообще говоря, не существует. Мы не можем просто написать в программе (воспользовавшись операторами Бейсика IF...THEN...ELSE..., чтобы напечатать ответ):

```
IF x делится на y THEN PRINT x; "делится на"; y
ELSE PRINT x; "не делится на "; y.
```

Условие " $x$  делится на  $y$ " будет непонятным компьютеру. Значит, нужно найти такое утверждение  $Q$ , которое будет эквивалентно высказыванию " $x$  делится на  $y$ " и истинность или ложность которого компьютер мог бы установить. Запишем эти условия в символической форме:

1.  $Q \leftrightarrow (x \text{ делится на } y)$
2. IF  $Q$  THEN PRINT  $x$ ; " делится на ";  $y$   
ELSE PRINT  $x$ ; " не делится на ";  $y$

Не забудем при этом, что  $x$  и  $y$  - неотрицательные целые числа, т.е. числа, удовлетворяющие усло-

---

<sup>19</sup> Точнее, о любых целых неотрицательных числах, которые могут быть представлены на данном компьютере. Вопросы, связанные с представимостью чисел, обсуждаются ниже.



вию  $(x \in N) \& (y \in N)$ , где  $N = \{0, 1, 2, \dots, n, \dots\}$  - множество всех неотрицательных целых чисел.

Представим себе, что имеется программа  $P$ , в которой выражение 2 используется в качестве последнего оператора и которая, предположительно, решит поставленную задачу. Какой смысл следует вкладывать в утверждение о правильности  $P$ ? Очевидно,  $P$  будет правильной, если всякий раз, когда перед выполнением программы  $P$  истинно условие  $(x \in N) \& (y \in N)$ , после выполнения  $P$  будет истинным утверждение  $Q \leftrightarrow (x \text{ делится на } y)$ . Сумев это доказать, мы тем самым докажем правильность программы.

Сама программа  $P$  пока отсутствует. Известен лишь оператор, который должен выполняться последним. Неизвестен, в частности, конкретный вид утверждения  $Q$ . Имеется много способов построить  $P$ . Первое, что приходит в голову - использовать в  $P$  операцию деления с последующим уточнением, является ли полученный результат целым числом. Однако реальный компьютер - это не абстрактная вычислительная машина математиков. Точность вычислений компьютера ограничена, причем она варьируется в зависимости от типа используемой ЭВМ. Так, числа 4,0000000225174 и 4,0000000000000, безусловно, различны. Первое не является целым числом, а второе целое. Но если точность компьютера ограничена восемью цифрами, разницу между ними он не сможет уловить. Чем ограничения подобного рода могут помешать нам? Вот реальный пример. При выполнении операции деления целого числа 2147483647 на число 2147483646 персональный компьютер "Асонт" выдает в качестве ответа 1. Ясно, что ответ неправилен.

Приведенные соображения отнюдь не доказывают, что операцию деления нельзя использовать при решении нашей задачи. Они свидетельствуют лишь о

том, что вопросом о точности вычислений нельзя пренебрегать. Обратим внимание, что числа  $x$  и  $y$  должны входить в область представимых на данном конкретном компьютере целых чисел. Именно этот смысл имелся в виду, когда в условии задачи говорилось о том, что числа  $x$  и  $y$  даны. Если  $y$  одного компьютера эта область одна, а у другого — другая, на правильность программы  $P$  данное обстоятельство никакого влияния оказывать не должно. Ведь мы надеемся написать программу, которая будет правильной независимо от того, сколько целых чисел представимо. Важно лишь, чтобы  $P$  давала правильный ответ для любых неотрицательных целых чисел из соответствующей области представимых чисел.

Но в условии задачи ничего не говорилось о том, что представим еще и результат деления  $x$  на  $y$ . Зато можно быть уверенным в том, что если представимо целое число  $n \geq 0$ , то представимо и любое из целых чисел  $m$  в интервале  $0 \leq m \leq n$ . Двойственным образом, если  $n \leq 0$  представимо, то любое из целых чисел  $m$  в интервале  $n \leq m \leq 0$  также представимо на данном компьютере. Действительно, что мы могли бы сказать о компьютере, на котором 6 и 10 представимы, а 4 — нет. Чему тогда равнялся бы результат вычитания  $10 - 6$ ? Да и кому была бы нужна такая вычислительная машина?

Более тонкий вопрос — это вопрос о количестве положительных и отрицательных целых чисел. Если на компьютере представимо наибольшее по абсолютной величине целое число  $n$ , то  $-n$ , увы, не обязательно представимо. Дело в том, что если общее количество представимых чисел является четным числом (а это удобно по техническим причинам), количество положительных чисел не может равняться количеству отрицательных — ведь одно из мест требуется отвести под нуль.

Таким образом, отличие абсолютной величины самого большого целого отрицательного числа на 1 от самого большого целого положительного числа можно оправдать. Но вот оправдать разницу, равную 2 и тем более большую, не представляется возможным. Поэтому опять-таки имеется уверенность, что если представимо целое число  $n$ , то представимо и целое число  $\pm(|n| - 1)$ , т.е. число, меньшее  $n$  на 1 по абсолютной величине. В противном случае можно считать, что компьютер либо требует ремонта, либо вообще мало пригоден для использования в вычислениях. Будем предполагать, что компьютеры, на которых мы работаем, устроены разумно.

Вернемся к нашей задаче. Когда целое число  $x$  делится на целое число  $y$ ? Необходимым, хотя и не достаточным условием является требование  $|x| \geq |y|$ , т.е.  $y$  по абсолютной величине не должен превосходить  $x$ . Так как по условию задачи  $x$  и  $y$  неотрицательны, данное выражение упрощается до  $x \geq y$ . Итак, если  $y > x$ , ответ следует немедленно:  $x$  на  $y$  не делится. Но если  $x \geq y$ , вопрос остается открытым. Нужно поискать более подходящее условие, которое было бы не только необходимым, но и достаточным. Такое условие содержится в определении делимости целых чисел. Обозначим множество всех целых чисел через  $Z$ . Тогда определение будет выглядеть следующим образом:

$$(\forall x \in Z) (\forall y \in Z) ((x \text{ делится на } y) \leftrightarrow (\exists z \in Z) (z * y = x)).$$

(Словами: для любых целых  $x$  и  $y$   $x$  делится на  $y$  тогда и только тогда, когда существует целое число  $z$ , дающее при умножении на  $y$  число  $x$ .)

Поскольку  $x \geq 0$  и  $y \geq 0$  по условию задачи, определение можно переписать, ограничив область действия кванторов всеобщности  $(\forall)$  и существования  $(\exists)$  областью натуральных чисел  $N$ :

$$(\forall x \in N)(\forall y \in N)((x \text{ делится на } y) \leftrightarrow (\exists n \in N)(n * y = x)).$$

Замена переменной  $z$  на переменную  $n$  не существенна, так как выражения  $\exists z P(z)$  и  $\exists n P(n)$  с логической точки зрения имеют одинаковый смысл.

Часто кванторы всеобщности, если область их действия известна, вообще не пишут. Поэтому будем использовать запись

$$(x \text{ делится на } y) \leftrightarrow (\exists n \in N)(n * y = x).$$

Однако следует помнить, что речь идет о любых  $x$  и  $y$  из соответствующей области, так что кванторы всеобщности подразумеваются.

К сожалению, утверждение  $(\exists n \in N)(n * y = x)$  снова не годится на роль  $Q$ , несмотря на свой формальный вид, поскольку обычные компьютеры не умеют пользоваться языком, содержащим кванторы.

Попробуем сформулировать утверждение о существовании  $n$  в терминах компьютерных операций. Компьютер способен установить, верно или нет равенство  $n * y = x$ , если вместо  $n$ ,  $y$  и  $x$  стоят конкретные числа. Вспомним, что  $y$  и  $x$  нам даны. Как проверить, существует ли искомое  $n$ ? Ясно, что если существует, то оно обязательно встретится в ряду чисел  $0, 1, 2, \dots$ . Возникает идея организовать цикл проверок на равенство, начиная с  $0$  и так далее, увеличивая с каждым новым циклом проверяемое число на  $1$ . Описанная идея на языке Бейсик воплощается в следующий фрагмент программы (номера строк, как и раньше, условны):

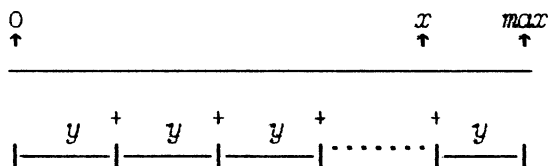
```
10. n = -1
20. REPEAT
30. n = n + 1
40. UNTIL n * y = x
```

В ходе выполнения фрагмента сначала переменной  $n$  будет присвоено значение  $-1$ . Затем при первом прохождении тела цикла (строка 30)  $n$  получит значение

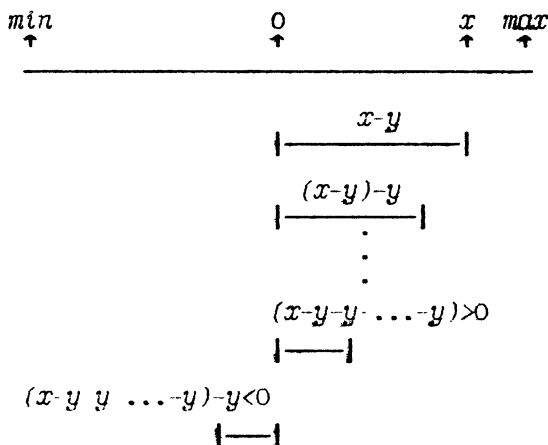
$n = -1+1 = 0$ . Если  $0*y = x$ , условие окончания цикла окажется истинным и работа закончится. Если  $0*y < x$ , цикл повторится,  $n$  получит значение  $n = 0+1 = 1$ , и снова будет проверено условие  $n*y = x$  и т.д. В конце концов либо выполнится  $n*y = x$  при некотором  $n$ , либо ЭВМ не сможет выполнить очередное сложение или умножение из-за непомерно возросшего  $n$  и программа закончит работу аварийно.

Такая программа не может считаться правильной. Вот если бы заранее знать, когда нужно остановиться, чтобы не было аваста (аварийного останова). Однако знать заранее этого нельзя. В самом деле, ясно только, что, если  $n*y > x$ , дальнейшие вычисления теряют смысл, поскольку тогда искомого  $n$  не существует и  $x$  на  $y$  не делится. Но представим себе, что  $x$  близок к наибольшему числу, представляемому на данной машине. В таких обстоятельствах при  $n*y > x$  число  $n*y$  уже может оказаться не представляемым и снова будет авост.

Попробуем подойти к проблеме с другой стороны. Умножение на  $n$  — это все равно, что сложение  $n$  раз. Изобразим возникшее затруднение на схеме:



Необходимое место справа отсутствует. А как обстоят дела слева? Слева располагается ряд отрицательных чисел. На следующей схеме ясно видно, что свободного места там гораздо больше. Может быть, следует вычитать  $y$  из  $x$  до тех пор, пока результат не будет нулевым или отрицательным?



Опираясь на наглядно воплощенную в схеме идею, рассмотрим наихудший из возможных вариантов "глубокого" попадания в область отрицательных чисел в предположении, что  $x \geq y$ . В худшем случае  $y = x-1$  и  $x = \max$ . Не "уйдут" ли теперь результаты вычитания влево за границу  $\min$ ?

Проверим:  $(x-y) = x-(x-1) = 1$ . Поскольку результат все еще положительный, продолжаем  $1-y = 1-(x-1) = -x+2$ . Если теперь  $-x+2 < \min$ , то  $x-2 > |\min|$ . Отсюда наибольшее положительное целое число, представимое в компьютере, отличается от наименьшего целого числа по крайней мере на 3 по абсолютной величине. Вряд ли мы привели аргументы, в силу которых допустимая разница не должна превышать 1. Таким образом, места слева действительно оказалось достаточно.

Посмотрим, каким будет эквивалент утверждения  $(\exists n \in \mathbb{N})(n * y = x)$  в терминах вычитания. Как мы видели,  $(n * y = x)$  истинно тогда и только тогда, когда истинно  $x = (y+y+y+\dots+y)$  ( $n$  раз). Но  $x = (y+y+y+\dots+y)$  ( $n$  раз) эквивалентно утверждению  $0 = (x-y-y-\dots-y)$  ( $n$  раз). Эквивалентность обладает свойством транзи-

$x \in N \text{ \& } y \in N$

(условие, налагаемое на входные данные)

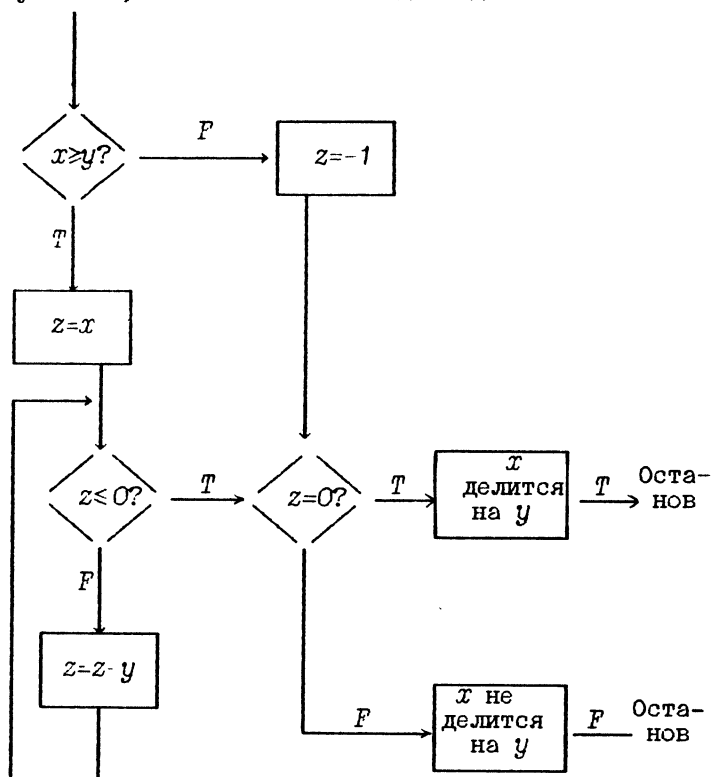


Рис. 4

тивности, т.е. из  $A \leftrightarrow B$  и  $B \leftrightarrow C$  следует  $A \leftrightarrow C$ . Поэтому  $(n * y = x) \leftrightarrow 0 = (x - y - y - \dots - y) (n \text{ раз})$ , откуда  $(\exists n \in N) (n * y = x) \leftrightarrow (\exists n \in N) (0 = (x - y - y - \dots - y) (n \text{ раз}))$ , что и требовалось. Что произойдет, если искомого  $n$  не существует? Было показано, что в таком случае процесс вычитания выйдет в область отрицательных значений, причем авоста из-за "нехватки" чисел не будет.

Блок-схема искомой программы  $P$ , построенная с учетом всех изложенных выше обстоятельств, показана на рис.4. Отметим, что в блок-схеме фигурирует

новая переменная  $z$ , необходимость в которой продиктована требованием напечатать в конце выполнения программы значения  $x$  и  $y$  (либо в варианте "делится", либо в варианте "не делится"). Следовательно,  $x$  и  $y$  меняться не должны.

Выполнение программы начинается с проверки условия  $x \geq y$ . Если оно ложно (стрелка  $F$  - *False*), значит,  $x$  не делится на  $y$  и переменной  $z$  приписывается  $-1$ . Так как в этом случае  $z \neq 0$ , второе проверяемое условие также ложно, поэтому на печать выдается вариант "не делится", после чего программа заканчивает работу (останов).

Если, напротив,  $x \geq y$  истинно (стрелка  $T$  - *True*),  $z$  получает значение  $x$ . Если  $x = 0$ ,  $z$  также равно  $0$ . Тогда условие  $z \leq 0$  истинно и, так как  $z = 0$ , попадаем на вариант "делит" и останавливаемся. Так и должно быть, поскольку  $0$  делится на любое число. Но если  $x \neq 0$ , то  $x > 0$ ,  $z > 0$  и условие  $z \leq 0$  оказывается ложным. В этом случае вычитаем из  $z$   $y$  и снова возвращаемся к условию  $z \leq 0$ . Если вновь  $z > 0$ , повторяем вычитание. Если в конце концов  $z \leq 0$  окажется истинным, перейдем к проверке условия  $z = 0$  и в зависимости от результата проверки получим либо вариант "делит", либо вариант "не делит".

Теперь ясно, что утверждение  $Q$ , поиском которого мы занимались, - это просто утверждение  $z = 0$ . Именно от проверки условия  $z = 0$  зависит ответ на поставленную задачу. Но как убедиться, что блок-схема и соответствующая ей программа  $P$  являются правильными? Существуют разные способы доказательства. Один из них - так называемое доказательство от противного.

Доказательство от противного имеет простую логическую структуру. Допустим, нам надо доказать утверждение  $A$ . Мы принимаем гипотезу о том, что  $A$  ложно, т.е. истинно  $\neg A$ . Если из  $\neg A$  удастся вывести



противоречие  $B \& \neg B$ , заключаем, что  $\neg A$  на самом деле ложно (поскольку влечет противоречие). Значит истинно  $A$ :

Какое утверждение может играть роль  $A$  в нашем случае? Очевидно, в качестве  $A$  нужно взять высказывание ( $z = 0 \leftrightarrow x$  делится на  $y$ ). Допустим, это утверждение неверно. Напомним, что  $B \leftrightarrow C$  эквивалентно  $B \rightarrow C \& C \rightarrow B$ . Отсюда ( $z = 0 \leftrightarrow x$  делится на  $y$ ) эквивалентно конъюнкции ( $z = 0 \rightarrow x$  делится на  $y$ ) & ( $x$  делится на  $y \rightarrow z = 0$ ). По предположению, конъюнкция ложна. Значит, ложен по крайней мере один из ее членов. Допустим, ( $z = 0 \rightarrow x$  делится на  $y$ ) ложно, т.е.  $z = 0$  истинно, но  $x$  на  $y$  не делится. В соответствии с блок-схемой  $z = 0$  либо когда  $x = 0$ , либо когда в процессе вычитания при некотором  $n$  ( $x - y - y - \dots - y$ ) ( $n$  раз) получается 0. В первой ситуации из  $x = 0$  следует ( $x$  делится на  $y$ ) в противоречии с допущением. Во второй в силу указанной выше эквивалентности имеем  $(\exists n \in N)(0 = (x - y - y - \dots - y) (n \text{ раз})) \leftrightarrow \leftrightarrow (\exists n \in N)(n * y = x) \leftrightarrow (x \text{ делится на } y)$ . Так как  $(\exists n \in N)(0 = (x - y - y - \dots - y))$  истинно, ввиду транзитивности эквивалентности оказывается истинным и утверждение ( $x$  делится на  $y$ ), что вновь противоречит допущению. Таким образом,  $z = 0$  влечет ( $x$  делится на  $y$ ).

Допустим теперь, что  $x$  делится на  $y$ , но  $z \neq 0$ . Неравенство  $z \neq 0$  может иметь место либо при невыполнении условия  $x \geq y$ , т.е. при  $x < y$ , либо после завершения цикла, либо в ходе его выполнения. Присваивание  $z = x$  при  $x \neq 0$  приводит к вхождению в цикл, поскольку из  $x \neq 0$  следует  $x > 0$  и  $z > 0$ . Так что никаких других возможностей для  $z \neq 0$ , кроме отмеченных трех, на блок-схеме не существует. Разберем эти три возможности поочередно.

Совершенно очевидно, что  $x < y$  влечет не только  $z \neq 0$ , но и ( $x$  не делится на  $y$ ). Остается рассмот-

реть цикл. Пусть  $z \neq 0$  и цикл завершен. Тогда  $z < 0$ . Снова используем эквивалентность ( $x$  делится на  $y$ )  $\leftrightarrow (\exists n \in \mathbb{N})(0 = x - y - y - \dots - y)(n \text{ раз})$ . Так как процесс вычитания в цикле привел к отрицательному  $z$ , нужного  $n$  не существует. Отсюда  $x$  не делится на  $y$ .

Остается третья и последняя возможность:  $z \neq 0$  в ходе выполнения цикла. Но в этом случае мы и не требуем, чтобы был дан ответ на вопрос о делимости  $x$  на  $y$ . Мы рассчитываем его получить после выполнения программы, а не в ходе выполнения. Но только что было доказано, что, если цикл завершится с  $z \neq 0$ ,  $x$  на  $y$  не делится. Стоп, стоп... Если завершится... Тогда, конечно, все будет правильно. А если программа не завершится нормально из-за авостата? Но вспомним, что переход от сложения к вычитанию и был предпринят для предотвращения авостата. Авостат предыдущими рассуждениями исключается. Однако отсюда еще не следует, что цикл завершится. Вдруг, при  $z \neq 0$  вычисления вновь и вновь будут повторяться в цикле без конца?

Посмотрим на цикл блок-схемы внимательнее. В ходе его выполнения  $z$  должна все время уменьшаться, пока не станет 0 или отрицательным числом. Тогда цикл рано или поздно закончится. Однако, увы! Про случай  $y = 0$  следовало бы вспомнить раньше. Действительно,  $z - 0 = z$  и при  $z > 0$  выполнение цикла не окончится никогда, пока компьютер в исправности и включен.

Но так и должно быть! Ведь мы пытаемся установить, делится  $x$  на  $y$  или нет, а делить отличное от нуля число на нуль запрещено. Хотя в блок-схеме программы операция деления не используется, в ней реализован подход, эквивалентный делению. Поэтому при  $x > 0$  и  $y = 0$  не только  $x/y$  неопределено, но и вычисления в цикле не могут завершиться с определенным результатом, если программа работает прави-

льно. Вместе с тем, программу, способную впасть в бесконечный цикл, правильной считать не хочется.

Немного оправившись от постигшей нас неудачи в доказательстве правильности, подумаем: а так ли уж плохо обстоят дела? Во-первых, мы доказали, что если программа  $P$  закончит работу, задача на делимость будет решена, причем решена правильно. Во-вторых, это доказательство было не лишним уже потому, что в ходе доказательства была выявлена пропущенная ранее возможность, связанная с условием окончания работы цикла. Наконец, в-третьих, легко модифицировать нашу блок-схему таким образом, чтобы окончание выполнения программы было гарантировано, а правильность не нарушена. Например, можно перед проверкой условия  $x \geq y$  проверить утверждение  $x > 0 \ \& \ y = 0$ . Если оно истинно, перейти к присваиванию  $z = -1$ , если же оно ложно – перейти к проверке  $x \geq y$ . Блок-схема просто дополняется еще одним условием, а в остальном остается без изменений.

Тем самым доказательство правильности программы  $P$  (точнее, ее модифицированной блок-схемы; но по блок-схеме нетрудно восстановить всю программу) завершено. Быть может, оно было слишком подробным. Зато можно надеяться, что выработался некоторый опыт и интуиция, которые позволят нам воспринимать более абстрактные идеи.

Теперь мы готовы сформулировать понятие правильной программы в общем виде, применяя единообразный способ записи утверждений о правильности. Пусть  $\alpha$  – программа,  $\Phi$  – утверждение, относящееся к входным данным, которое должно быть истинно перед выполнением программы  $\alpha$ , и  $\mathfrak{K}$  – утверждение, которое должно быть истинно после выполнения программы  $\alpha$ .  $\Phi$  называется *предусловием*, а  $\mathfrak{K}$  – *постусловием* программы  $\alpha$ . Полезно различать два вида правильности: частичную и тотальную (полную). Про-

рамма  $\alpha$  называется *частично правильной* по отношению к  $\mathcal{P}$  и  $\mathcal{K}$ , если всякий раз, когда предусловие  $\mathcal{P}$  истинно перед выполнением  $\alpha$  и  $\alpha$  заканчивает работу, постусловие  $\mathcal{K}$  также будет истинно. В этом случае будем использовать запись  $\mathcal{P}(\alpha)\mathcal{K}$ . Программа  $\alpha$  называется *тотально правильной* по отношению к  $\mathcal{P}$  и  $\mathcal{K}$ , если она частично правильна по отношению к  $\mathcal{P}$  и  $\mathcal{K}$  и обязательно завершает свою работу, если  $\mathcal{P}$  истинно. В этом случае пишем  $\mathcal{P}(\alpha\downarrow)\mathcal{K}$ .

Подчеркнем, что понятие правильности программы  $\alpha$  сформулировано относительно соответствующих утверждений  $\mathcal{P}$  и  $\mathcal{K}$ . Поэтому из истинности утверждения  $\mathcal{P}(\alpha)\mathcal{K}$  (или  $\mathcal{P}(\alpha\downarrow)\mathcal{K}$  соответственно) не обязательно следует истинность утверждения о правильности при других пред- и постусловиях. Аналогичным образом, замена в  $\mathcal{P}(\alpha)\mathcal{K}$  (или  $\mathcal{P}(\alpha\downarrow)\mathcal{K}$ ) программы  $\alpha$  на программу  $\beta$ , вообще говоря, не сохраняет истинностного значения утверждения о правильности. Хотя не следует думать, будто при данных  $\mathcal{P}$  и  $\mathcal{K}$  существует только одна программа  $\alpha$ , для которой высказывание  $\mathcal{P}(\alpha)\mathcal{K}$  (или  $\mathcal{P}(\alpha\downarrow)\mathcal{K}$ ) истинно. На самом деле существуют отличающиеся от  $\alpha$  программы, для которых утверждения о правильности при тех же  $\mathcal{P}$  и  $\mathcal{K}$  имеют то же самое истинностное значение.

Говорить о правильности программы самой по себе бессмысленно (если не иметь в виду отсутствие синтаксических ошибок, авоста или бесконечных циклов). Программы пишутся с целью получить решения задач, а каждая правильно поставленная задача содержит в себе условие (то, что дано) и вопрос, на который нужно дать ответ. Если задача вообще поддается решению с помощью компьютера, ее условие превращается в предусловие, а вопрос преобразуется в постусловие, имеющее уже форму не вопроса, а утверждения, причем это утверждение должно быть истинно всякий раз, когда ответ на вопрос задачи

правилен. Применительно к данным предусловию  $P$  и постусловию  $R$  пишется соответствующая программа  $\alpha$ , правильность которой необходимо доказать. Лучше всего ~~делать~~ это не после написания программы, а в ходе ее составления. Собственно, именно это и было проделано при решении задачи на делимость. Причем сначала была доказана частичная правильность программы  $P$ , а затем, после небольшой модификации  $P$ , и тотальная ее правильность по отношению к данной задаче. Доказательство убедило нас в том, что утверждение

$$(x \in N \ \& \ y \in N) \{P\} (z=0 \leftrightarrow x \text{ делится на } y)$$

истинно при любых  $x$  и  $y$ , удовлетворяющих предусловию (если они вообще представимы на данном компьютере; но это обстоятельство подразумевается). Так как компьютер умеет проверять истинность утверждения  $z=0$ , тем самым (в силу эквивалентности, содержащейся в постусловии) дается ответ на вопрос о делимости  $x$  на  $y$ .

Скажем несколько слов о частичной и тотальной правильности. Из определений следует, что всякая тотально правильная программа является частично правильной при тех же пред- и постусловиях. Обратное, конечно, не верно (мы уже имели возможность убедиться в этом). По-видимому, все согласятся с тем, что тотальная правильность "лучше" частичной, хотя осуществить доказательство тотальной правильности сложнее (это понятно: ведь любое доказательство тотальной правильности содержит в себе доказательство частичной правильности и, кроме того, доказательство отсутствия бесконечных циклов).

Иногда утверждения о частичной правильности программ могут выглядеть довольно странно. Рассмотрим, например, следующий фрагмент программы (обозначим его буквой  $\gamma$ ):

10. REPEAT

20.  $x = x + 0$

30. UNTIL  $x > 0$

Утверждение  $(x=0)\{\gamma\}(x=0 \ \& \ x \neq 0)$  является истинным! Действительно, импликация "если  $x=0$  и  $\gamma$  закончит работу, то  $x=0 \ \& \ x \neq 0$ " является истинной, поскольку по правилам логики из ложной посылки следует все, что угодно. Но посылка " $x=0$  и  $\gamma$  закончит работу" всегда ложна, так как конъюнкция ложна, если ложен хотя бы один из ее членов. В рассматриваемом примере утверждения " $x=0$ " и " $\gamma$  закончит работу" исключают друг друга, т.е. не могут быть вместе истинными. Следовательно, их конъюнкция является ложной и влечет любое утверждение, в том числе заключение " $x=0 \ \& \ x \neq 0$ ", которое, конечно, противоречиво, а потому ложно. Однако вся импликация будет истинна, откуда утверждение  $(x=0)\{\gamma\}(x=0 \ \& \ x \neq 0)$  также истинно, что по определению означает частичную правильность программы  $\gamma$ , хотя в интуитивном смысле программу  $\gamma$  относительно данных пред- и постусловий трудно считать правильной. Вместе с тем утверждение

$$(x=0)\{\gamma\}(x=0 \ \& \ x \neq 0)$$

ложно, поэтому понятие тотальной правильности в большей степени соответствует интуиции.

### 3.3. Доказательство правильности программ: метод математической индукции

Правильность программы  $P$  была доказана методом рассуждений от противного. Другим способом доказательства правильности программ является метод математической индукции. Этот метод удобно проиллюстрировать на арифметических примерах, поскольку в арифметике доказательства по индукции применяются

очень часто и большинство читателей, по-видимому, с ними встречались. Напомним суть дела.

Пусть требуется показать, что все неотрицательные числа обладают свойством  $P$ . Вначале доказываем, что  $P(0)$ , т.е. что 0 обладает свойством  $P$ . Затем доказываем, что всякий раз, когда  $P(n)$  истинно,  $P(n+1)$  также истинно. После этого заключаем, что каждое неотрицательное целое число обладает свойством  $P$ . В символической форме последовательность действий можно записать следующим образом.

1.  $P(0)$
2.  $(\forall n \in N)(P(n) \rightarrow P(n+1))$
3.  $(\forall n \in N)P(n)$

Как убедиться в том, что заключение 3 следует из посылок 1 и 2? Рассуждаем от противного<sup>20</sup>. Допустим, посылки 1 и 2 верны, а заключение 3 ложно. Из  $\neg(\forall n \in N)P(n)$  следует  $(\exists n \in N) \neg P(n)$ , т.е. из отрицания заключения 3 следует утверждение о существовании неотрицательного целого числа, не обладающего свойством  $P$ . Пусть  $m \in N$  — наименьшее такое число. Возможны два случая:  $m=0$  и  $m>0$ . Если  $m=0$ , имеем  $\neg P(0)$  в противоречии с предположением об истинности посылки 1. Если  $m>0$ , оно представимо в виде  $m=n+1$ , где  $n \in N$ . Так как  $m$  — наименьшее неотрицательное целое число, которое не обладает свойством  $P$ , число  $n$  этим свойством обладает, т.е. верно  $P(n)$ . В силу посылки 2 отсюда следует истинность  $P(n+1)$ . Но  $n+1 = m$ , откуда истинно  $P(m)$ , в противоречии с допущением  $\neg P(m)$ . Итак, истинность посылок 1 и 2 влечет истинность заключения 3.

---

<sup>20</sup> В том, что в обосновании метода математической индукции применяется доказательство от противного, нет ничего удивительного — метод доказательства от противного обладает более широкой областью применимости, чем метод математической индукции. Но мы не будем здесь углубляться в этот вопрос.

Впрочем, если применять описанный метод, не вникая в складывающуюся ситуацию, можно прийти к неверным выводам. Рассмотрим следующее утверждение о правильности (записанное для удобства в виде столбца):

$(x \in N \ \& \ x - \text{небольшое число})$

$\{x = x+1\}$

$(x \in N \ \& \ x - \text{небольшое число})$

Попытаемся доказать частичную правильность присваивания  $x = x+1$  при данных (совпадающих) пред- и постусловиях. Допустим,  $(x \in N \ \& \ x - \text{небольшое число})$  истинно и выполнено присваивание  $x = x+1$ . Тогда снова  $(x \in N \ \& \ x - \text{небольшое число})$ , поскольку сумма небольшого числа и 1 не может быть большой. Таким образом, присваивание является частично правильным: истинность предусловия после выполнения присваивания влечет истинность постусловия.

Казалось бы, никаких проблем не возникает. Однако стоит добавить к предыдущему рассуждению очевидное утверждение  $(0 \in N \ \& \ 0 - \text{небольшое число})$ , как мы приходим к нелепому выводу, используя метод математической индукции. В самом деле, имеем

1.  $(0 \in N \ \& \ 0 - \text{небольшое число})$ .

2.  $(\forall n \in N)(n \in N \ \& \ n - \text{небольшое число} \rightarrow n+1 \in N \ \& \ n+1 - \text{небольшое число})$

Но из 1 и 2 следует  $(\forall n \in N)(n \in N \ \& \ n - \text{небольшое число})$ , т. е. утверждение о том, что любое неотрицательное целое число является небольшим, что явно неверно. Значит, где-то была допущена ошибка, и утверждение о правильности присваивания  $x = x+1$  не является правильным при данных пред- и постусловиях? Не торопитесь. Затруднение возникло из-за того, что понятие "небольшое число" не было определено логически точно. Сравните: успех с определением делимости при помощи компьютера был достигнут не в последней степени благодаря тому, что



утверждение " $x$  делится на  $y$ " было эквивалентным образом связано с утверждением " $z=0$ ", с которым компьютер мог оперировать самостоятельно. Ничего подобного в случае свойства "быть небольшим числом" проделано не было. Более того, определение этого свойства в арифметике вообще отсутствует. А пока его нет, нельзя доказать ни правильность, ни неправильность присваивания  $x = x+1$  при рассматриваемых пред- и постулатах. Вывод один - нельзя полагаться на интуитивную ясность или самоочевидность каких-либо утверждений, если мы не в состоянии определить точно условия их истинности или ложности.

Разберем теперь пример доказательства правильности программы с помощью математической индукции, где все используемые понятия будут точно определены. Предположим, требуется написать программу, вычисляющую результат возведения  $x$  в степень  $y$  ( $x^y$ ), где  $x$  целое и  $y$  - целое неотрицательное число. При этом нельзя пользоваться самой операцией возведения в степень. Чтобы решить задачу, начнем с точного определения операции возведения в степень. Как известно, возвести  $x$  в степень  $y$  (всегда будем считать, что действуют упомянутые выше ограничения на  $x$  и  $y$ ) означает взять  $x$  сомножителем  $y$  раз:

$$x^y = x * x * x \dots * x \text{ (} y \text{ раз)}.$$

Поскольку остается непонятным, что значит  $x$  взят сомножителем 0 раз, этот случай выделим особо и положим по определению  $x^0 = 1$  для любого  $x \in \mathbb{Z}$ . Дальше никаких неясностей нет:  $x^1 = x$ ,  $x^2 = x * x$ ,  $x^3 = x * x * x$  и т.д. Непосредственно "сказать" машине, что нужно вычислить произведение, в котором  $x$  повторяется  $y$  раз, нельзя. Но той же самой цели можно добиться, соответствующим образом организовав цикл умножений. Особый случай  $y=0$ , как и раньше, выделим отдельно. Для подсчета числа умно-

жений будем использовать вспомогательную переменную  $I$ , а результаты умножения будем записывать в переменную  $z$ . Мы хотим, чтобы в конце цикла было истинным утверждение  $z = x^y$ . Таким образом, налицо пред- и постусловия будущей программы (обозначим ее через  $P_1$ ): утверждение  $x \in Z \ \& \ y \in N$  является предусловием, а утверждение  $z = x^y$  — постусловием программы  $P_1$ .

Ввиду простоты намеченной программы  $P_1$  мы отступим от рекомендации разрабатывать программу параллельно с доказательством ее правильности (как это было сделано в примере с программой  $P$ ), что позволит сэкономить место и время изложения. Не прибегая к помощи блок-схемы, сразу предъявим  $P_1$  целиком в предположении, что значения переменным  $x$  и  $y$  уже присвоены.

```

10 I = 0 : z = 1
20 IF y = 0 THEN GO TO 80
30 IF x = 0 THEN z = 0 : GO TO 80
40 REPEAT
50 I=I+1
60 z = z*x
70 UNTIL I = y
80 END

```

Требуется доказать тотальную правильность программы  $P_1$  при данных пред- и постусловиях, т.е. доказать утверждение  $(x \in Z \ \& \ y \in N) \{P_1\} (z = x^y)$ . Докажем вначале, что при истинном предусловии возникновение в программе  $P_1$  бесконечных циклов исключается. Бесконечные циклы могли возникнуть либо в области действия оператора REPEAT, либо из-за использования операторов безусловного перехода GO TO в том случае, если они заставляют вновь и вновь, без конца возвращаться к уже выполнявшимся ранее строкам программы. Но выполнение любого из имеющихся в программе  $P_1$  операторов GO TO ведет к окончанию

работы  $P_1$  (будет выполнен оператор END). Остается проверить единственный содержащийся в  $P_1$  цикл.

Окончание цикла (строки 40-70) зависит от выполнения условия  $I = y$ . Из предусловия следует, что целое число  $y \geq 0$ . Если реализуется случай  $y=0$ , выполнение строки 20 программы завершится переходом к строке 80, минуя цикл. При всех остальных значениях  $y$  попадание в цикл возможно. Посмотрим, скажем, что будет при  $y=1$ . Если цикл вообще будет выполняться (что вовсе не обязательно, так как при  $x=0$  произойдет переход к строке 80), при первом попадании в него имеем  $I=1$ . Условие  $I=y$  тем самым будет выполнено и цикл завершен. При  $y>1$  каждое выполнение цикла будет увеличивать целочисленное значение  $I$  на единицу и неминуемо рано или поздно оно сравняется со значением  $y$ , поскольку  $y$  - целое число и его значение остается неизменным. Таким образом, выполнение цикла обязательно завершится при любом входном значении  $y$ , удовлетворяющем предусловию и представимом на данном компьютере.

Попробуем доказать утверждение о частичной правильности  $P_1$ . Прежде всего рассмотрим особый случай  $y=0$ . Выполнение строки 10 закончится присвоением  $z=1$ . Затем в строке 20 будет проверено условие  $y=0$  и, так как оно истинно, произойдет переход к строке 80, завершающей программу  $P_1$ . Но  $z = x^0 = 1$  для любого  $x$ , что и требовалось.

Пусть теперь  $y > 0$ . При  $x = 0$  выполнение строки 30 закончится переходом к окончанию программы и  $z$  будет присвоено значение 0. Но из  $y>0$ ,  $x=0$  следует  $x^y=0$ , откуда  $z=x^y=0^{21}$ . Начиная со следующей строки

<sup>21</sup> На самом деле строка 30 является излишней в том смысле, что программа  $P_1$  останется правильной и после ее исключения. Цель введения этой строки - предотвратить бесплодное вычисление  $0*0*0*...*0$  в цикле, который будет повторяться напрасно много раз при больших  $y$ .

40, мы попадаем в цикл. В нашей программе цикл должен завершиться с  $z = x^y$ , поскольку никаких других команд, которые могли бы изменить значение  $z$  после выполнения команд цикла, в программе  $P_1$  нет (оператор END, понятное дело, не в счет). Это возможно лишь в том случае, если  $I = y$  (условие окончания цикла). Но  $y$  может принимать любые значения из последовательности  $1, 2, 3, \dots, n, \dots$ <sup>22</sup>. Значит, при  $y=1$  цикл закончится в состоянии с  $I=1$ , при  $y=2$  с  $I=2$ , при  $y=3$  с  $I=3$  и т.д. Значения  $z$  соответственно должны быть равны  $x^1, x^2, x^3$  и т.д. Что же получается? Если, например, при  $I=1$  равенство  $z=x^I=x^1$  не будет выполнено, значит, и при  $y=1$  цикл, который, как мы показали, обязательно закончится с  $I=y$ , должен повлечь отсутствие равенства  $z = x^y$  ввиду  $x^y=x^I=x^1$ . То же самое верно для  $I=2, I=3$  и т.д. Иначе говоря, если для некоторого значения  $n$  переменной цикла  $I$  равенство  $z=x^I=x^n$  не выполнено, рассчитывать на выполнение равенства  $z=x$  тоже не приходится, коль скоро  $y=I=n$ . Значит, если программа  $P_1$  правильна, равенство  $z_I=x^I$  должно иметь место при любом полученном в цикле значении  $I$ . (Обозначение  $z_I$  указывает на зависимость значения переменной  $z$  от значения  $I$ : при данном  $x$   $z_0$  есть число, присвоенное переменной  $z$  при  $I=0$ ,  $z_1$  - значение  $z$  при  $I=1$  и т.д. Без индексации в той или иной форме трудно обойтись - возникнут сложности с записью утверждений типа  $z=x^I$  для конкретных  $I$ . Так,  $z = x^1$  и  $z = x^2$  влечет  $x^1 = x^2$ , тогда как  $z_1 = x^1$  и  $z_2 = x^2$  не влечет  $x^1 = x^2$ , что нам и требуется.)

Более того, истинность утверждения  $(\forall I \in N^+)(z_I = x^I)$ , где  $N^+$  - это множество  $N$ , из которого уда-

<sup>22</sup> С учетом конечности множества представимых на ЭВМ целых чисел эта последовательность также будет конечной в каждом конкретном случае.

лен 0, влечет истинность утверждения  $z = x^y$  для любых фиксированных  $x$  и  $y$ . Действительно,

$I$  в программе  $P_1$  последовательно принимает значения  $0, 1, 2, 3, \dots, y$ , увеличиваясь в цикле на 1 до тех пор, пока не будет  $I=y$ . Из  $(\forall I \in N^+)(z_I = x^I)$  следует, что при  $I=y$  верно  $z_I = x^I = z_y = x^y$ . Но  $z_y$  — это и есть то самое  $z$ , значение которого мы ищем и которое должно быть равно  $x^y$ !

Итак, из истинности утверждения  $(\forall I \in N^+)(z_I = x^I)$  следует истинность постусловия  $z = x^y$ . Напротив, если  $(\forall I \in N^+)(z_I = x^I)$  ложно хотя бы для одного  $I$ , постусловие  $z = x^y$  также будет ложно при  $y$ , равном этому самому  $I$ . Таким образом, цикл работает правильно, если утверждение  $z_I = x^I$  истинно при любом  $I \in N$ . Между прочим, отсюда вытекает, что истинностное значение  $z_I = x^I$  не должно меняться в ходе выполнения цикла. Истинное утверждение, не меняющее своего истинностного значения при выполнении цикла, называется инвариантом цикла. Если удастся доказать, что  $z_I = x^I$  действительно инвариант цикла программы  $P_1$ , будет доказана истинность постусловия  $z = x^y$  и завершён разбор случая  $y > 0$ .

Но прежде отметим, что инварианты циклов играют важную роль в доказательствах правильности программ, поэтому рассматриваемый пример отнюдь не является в этом смысле исключением. В инварианте цикла заключена информация о том, на что мы можем рассчитывать после очередного выполнения цикла и после его завершения. Целесообразно поэтому организовывать цикл, заранее зная его инварианты<sup>23</sup>. В примере с программой  $P_1$  предполагалось использовать цикл для вычисления результатов возведения в

<sup>23</sup> В противном случае не совсем понятно, для чего вводится цикл — ведь ясное понимание того, что осуществляется в данном цикле, как раз и связано со знанием инвариантов.

степень. Инвариант  $z_I = x^I$  показывает, что в цикле рассматриваемой программы  $P_1$  делается именно то, что требуется. Осталось убедиться, что перед нами действительно инвариант.

Забыв на время о том, что компьютеру доступны не все числа, доказательство истинности утверждения  $(\forall I \in \mathbb{N}^+)(z_I = x^I)$  осуществим индукцией по  $I$ . Утверждение  $(\forall I \in \mathbb{N}^+)(z_I = x^I)$  осуществим индукцией по числу попаданий в цикл программы  $P_1$ . Отметим, что число попаданий в цикл совпадает со значением переменной  $I$ . Так как  $I > 0$  (из-за  $I \in \mathbb{N}^+$ ), начнем индукцию с первого попадания в цикл, т.е. со случая  $I=1$ .

Могут возразить, что ранее в описании метода математической индукции ее полагалось начинать с 0. Возражение было бы справедливым, если бы индукция велась по множеству  $N = \{0, 1, \dots, n, \dots\}$ . Однако мы имеем дело с множеством  $\mathbb{N}^+ = \{1, 2, \dots, n, \dots\}$ , в котором наименьший элемент не 0, а 1. В рассуждениях по индукции важно не пропустить ни одного случая, поэтому следует начинать с наименьшего элемента. Каков на самом деле этот элемент — уже не столь существенно. Например, если требуется доказать наличие некоторого свойства у целых чисел начиная с 10:  $\{10, 11, 12, \dots, n, \dots\}$ , индукцию также следует начинать с 10.

1. При первом попадании в цикл  $I$  будет присвоено значение 1. Затем выполняется присваивание  $z = 1 * x$ , откуда  $z_1 = x = x^1$  после первого прохождения цикла...

2. Допустим, после  $n$ -го попадания в цикл истинно  $z_n = x^n$ . Необходимо показать, что  $z_{n+1} = x^{n+1}$  после  $n+1$ -го попадания в цикл. При  $n+1$ -м попадании выполнится присваивание  $I = n+1$ . В следующей строке осуществится присваивание  $z = z_n * x$ . Так как это  $n+1$ -е присваивание, имеем  $z_{n+1} = z_n * x$ . Ввиду  $z_n = x^n$

получаем  $z_{n+1} = x^n * x = x^{n+1}$  после  $n+1$ -го выполнения цикла, что и требовалось.

3. Так как пункты 1 и 2 истинны,  $(\forall I \in \mathbb{N}^+)(z_I = x^I)$  тоже истинно.

Обратим внимание, что мы нигде в доказательстве не предполагали, что  $x \neq 0$ . Поэтому строка 30 программы  $P_1$  действительно может быть удалена из нее без ущерба для дела. Завершается ли данным замечанием доказательство утверждения  $(x \in \mathbb{Z} \ \& \ y \in \mathbb{N})(P_1 \downarrow) (z = x^y)$ ? К сожалению, остается одно затруднение, связанное с вопросом о представимости целых чисел на компьютерах.

Мы помним, что программа  $P$  обеспечивала **правильный** ответ при условии представимости исходных данных. Напротив, очевидно, что при достаточно больших представимых  $x$  и  $y$  компьютер не сможет вычислить  $x^y$  по программе  $P_1$  из-за выхода числа  $x^y$  за границы области представимых на этой машине чисел. Представимость  $x$  и  $y$  не *гарантирует* представимости  $x^y$ . Разве можно после этого утверждать, что программа  $P_1$  является **правильной**?

Строго говоря, этого утверждать нельзя. Согласно определению частичной правильности, при истинности  $(x \in \mathbb{Z} \ \& \ y \in \mathbb{N})$  и завершении работы программы  $z = x^y$  также должно быть истинным. Но если результат возведения в степень не представим, выполнение программы  $P_1$  завершится сообщением об ошибке<sup>24</sup> при  $z < x^y$ . Тем более не приходится говорить о **тотальной** правильности  $P_1$ .

Таким образом, между программами  $P$  и  $P_1$  существует важное логическое различие. Программа  $P$  давала ответ на вопрос о наличии или отсутствии *отношения* делимости между числами. Требование, чтобы программа давала **правильный** ответ для любой пары

<sup>24</sup> Что-нибудь вроде "Too big at line 60" - "слишком большое (число) в строке 60".

представимых чисел, в этих условиях было вполне естественным. Программа  $P_1$  давала ответ на вопрос о том, чему равен результат операции возведения одного числа в степень другого. В то время как вопрос о наличии или отсутствии свойства или отношения между объектами не выводит нас за границы рассматриваемой области объектов, вопрос о результате операции может вывести за ее пределы.

Простой пример. Пусть рассматриваемая область представлена двухэлементным множеством  $\{2, 4\}$ . Вопрос о делимости решается исчерпывающим образом: 2 делится на 2, 4 делится на 4, 4 делится на 2, но 2 не делится на 4. Никаких других вопросов о делимости нацело для данной области не существует. Иначе выглядит картина в случае применения операции возведения в степень. Так, число  $2^2$  находится в рассматриваемой области, но  $4^2$  уже нет, хотя и 2, и 4 ей принадлежат.

Для того чтобы для любых двух чисел  $x$  и  $y$  из области  $X$  число  $x^y$  также принадлежало  $X$ ,  $X$  должна быть бесконечной. Но реальные компьютеры являются конечными устройствами. В этих условиях требовать от компьютера представимости  $x^y$ , если  $x$  и  $y$  представимы, неразумно.

Вывод, который следует из сказанного, не утешителен: не существует программы  $\alpha$ , делающей истинным утверждение  $\forall x \forall y ((x \in \mathbb{Z} \ \& \ y \in \mathbb{N}) \rightarrow \alpha)(z = x^y)$ .

Приходится смириться с тем, что доказать истинное утверждение о правильности  $P_1$  не удалось. Более того, это невозможно. В том виде, в каком задача поставлена, она не имеет решения. Однако проведенный анализ не был бесплодным. Во-первых, было установлено, что при некоторых (не всех) входных данных программа  $P_1$  работает правильно. Во-вторых, выявлена причина, по которой не удалось завершить доказательство правильности. А что, если, исполь-



зую полученные результаты, определить область целых чисел, в которой программа  $P_1$  не только частично, но и тотально правильна? Очевидно, эта область определяется теми представимыми  $x$  и  $y$ , для которых число  $x^y$  также представимо. Значит, если  $x^y$  представимо, то выполнение  $P_1$  непременно закончится, и притом в состоянии, обеспечивающем истинность утверждения  $z = x^y$ . Отсюда ясно, что утверждение " $x^y$  представимо" следует ввести в предусловие, если мы хотим все-таки получить истинное утверждение о тотальной правильности  $P_1$ :

$$\forall x \forall y ((x \in \mathbb{Z} \ \& \ y \in \mathbb{N} \ \& \ x^y \text{ представимо}) (P_1 \downarrow) (z = x^y)).$$

В силу приведенных выше рассуждений, истинность данного утверждения *доказана*. В условиях невозможности определить при помощи компьютера операцию возведения в степень для любых  $x \in \mathbb{Z}$  и  $y \in \mathbb{N}$  это не такое уж плохое достижение.

Обсуждая понятия, связанные с доказательством правильности программ, мы совсем не касались вопроса об *эффективности* использования машинных ресурсов — времени и памяти. Это не случайно, поскольку правильность и эффективность — независимые свойства программ. Хотя не совсем понятно, как можно называть эффективной неправильную программу. Но представим себе, что имеющиеся в программе ошибки так редки, что могут оставаться незамеченными годами, не приводя при своем появлении к трудноустраняемым последствиям. В подобных условиях неправильная в строгом смысле, но эффективная программа может оказаться более предпочтительной, чем неэффективная правильная. Напротив, если речь идет о доказательстве математической теоремы с использованием компьютера или если ошибка в программе способна повлечь за собой необратимые последствия (катастрофу, например), тщательное доказательство

правильности становится просто необходимым, о чем мы еще поговорим ниже.

Разумеется, лучше всего, если правильность и эффективность мирно уживаются в одной и той же программе. С этой точки зрения разобранные выше программы далеки от идеала<sup>25</sup>. Но цель рассмотренных здесь простых примеров состояла лишь в демонстрации некоторых ключевых идей, связанных с доказательствами правильности программ<sup>26</sup>.

### 3.4. Доказательства правильности программ и доказательства теорем на компьютерах

Вернемся теперь к вопросу о том, можно ли доказывать математические теоремы на компьютерах. Допустим, мы утверждаем, что  $\exists n(9742656 * n = 1286030592)$ , т.е. что 1286030592 делится на 9742656. Утверждение о делимости можно рассматривать как пример теоремы арифметики, которую надлежит доказать (или опровергнуть).

Очевидно, перед нами очень простая теорема, и ее доказательство не доставит нам никаких особых хлопот. Предложите доказать эту теорему своему товарищу и параллельно запустите программу P, введя в нее соответствующие входные данные. Какому из

---

<sup>25</sup> Так, вряд ли кто-либо будет возводить в степень по программе P1, поскольку практически на всех компьютерах реализована операция возведения в степень, время выполнения которой будет лучше по сравнению с временем выполнения P1. Определение наличия или отсутствия делимости также нетрудно сделать гораздо более эффективным. Например, можно воспользоваться для этого операцией деления умножения и функцией Бейсика INT, вычисляющей целую часть числа. Заинтересованный читатель может попробовать написать соответствующую программу, более эффективную, чем P1, и доказать ее правильность.

<sup>26</sup> Систематическое изложение вопросов, относящихся к доказательствам правильности, можно найти в книгах [1, 8].

результатов, если они не совпадут, вы склонны доверять больше? Учитывая, что правильность программы  $P$  доказана, по-видимому, предпочтение будет отдано компьютеру, а не человеку. Ведь при наличии правильного общего плана проверки делимости у компьютера (программа  $P$ ) и у человека (например, знание алгоритма деления "столбиком") в отдельной арифметической операции ошибку скорее допустит человек, чем компьютер.

Тем не менее, даже работая по правильной программе, компьютер все же способен совершить ошибку в вычислениях? Разумеется, ЭВМ не застрахованы от различного рода "сбоев", порождающих ошибочные результаты. Но, к сожалению, человеку не приходится говорить в этой связи о превосходстве над машиной. Ошибками в математических публикациях никого не удивишь, особенно если речь идет о таких знаменитых проблемах, как проблема четырех красок, "заманчивая простота" которой, по выражению А.А.Зыкова, "явилась причиной того, что за ее решение, как и за доказательство Великой теоремы Ферма, взялись, кроме серьезных математиков, также целые полчища дилетантов" [9, с.453].

Однако и "серьезным" математикам не удалось избежать ошибок. Еще в 1880 г. англичанин Кемпе опубликовал работу, в которой приводилось доказательство гипотезы четырех красок. Лишь десять лет спустя Хивуд обнаружил пробел в доводах Кемпе. Количество примеров промахов и неточностей в математических работах можно было бы без труда умножить. Известная шутка о том, что 99% всех опубликованных математических работ ошибочна, таким образом, отнюдь не является беспочвенной.

Противники использования компьютеров как средства доказательства математических теорем могут возразить, что ошибку, допущенную человеком, спо-

собен обнаружить и устранить другой человек, тогда как незаметные компьютерные сбои человек обнаружить не в состоянии, вынужденный обращаться в этом случае за помощью к другим компьютерам, вновь не дающим гарантий отсутствия ошибок. А разве после того как один математик добросовестно проверил работу другого и не обнаружил в ней ошибок, можно с уверенностью сказать, что ошибок в данной работе нет и быть не может? А что, если проверяющий также допустил "сбой", не позволивший ему найти ошибку? Таким образом, и в этом пункте человек не имеет никаких преимуществ перед искусственными интеллектуальными устройствами - компьютерами.

Кроме того, если компьютером, работающим по программе, правильность которой доказана, все же была допущена ошибка, какие есть основания утверждать, что другие компьютеры, проверяющие работу первого (в том смысле, что они выполняют ту же самую программу), также допустят ошибку? На наш взгляд, верить в то, что допущенная одним компьютером в процессе выполнения правильной программы ошибка не будет обнаружена в ходе проверки другими ЭВМ -- это все равно, что верить в реальность заговора машин.

Напротив, если правильность программы не доказана, ошибочный результат может порождаться самой программой, а не тем, что она выполняется неправильно. Правильно выполненная компьютером неправильная программа, даже успешно прошедшая тестирование, не дает гарантий правильности результата вычислений.

Итак, нет оснований считать теорему, полученную компьютером в ходе выполнения правильной программы, менее надежной, чем теорему, доказанную человеком. В принципиальном отношении между этими способами обоснования разницы нет. Тем самым мы еще

раз убеждаемся в важности процедур и понятий, связанных с доказательствами правильности программ.

### 3.5. Доказательство правильности программ с помощью временной логики

В этом параграфе мы обратимся к верификации компьютерных программ с помощью языковых и дедуктивных средств временной логики. За последнее десятилетие эта ветвь модальной логики получила применение как в области изучения искусственного интеллекта, так и в области разработки программного обеспечения.

Процесс конструирования программы включает по крайней мере три различные ступени:

- 1) точное установление того, что программа должна делать,
- 2) собственно написание программы с учетом того, для чего она предназначается (синтез программы),
- 3) проверка того, что программа действительно выполняет задачи, для решения которых она создавалась (тестирование и верификация).

Последний этап в процессе конструирования программы очень важен. Тестирование представляет собой чисто эмпирическую процедуру, а именно, "прогон" программы с целью убедиться, работает ли она. Однако, поскольку число различных возможных вводов данных в программу может быть в целом неограниченно, а путем тестирования мы сможем проверить лишь конечное их число, то возникает требование более строгого пути проверки программы. Таковым является верификация программы, которая представляет собой теоретическое обоснование правильного вычисления программы и включает логико-математическое доказательство того, что вычислительное поведение программы правильно.

В рамках приложения в области теории построения и проверки правильности программ временная логика может применяться на этапе синтеза и на этапе верификации. На этапе синтеза такое применение состоит в использовании временных спецификаций (т.е. формул языка временной логики, описывающих вычислительное поведение программ) для конструирования заранее правильных программ. На этапе верификации временная логика используется как средство проверки правильности уже построенных программ. На последнем мы и остановимся в настоящем параграфе.

Первое упоминание о том, как временные операторы "всегда" и "иногда" могут быть использованы при верификации программ появилось у Р.Бурсталла [28] и касалось последовательных программ, т.е. программ, представляющих собой один вычислительный процесс. Однако интенсивного развития направление временной верификации программ достигло в отношении параллельных программ [41] - программ, состоящих из нескольких процессов, работающих одновременно и взаимодействующих между собой. И этому есть причины. Для выяснения некоторых из них кратко остановимся на отличиях последовательных программ от параллельных, рассмотрев примеры тех и других.

**Пример 1.** Последовательная программа вычисления биномиального коэффициента  $\binom{n}{k}$  для целых чисел  $n$  и  $k$ , таких, что  $0 \leq k \leq n$ <sup>27</sup>.

$$y_1 = n, y_2 = 0, y_3 = 1.$$

```

WHILE  $y_1 \neq n - k$  DO
     $y_2 := y_2 + 1$ 
     $y_3 := y_3 * y_1$ 

```

---

<sup>27</sup> Вычисление биномиального коэффициента следует формуле  $\binom{n}{k} = \frac{n * (n-1) * \dots * (n-k+1)}{1 * 2 * \dots * k}$ .

```

 $y_1 := y_1 - 1$ 
 $y_3 := y_3 / y_2$   END

```

Эта программа представляет собой вычислительный процесс с циклом WHILE ("пока"), в ходе которого поочередно выполняются команды присваивания программным (локальным) переменным  $y_1$ ,  $y_2$ ,  $y_3$  (т.е. переменным, которые меняют свои значения в течение выполнения программы, в отличие от глобальных переменных - в данном случае  $n$  и  $k$ , - которые не меняют своих значений в ходе выполнения программы). После команды  $y_3 := y_3 / y_2$  происходит тестирование, равно ли значение  $y_1$  результату вычитания  $n-k$ . Если нет, последовательность команд присваивания повторяется до тех пор, пока  $y_1$  не станет равным  $n-k$ , что будет означать, что программа завершает работу с результатом  $y_3 = \binom{n}{k}$ , т.е. значение биномиального коэффициента для данных  $n$  и  $k$  получено. Например, для  $n = 3$  и для  $k = 1$  при первом прохождении цикла имеем:  $y_1 = 2$ ,  $y_2 = 1$ ,  $y_3 = 3$ . Поскольку  $y_1$  еще не равен 1 ( $n-k = 1$ ), то цикл повторяется снова, в результате чего  $y_1 = 1$ ,  $y_2 = 2$ ,  $y_3 = 3$ . Так как в этот раз  $y_1 = n-k$ , то вычисление выходит из цикла, и программа завершается с результатом  $y_3 = \binom{n}{k} = 3$ .

Эту же задачу можно решить и с помощью параллельной программы.

Пример 2.  $y_1 = n$ ,  $y_2 = 0$ ,  $y_3 = 1$ .

COBEGIN

```

WHILE  $y_1 \neq n-k$ 
   $y_3 := y_3 * y_1$ 
   $y_1 := y_1 - 1$ 

```

END

COEND

-  $P_1$  -

```

WHILE  $y_2 \neq k$ 
   $y_2 := y_2 + 1$ 
  WAIT  $y_1 + y_2 \leq n$ 
   $y_3 := y_3 / y_2$ 
END

```

-  $P_2$  -

Как видим, теперь вычисление организовано в двух параллельно работающих процессах (на это указывает оператор SOBEGIN-SOEND), взаимодействующих благодаря общим программным переменным  $y_1$ ,  $y_2$ ,  $y_3$ , распределенным по обоим процессам. Однако, в отличие от предыдущего случая, мы уже не сможем столь однозначно описать ход этой программы, так как здесь может быть не одно описание, а несколько, поскольку машина по-разному может спланировать параллельное выполнение процессов. В случае последовательной программы компьютер для данного ввода будет выполнять всегда одну и ту же последовательность программных команд. При наличии параллельных процессов возможны различные "сценарии" выполнения данной параллельной программы, различные пути исполнения ее компьютером, что связано с возможностью разнообразных комбинаций команд из взаимодействующих параллельных процессов. Поэтому и установить правильность такого рода программ несравненно труднее.

Для последовательных программ критерием правильности является завершение (терминация) с правильным результатом, что находит свое выражение в свойствах частичной и полной корректности (об этих свойствах речь пойдет ниже). Для параллельных программ такого критерия недостаточно, ибо бывает так, что завершения программы и не предполагается<sup>28</sup>, и тогда вопрос о правильной терминации вообще теряет смысл. Да и само понятие терминации в отношении параллельных программ следует уточнить: должны ли мы требовать, чтобы все возможные вычислительные пути данной программы завершались с правильным результатом или достаточно одного.

Можно сказать, что завершающаяся последователь-

---

<sup>28</sup> Примером может служить программа резервирования авиабилетов.



ная программа есть реализация отношений ввода-вывода. При параллельной работе процессов мы не можем установить отношения ввода-вывода, исходя только из отношений ввода-вывода каждого из них. Причина в том, что процессы в ходе вычисления влияют друг на друга. Процесс при наличии параллельного ему может протекать иначе, чем сам по себе. Поэтому надо учитывать и события в ходе выполнения программы (например, синхронизацию процессов).

На основании чего временная логика может быть использована в анализе вычислительного поведения программ? В чем состоит суть такого приложения, и почему именно в отношении параллельных программ оно имеет большую значимость? Каковы перспективы дальнейшего развития этого направления в теории верификации программ? Наше последующее изложение будет построено в соответствии с ответами на эти вопросы.

Чтобы выяснить, на основании чего временная логика может иметь приложение к рассуждениям о программах, рассмотрим прежде всего процессы вычисления последовательной и параллельной программ.

На рис.5 приведенная выше последовательная программа вычисления биномиального коэффициента (Пример 1) представлена направленным графом  $P$  с точками-метками  $l^0, l^1, l^2, l^3, l^{end}$  и множеством исходных условий - исходными значениями программных переменных  $y_1, y_2, y_3$  и предусловием  $0 \leq k \leq n$ . Переходы между инструкциями представляют собой выполнение программных команд<sup>29</sup>.

С каждой меткой мы можем связать определенные

<sup>29</sup> Переход представляет собой выполнение соответствующей команды вида  $C \triangleright \gamma$ , где  $C$  - "охраняющее" условие (может отсутствовать и тогда используем в записи *true*),  $\gamma$  - атомарное программное утверждение типа присваивания (может также отсутствовать - [ ]).

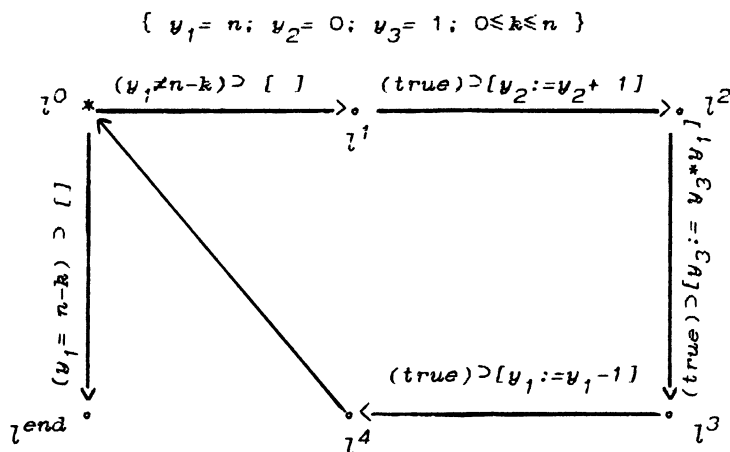


Рис. 5

значения всех программных переменных. Так, если в качестве значений переменных  $n$  и  $k$  положим 3 и 2 соответственно, то с исходной меткой  $l^2$  будут связаны значения 3, 0, 1 программных переменных  $y_1$ ,  $y_2$ ,  $y_3$ . Если мы захотим описать текущее вычислительное состояние программы, то его характеристиками будут, во-первых, метка (контрольный компонент) и, во-вторых, значения программных переменных в текущий момент. Изменение этих характеристик, а следовательно, и вычислительного состояния программы будет происходить в результате осуществления следующего за меткой перехода, предстояющего выполнения программной команды. Тогда в целом вычисление этой программы может быть представлено как последовательность вычислительных состояний, каждое из которых, кроме исходного, получилось из предшествующего в результате осуществления соответствующего перехода:

$$\delta = \alpha_0 = \langle l^0, 3, 0, 1 \rangle; \quad \alpha_1 = \langle l^1, 3, 0, 1 \rangle; \quad \alpha_2 = \langle l^2, 3, 1, 1 \rangle; \\ \alpha_3 = \langle l^3, 3, 2, 3 \rangle; \quad \dots \alpha_{10} = \langle l^0, 1, 2, 3 \rangle; \quad \alpha_{11} = \langle l^{end}, 1, 2, 3 \rangle$$

С выполнением параллельной программы все обсто-

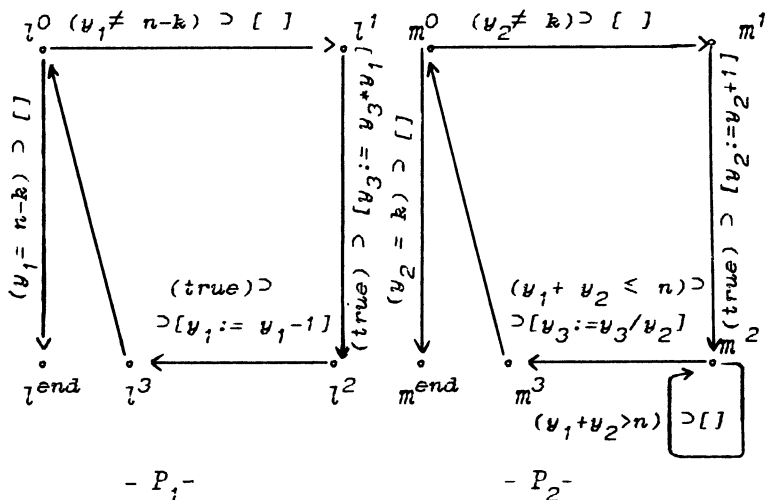


Рис.6

ит сложнее. На рис.6 параллельная программа (Пример 2) представлена двумя направленными графами. Программное состояние в этом случае характеризуется, помимо значений программных переменных, двумя контрольными компонентами - меткой процесса  $P_1$  и меткой процесса  $P_2$ . Исходное состояние  $z_0 = \langle l^0, m^0, 3, 0, 1 \rangle$ . Что же касается вычисления параллельной программы, то при его моделировании принимаются следующие две предпосылки, позволяющие в значительной мере упростить анализ реального параллельного вычисления, которое производится процессорами машины, и в то же время адекватно его представить.

Первая - мультипрограммность, или интерливинг (interleaving - переслаивание, перемешивание). При такой предпосылке вычисление параллельной программы мыслится как последовательность дискретных шагов. На данном шаге выбирается один из параллельных процессов и один из его переходов полностью выполняется без взаимодействия с переходами из других процессов. За один шаг разрешается выполне-

ние только одного перехода одного из параллельных процессов программы. В результате получается некоторая последовательность переходов из разных процессов. И таких последовательностей для одной параллельной программы может оказаться довольно много в связи с тем, что число комбинаций переходов из составляющих процессов может быть велико.

Для того чтобы при представлении реальной параллельности через интерливинг были учтены все возможные ситуации, возникающие при действительном параллельном вычислении, и наша мультипрограммная вычислительная модель была адекватной, необходимо, чтобы "переслаивающиеся" инструкции были максимально атомарны. Поясним почему.

Предположим, что две следующие команды вычисляются в программе параллельно:

$$y = 0$$

$$l: y := y + 1$$

$$m: y := y - 1$$

При допущении интерливинга здесь будет два варианта выполнения:

$$y = 0$$

$$y = 0 + 1 = 1$$

$$y = 1 - 1$$

$$y = 0$$

$$y = 0$$

$$y = 0 - 1 = -1$$

$$y = -1 + 1$$

$$y = 0$$

Оба дают при завершении  $y = 0$ . Однако реальное параллельное выполнение этого фрагмента в качестве возможных результатов даст еще  $y = 1$  и  $y = -1$ , потому что процессоры, осуществляющие эти процессы, одновременно получили доступ к  $y = 0$ . После выполнения первый процессор  $P_1$  получит значение 1, которое он должен положить в ячейку памяти для  $y$ , в то время как второй процессор  $P_2$  получит значение -1, которое также должен будет занести в ячейку памяти для  $y$ . Если  $P_1$  получит свое значение чуть

позже<sup>30</sup>, то мы будем иметь результат  $y=1$ , если чуть раньше —  $y = -1$ .

Причина неудачи нашего мультипрограммного вычисления в том, что данные команды ( $y:=y+1$  и  $y:=y-1$ ) рассматривались нами как атомарные, какими они в действительности не являются. При реальном параллельном выполнении они включают в себя несколько действий: извлечение значения программной переменной из ячейки памяти, вычисление выражения и, наконец, занесение полученного значения обратно в ячейку. Эти события могут перемежаться с аналогичными из параллельного процесса, вызывая взаимодействие. Поэтому, чтобы адекватно отразить параллельное вычисление, протекающее в процессорах, надо раздробить команды  $y:=y+1$  и  $y:=y-1$  на максимально атомарные, введя вспомогательные переменные:

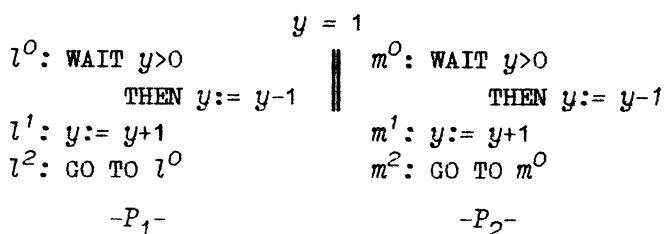
$$\begin{array}{lll} & y = 0 & \\ l : t_1 := y & \parallel & m : t_2 := y \\ l' : y := t_1 + 1 & & m' : y := t_2 - 1 \end{array}$$

В этом случае в качестве возможных результатов получаем  $y = 0$ ,  $y = 1$  и  $y = -1$ , что совпадает с результатами параллельного вычисления, которое бы происходило в компьютере. Таким образом, из сказанного очевидно, почему для представления реальной параллельности через интерливинг "переслаивающиеся" команды должны быть максимально атомарны.

Вторая предпосылка — так называемое справедливое планирование процессов в ходе вычисления (fair scheduling). Суть его состоит в следующем. Если условия всех переходов данного процесса выполняются (т.е. для каждого перехода его охраняющее усло-

<sup>30</sup> Одновременно занести эти значения в ячейку памяти для  $y$  процессоры не смогут, так как в ячейке памяти может храниться только одно значение.

вие при соответствующих значениях программных переменных истинно), то, поскольку планирование процессов ничем не детерминировано, постоянно может выбираться только этот параллельный процесс и только его переходы будут постоянно осуществляться, в то время как другие процессы, равноправные с этим, будут дискриминироваться в отношении выполнения их возможных переходов и "незаконно" забываться. При таком положении дел реальное параллельное вычисление не будет адекватно представлено. Например, в программе, приведенной ниже, в процессе  $P_1$  условия всех переходов выполняются, поэтому возможна такая последовательность:  $l^0 \rightarrow l^1 \rightarrow l^2 \rightarrow l^0 \rightarrow l^1 \rightarrow l^2 \rightarrow l^0 \rightarrow \dots$ . Но она не отражает параллельного



вычисления этой программы. Процесс  $P_2$  вовсе не учитывается в таком "сценарии", хотя его переходы могут выполняться, когда процесс  $P_1$  находится в метке  $l^0$  или  $l^2$ . Примером справедливого планирования процессов для этой программы будет последовательность  $l^0 \rightarrow l^1 \rightarrow l^2 \rightarrow l^0 \rightarrow m^0 \rightarrow m^1 \rightarrow m^2 \rightarrow \dots$ .

В связи с этим при моделировании параллельного вычисления принимается следующая предпосылка справедливого планирования процессов: "Ни один процесс, готовый к работе (т.е. который возможен), не должен оставаться незапланированным" [41].

В соответствии с этими предпосылками мы можем теперь определить выполнение параллельной программы как последовательность программных состояний  $\delta = s_0, s_1, s_2, \dots$ , в которой: а) каждое состояние

характеризуется текущими метками всех параллельных процессов программы и текущими значениями программных переменных; б) каждое последующее состояние получено из предшествующего в результате осуществления соответствующего перехода (представляющего выполнение команды) одного из параллельных процессов; в) эта последовательность — результат справедливого планирования параллельных процессов данной программы.

В нашем примере одна из возможных выполняющих последовательностей следующая:

$s_0 = \langle l^0, m^0, 3, 0, 1 \rangle$ ;  $s_1 = \langle l^1, m^0, 3, 0, 1 \rangle$ ;  $s_2 = \langle l^2, m^0, 3, 0, 3 \rangle$ ;  $s_3 = \langle l^2, m^1, 3, 0, 3 \rangle$ ;  $s_4 = \langle l^2, m^2, 3, 1, 3 \rangle$ ; и т.д.

Таким образом, представляя вычисление программы (и последовательной, и параллельной) как выполняющую последовательность программных состояний, мы получаем основание для применения временной логики, изначально приспособленной к рассуждениям о последовательностях, к анализу вычислительного поведения программ.

В чем состоит суть приложения временной логики? Во-первых, с помощью языка временной логики, специализированного определенным образом, выражаются свойства программ, характеризующие их правильное вычислительное поведение. Во-вторых, для доказательства того, что данная программа обладает интересующим нас свойством, используется дедуктивный аппарат временной логики.

Обычно для описания свойств программ используется первопорядковый язык с временными операторами будущего времени "всегда будет" ( $G$ ), "когда-нибудь будет" ( $F$ ), "в следующий момент будет" ( $X$ ) при условии некоторых дополнений, а именно, прибавления к нему множества пропозициональных переменных вида  $atl$ , которые пробегают по выражениям "про-

цесс  $P_i$  находится в метке  $l^o$ . Это дает нам средства для описания контрольного компонента программных состояний. Кроме того, индивидуальные переменные первого порядка языка подразделяются на глобальные переменные, значения которых не меняются от состояния к состоянию, и локальные переменные, значения которых от состояния к состоянию меняются. Имея в своем распоряжении такой специализированный язык, мы можем описывать свойства, наличие которых у программы определяют ее правильность. Остановимся кратко на этих свойствах и их выражении в языке временной логики.

Выделяют два вида программных свойств: инвариантные и эвентуальные.

Инвариантными называются свойства, которые имеют место постоянно на протяжении каждого вычисления программы. В общем виде они выразимы временными формулами вида  $G\beta$  ("свойство  $\beta$  будет иметь место постоянно на протяжении всего вычисления программы"). К инвариантным относятся свойства частичной корректности, взаимного исключения критических секций, отсутствия тупиковой ситуации, глобальные инварианты и др.

*Частичная корректность.* Пусть  $\phi$  — предусловие, т.е. некоторое ограничение, накладываемое на множество вводов, для которых данная программа предполагается правильной, а  $\psi$  — постусловие, выражающее тот факт, что программа выполняет задачу, ради решения которой создавалась.

Пусть  $l_1^o, \dots, l_n^o$  — исходные метки процессов параллельной программы  $P = P_1 \parallel \dots \parallel P_n$ ,  $l_1^e, \dots, l_n^e$  — конечные метки этих же процессов. Тогда частичная корректность данной программы относительно  $\phi$  и  $\psi$  может быть выражена следующей формулой:

$$(at l_1^o \& \dots \& at l_n^o \& \phi) \supset G (at l_1^e \& \dots \& at l_n^e \supset \psi).$$



Если программа, начавшаяся при соответствующих исходных условиях, выполняющих предусловие, завершается, то она завершается с правильным результатом, т.е. постусловие выполняется.

Свойство частичной корректности значимо и для последовательных, и для параллельных завершающихся программ<sup>31</sup>.

*Взаимное исключение критических секций.* В некоторых параллельных программах приходится сталкиваться с такой ситуацией, когда параллельные процессы включают в себя блоки (секции), содержащие команды, критические для кооперирования этих процессов, поскольку они не могут быть выполнены параллельно без ущерба правильности программы в целом. Поэтому если один из таких процессов находится в своей критической секции, то параллельные ему процессы не должны в это же время находиться в своих критических секциях. Чтобы этого не случилось, в параллельном программировании используется механизм защиты в виде так называемых семафорных инструкций.

**Пример 3.** Рассмотрим параллельную программу, содержащую критические секции (КС). Семафорной инструкцией здесь является инструкция `WAIT...THEN` (ждать..., затем...) с семафорной переменной  $y$ :

$$y_4 = 1.$$

$  \begin{array}{l}  l^0: \text{WAIT } y_4 > 0 \\  \quad \text{THEN } y_4 := y_4 - 1 \\  \hline  l^1: t_1 := y_3 * y_1 \\  \text{KC1 } l^2: y_3 := t_1 \\  \quad l^3: y_4 := y_4 + 1 \\  \hline  l^4: \text{END}  \end{array}  $	$\parallel$	$  \begin{array}{l}  m^0: \text{WAIT } y_4 > 0 \\  \quad \text{THEN } y_4 := y_4 - 1 \\  \hline  m^1: t_2 := y_3 / y_2 \\  \text{KC2 } m^2: y_2 := t_2 \\  \quad m^3: y_4 := y_4 + 1 \\  \hline  m^4: \text{END}  \end{array}  $
$-P_1-$		$-P_2-$

<sup>31</sup> При  $n=1$  указанная временная формула будет выражением частичной корректности для последовательных программ.

Допустим, что процесс  $P_1$  выполнил эту команду и значение  $y_4$  уменьшилось до 0. Теперь компьютеру ничего не остается делать, кроме как выполнять последовательно команды под метками  $l^1 - l^3$ , так как начать вычисление процесса  $P_2$  невозможно: преградой будет семафорная инструкция под меткой  $m^0$ , для выполнения которой значение  $y_4$  должно стать больше 0. А это произойдет только после вычисления команды под меткой  $l^3$ . Следовательно, пока процесс  $P_1$  находится в своей критической секции, процесс  $P_2$  не сможет достичь своей критической секции. Аналогичная ситуация возникнет, если мы начнем с выполнения процесса  $P_2$ .

Свойство, констатирующее, что процессы никогда не выполняют одновременно своих критических секций, называют взаимным исключением критических секций и выражают временной формулой  $\theta \supset G\gamma(atC_1 \& \dots \& atC_n)$ , где  $C_i$  - критическая секция  $i$ -го параллельного процесса;  $\theta$  - формула, выражающая исходные условия. (Напомним, что исходные условия включают в себя значения программных переменных, предусловие (если оно есть) и исходные метки всех параллельных процессов программы.) В приведенном выше фрагменте параллельной программы свойство взаимного исключения таково:

$(atl^0 \& atm^0 \& y_4=1) \supset G\gamma(atC_1 \& atC_2)$ ,  
 где  $C_1 \equiv atl^1 \vee atl^2 \vee atl^3$ ,  $C_2 \equiv atm^1 \vee atm^2 \vee atm^3$   
 $\equiv$  - знак графического равенства<sup>32</sup>.

*Исключение тупиковой ситуации.* Программа оказывается в тупиковой ситуации, если ни один из ее процессов не может работать дальше в связи с тем, что охраняющие условия очередных переходов (обоз-

<sup>32</sup> Свойства взаимного исключения критических секций и исключения тупиковой ситуации характерны только для параллельных программ.

начим их  $E_{l_1}, \dots, E_{l_n}$ ) всех параллельных процессов оказываются ложными при текущих значениях программных переменных. (Терминация, т.е. завершение программы, не рассматривается как "тупик".) Для общей правильности параллельной программы необходимо отсутствие подобной ситуации, что выражается формулой  $\theta \supset G(atl_1 \& \dots \& atl_n \supset E_{l_1} \vee \dots \vee E_{l_n})$  - "всякий раз, когда все процессы параллельной программы, начавшейся при соответствующих исходных условиях, находятся в метках  $l_1, \dots, l_n$ , не являющихся конечными, по крайней мере один из этих процессов возможен, т.е. охраняющее условие его очередного перехода при текущих значениях программных переменных истинно". Для параллельной программы (Пример 3) это свойство выразимо следующей формулой:

$$(atl^0 \& atm^0 \& y_4 = 1) \supset G[(atl^0 \& atm^4 \supset y_4 > 0) \& \\ \& (atm^0 \& atl^4 \supset y_4 = 0) \& (atl^0 \& atm^0 \supset y_4 = 0)].$$

Эвентуальными называются свойства, которые в течение выполнения программы когда-либо будут иметь место. К ним относятся свойства полной корректности, достижимости критических секций, эвентуальности и др. В общем виде эвентуальные свойства выражаются временными формулами вида  $F\beta$  или  $\theta \supset F\beta$ .

*Полная корректность*<sup>33</sup>. Программа полностью корректна относительно предусловия  $\theta$  и постусловия  $\Psi$ , если для каждого ввода завершение вычисления гарантировано и значения программных переменных на выходе выполняют постусловие:

$$\theta \supset F(atl_1^0 \& \dots \& atl_n^0 \& \Psi).$$

<sup>33</sup> Свойство полной корректности значимо как для последовательных, так и для параллельных программ. Свойства достижимости и эвентуальности встречаются только у параллельных программ.

*Достижимость критических секций.* Свойство достижимости критических секций заключается в том, что если параллельный процесс содержит критическую секцию, то он когда-либо войдет в нее и не будет остановлен механизмом защиты в виде семафорных инструкций. Пусть  $l$  – метка непосредственно до критической секции  $C$ . Тогда формула  $atl > F atC$  является выражением свойства достижимости. В частности, для последнего примера параллельной программы с критическими секциями ситуация достижимости параллельными процессами своих критических секций выразится следующей формулой:

$$(atl^0 > F atC_1) \& (atm^0 > F atC_2).$$

*Эвентуальность.* Это свойство отражает тот факт, что в процессе продолжения вычисления параллельной программы мы не должны оказаться заблокированными в метке, которая не является завершающей, даже если остальные процессы продолжают работу:

$$atl > F \neg atl$$

"если процесс достигнет метки, не являющейся завершающей, то выход его из этой метки гарантирован".

Существуют и другие свойства эвентуального и инвариантного типов, но мы, перечислив основные, не будем на них останавливаться и перейдем к нашей главной задаче – установлению того, обладает ли данная программа искомыми свойствами. Продемонстрируем это на конкретном примере, опираясь в своем изложении на работу А.Пнуели [44].

Для доказательства наличия у данной программы свойств, характеризующих ее правильное вычислительное поведение, автор этой статьи предлагает следующую систему аксиом и правил вывода.

Аксиомы:

$$A1. G(A > B) > (GA > GB).$$

$$A2. \chi(A \supset B) \supset (\chi A \supset \chi B).$$

$$A3. GA \supset A.$$

$$A4. GA \supset \chi GA.$$

$$A5. GA \supset \chi A.$$

$$A6. \chi \neg A \equiv \neg \chi A.$$

$$A7. (A \& G(A \supset \chi A)) \supset GA.$$

Правила вывода:

R1) если  $A$  и  $\vdash A \supset B$ , то  $\vdash B$  (*modus ponens*);

R2) если  $\vdash A$ , то  $\vdash GA$ .

Используя аксиомы и правила вывода этой системы, А.Пнуэли выводит так называемое инвариантное правило. Суть его состоит в следующем. Чтобы установить инвариантность (неизменность) некоторого свойства  $A$  в данной программе, надо установить, что: а) оно имеет место в начале программы и б) сохраняется после выполнения каждой команды этой программы:

$$a) \vdash \theta \supset A,$$

$$b) \vdash atl \& A \supset \chi A \quad \text{для всех } l \in \mathcal{L}, \text{ где } \mathcal{L} - \text{множество всех меток программы,}$$

---


$$c) \vdash \theta \supset GA.$$

С помощью этого правила можно доказывать наличие у данной программы разнообразных инвариантных свойств, в частности, свойство исключения критических секций.

Рассмотрим доказательство того, что программа из Примера 3 действительно обладает инвариантным свойством взаимного исключения критических секций:

$$\theta \supset G \neg (atL \& atM),$$

где  $\theta$  — формула, описывающая исходные условия этой программы:  $\theta \equiv atl^0 \& atm^0 \& y_4 = 1$ ,  $atL \equiv atl^1 \vee atl^2 \vee atl^3$ ,  $atM \equiv atm^1 \vee atm^2 \vee atm^3$ .

Сначала с помощью инвариантного правила, приведенного выше, установим вспомогательный инвариант  $G\alpha$ , где  $\alpha \equiv (atL \& \neg atM \& y_4 \neq 1) \vee (\neg atL \& \neg atM \&$

$\& y_4 \neq 1) \vee (\neg atL \& \neg atM \& y=1)$ . Для этого докажем посылки а) и б) правила применительно к рассматриваемой программе.

а)  $\theta \supset \alpha$ .

1.  $atl^0 \& atm^0 \& y = 1$  - исходные условия программы
2.  $atl^0 \& atm^0 \& y_4 = 1 \supset y_4 = 1$  - исчисление высказываний, 1
3.  $atl^0 \supset \neg atL$  - программа Примера 3
4.  $atm^0 \supset \neg atM$  - программа Примера 3
5.  $atl^0 \& atm^0 \& y_4 \supset \neg atL \& \neg atM \& y_4 = 1$  - исчисление высказываний, 2-4
- $\alpha \equiv$  6.  $(\neg atL \& \neg atM \& y_4 = 1) \vee$   
 $\vee (atL \& \neg atM \& y_4 \neq 1) \vee$  - модус роленз,  
 $\vee (atM \& \neg atL \& y_4 \neq 1)$  1,5
7.  $\theta \supset \alpha$  исчисление высказываний, 1,6

б) Для доказательства второй посылки инвариантного правила применительно к данной программе надо проверить, сохраняется ли  $\alpha$  неизменным каждым переходом каждого процесса этой программы. Однако в действительности нас будут интересовать только переходы, которые могут изменить какие-либо из составляющих формулы  $\alpha$ . Такими переходами будут  $l^0 \rightarrow l^1$ ,  $l^3 \rightarrow l^4$ ,  $m^1 \rightarrow m^4$ .

Рассмотрим переход  $l^0 \rightarrow l^1$ . Надо доказать, что  $atl^0 \& \alpha \supset \chi \alpha$ . ( $\alpha \equiv (\neg atL \& \neg atM \& y_4 \neq 1) \vee (\neg atL \& \neg atM \& y_4 \neq 1) \vee (\neg atL \& \neg atM \& y=1)$ ).

1.  $atl^0 \& [(\neg atL \& \neg atM \& y_4 = 1) \vee \vee (atL \& \neg atM \& y_4 \neq 1) \vee \vee (\neg atL \& atM \& y_4 \neq 1)]$  - допущение
2.  $\neg \chi \alpha$  - допущение
3.  $(\neg atL \& \neg atM \& y_4 = 1) \vee$  - удаление  $\&$ , 1

- $\vee(atL \& \neg atM \& y_4 \neq 1) \vee$   
 $\vee(\neg atL \& atM \& y_4 \neq 1)$
4.  $G[(\neg atL \& \neg atM \& y_4 = 1) \vee$  — правило  
 $\vee(atL \& \neg atM \& y_4 \neq 1) \vee$  R2,3  
 $\vee(\neg atL \& atM \& y_4 \neq 1)]$
  5.  $\chi[(\neg atL \& \neg atM \& y_4 = 1) \vee$  — аксиома  
 $\vee(atL \& \neg atM \& y_4 \neq 1) \vee$  A5,4  
 $(\neg atL \& atM \& y_4 \neq 1)]$
  6.  $\chi(\neg atL \& \neg atM \& y_4 = 1) \vee$  — по теореме  
 $\vee \chi(atL \& \neg atM \& y_4 \neq 1) \vee$  из  
 $\vee \chi(\neg atL \& atM \& y_4 \neq 1)$  A1-A7+R1, R2,5
  7.  $\neg \chi(\neg atL \& \neg atM \& y_4 = 1) \&$  — по теореме  
 $\neg \chi(atL \& \neg atM \& y_4 \neq 1) \&$  из  
 $\neg \chi(\neg atL \& atM \& y_4 \neq 1)$  A1-A7+R1, R2,2
  8.  $\neg \chi(\neg atL \& atM \& y_4 \neq 1)$  — удаление  $\&$ ,7
  9.  $\chi(\neg atL \& atM \& y_4 \neq 1)$  — удаление  $\vee$ ,  
6,7
  10.  $\chi \alpha$  — противоречие  
8,9, (-доп.2)
  11.  $atl^0 \& \alpha \supset \chi \alpha$  — введение  $\supset$ ,  
1,10, (-доп.1)

Таким же образом рассматриваются и остальные переходы. Все они оставляют  $\alpha$  неизменным (инвариантным). Посылка  $\beta$ ) доказана.

В соответствии с инвариантным правилом можно заключить, что  $\theta \supset G\alpha$ , т.е.  $(atl^0 \& atm^0 \& y_4 = 1) \supset \supset G[(atL \& \neg atM \& y_4 \neq 1) \vee (\neg atL \& \neg atM \& y_4 \neq 1) \vee \vee(\neg atL \& \neg atM \& y_4 = 1)]$ .

Теперь переходим к доказательству искомой формулы  $\theta \supset G(\neg atL \vee \neg atM)$ :

1.  $\theta \supset G[(atL \& \neg atM \& y_4 \neq 1) \vee (\neg atL \& \neg atM \& y_4 \neq 1) \vee (\neg atL \& \neg atM \& y_4 = 1)]$
2.  $\theta \supset G(\neg atL \vee \neg atM)$  — исчисление высказываний, R2, A1,1

3.  $\theta \supset G\Gamma(atL \ \& \ atM)$  - исчисление высказываний, 2

Итак, доказано, что параллельная программа из Примера 3 обладает свойством взаимного исключения критических секций.

Рассмотрев в общих чертах, в чем суть приложения временной логики, нам легче будет ответить на вопрос, почему использование этой ветви модальной логики имеет наибольшую значимость в отношении параллельных программ.

Для изучения параллельных программ, установления их правильности важно, чтобы была возможность обсуждать не только отношения ввода-вывода, начала и конца вычисления, но и какие-то события в ходе самого выполнения (например, взаимное исключение критических секций). Последнее в свою очередь определяется тем, что вычислительное поведение параллельных программ отличают гораздо большая сложность и разнообразие свойств, являющиеся результатом взаимодействия параллельных процессов в течение вычисления программы. Будучи приспособленной к рассуждениям о последовательностях, временная логика дает такую возможность: с помощью ее языка можно описать различные динамические ситуации, возникающие "внутри" выполняющих последовательностей.

Для изучения разнообразных свойств параллельных программ немаловажен и единый способ их выражения. Именно такой интуитивно ясный способ обеспечивает язык с временными операторами.

Интерливинговая модель вычисления параллельной программы базируется на понятии дискретности, и для ее описания нужен инструмент, также предполагающий дискретность. Временная логика отвечает и этому требованию. Кроме того, она обладает хорошо



разработанным дедуктивным аппаратом, позволяющим доказывать наличие искомым свойств у данной параллельной программы.

В завершение обсуждения этого вопроса приведем мнение А.Пнуэли [44]: "... Нужно ли понятие внешнего времени, или темпоральности, для рассуждений о программах?... для последовательных программ темпоральность не является существенной. Это так потому, что эти программы имеют "внутренние часы", а именно, само выполнение. Зная метку в программе и значения программных переменных, мы можем точно определить, в каком месте выполнения мы находимся. Поэтому для таких программ простые временные понятия как "до" и "после" выполнения программного сегмента ... адекватны. Но при обращении к программам индетерминистическим, параллельным, в которых выполнение состоит из перемешанных между собой операций из различных процессов, мы должны различать "где" и "когда" и сохранять внешнюю временную шкалу, независимую от выполнения".

Каковы дальнейшие перспективы использования временной логики в анализе динамических свойств компьютерных программ и их верификации?

Одним из новых направлений является применение в изучении вычислительного поведения параллельных программ логики ветвящегося времени. (В предшествующем изложении имелась в виду логика линейного времени.) Такое применение возможно, если в качестве вычислительной модели параллельной программы рассматривается не выполняющая последовательность программных состояний, а выполняющее дерево всех возможных выполняющих последовательностей программных состояний. Аргументом в пользу указанного приложения является то, что при таком подходе адекватно отражается индетерминистический характер параллельных вычислений, источник которого - про-

извольный режим планирования очередного параллельного процесса.

Одной из перспектив развития является приложение временных логик (линейных и ветвящихся), язык которых обогащается новыми нестандартными временными операторами, например, *UNTIL* ("вплоть до"), *ATNEXT* (семантика этого бинарного оператора следующая:  $A \text{ ATNEXT } B$  – "A будет иметь место в первом же моменте, в котором имеет место B" и др. Это позволяет описывать некоторые другие свойства параллельных программ, например свойства предшествования (proceeding properties), выражающиеся временными формулами вида  $\chi \text{ U } \gamma$  (событие  $\chi$  имеет место вплоть до наступления события  $\gamma$ ). Эти свойства схожи с эвентуальными. В обоих случаях появление события  $\gamma$  гарантировано, но в случае свойств предшествования гарантировано также, что начиная с настоящего момента и вплоть до момента появления события  $\gamma$  постоянно будет иметь место и событие  $\chi$ .

Идут поиски путей упрощения стадии доказательства, связанных с модуляризацией этого процесса. Вместо того чтобы применять временные рассуждения к программе в целом, их применяют к меньшим программным модулям. Затем рассуждение переносится от отдельных модулей на всю программу.

В настоящем параграфе рассматривался подход, при котором временная логика используется как инструмент верификации уже построенных программ, с помощью которого мы сможем только ответить, правильна или нет данная программа. Развивается направление, связанное с синтезом программ с помощью временных спецификаций. Например, в работах Э.Эмерсона и Д.Халперна исследуется применение временных спецификаций для синтеза синхронизирующей части параллельных программ, отвечающей за правильную координацию выполнения составляющих процессов.

## Глава 4. ПОИСК ДОКАЗАТЕЛЬСТВА

### 4.1. Возможны ли логические процедуры поиска выводов и доказательств?

В гл.1 были описаны методы правильного рассуждения в рамках первопорядковой логики и даны различные способы их формализации. Если нам предъявлен вывод (доказательство), то всегда можно установить, является ли он действительно выводом (доказательством). Но естественно встанет вопрос, как можно построить этот вывод (доказательство). Следует четко различать три вопроса:

1. Что такое правильное рассуждение, вывод, доказательство?

2. Как искать доказательство (вывод)?

3. Как выбрать доказываемое утверждение?

Дедуктивная логика на первых этапах своего развития ограничилась решением первого вопроса. Успехи в решении этой задачи были обусловлены отказом от психологизма, господствовавшего в логике в XIX веке. Логика абстрагируется от того, употреблял ли кто-либо тот или иной способ рассуждения или нет. Важно систематически описать способы рассуждения, гарантирующие при истинности посылки истинность заключения.

Является ли второй вопрос, вопрос о поиске доказательства логическим? Сформулированное в гл.1 аксиоматическое исчисление предикатов не дает нам средств, методов систематического поиска, построения доказательств.

Пусть поставлена задача доказать формулу  $p \supset p$ . Как искать доказательство этой формулы? Задача является творческой и для своего решения требует навыков, определенной изощренности. Попробуем описать процедуру поиска доказательства. Во-первых, отметим, что эта формула содержит только знак имп-

ликация и не содержит других знаков. На этом основании выдвинем гипотезу, что для ее доказательства достаточно использовать первые две схемы аксиом — закон утверждения консеквента:  $A \supset (B \supset A)$  и самодистрибутивность импликации:  $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$ . Заметим, что в общем случае принятая гипотеза не верна; так, для доказательства закона Пирса  $((A \supset B) \supset A) \supset A$  недостаточно двух импликативных аксиом. Не видно, как закон тождества  $p \supset p$  может быть получен из первой аксиомы.

Обратимся ко второй схеме аксиом. Если мы подставим вместо  $A$  и  $C$  переменную  $p$ , то получим

$$(p \supset (B \supset p)) \supset ((p \supset B) \supset (p \supset p)).$$

Если при некотором  $B$  мы сумеем доказать

$$\text{I. } p \supset (B \supset p),$$

$$\text{II. } p \supset B,$$

то, дважды применяя *modus ponens*, получим искомую формулу  $p \supset p$ .

Теперь надо догадаться, что это за формула  $B$ . Мы видим, что формула I, независимо от того, какая формула  $B$ , является аксиомой. Теперь надо найти такое  $B$ , чтобы и формула II была бы доказуемой, в частности, имела бы вид одной из аксиом. Чтобы она была первой аксиомой, достаточно заменить  $B$  формулой  $q \supset p$ . Теперь можно построить искомое доказательство:

- |   |            |
|---|------------|
| 1. $(p \supset ((q \supset p) \supset p)) \supset ((p \supset (q \supset p) \supset (p \supset p))$ | аксиома 2  |
| 2. $p \supset ((q \supset p) \supset p)$  | аксиома 1  |
| 3. $(p \supset (q \supset p)) \supset (p \supset p)$  | т.р.; 1, 2 |
| 4. $p \supset (q \supset p)$  | аксиома 1  |
| 5. $p \supset p$  | т.р.; 3, 4 |

Для доказательства других утверждений потребуются другие эвристические соображения. Накопленный опыт, использование производных правил вывода позволяют осуществлять поиск.

Казалось бы, проблема поиска вывода состоит в

моделировании деятельности человека по построению доказательств. Такой концепции придерживались создатели Общего Решателя Задач - С.Ньюэл, Д.Шоу и Р.Саймон (см. [20]). Однако логические исследования проблемы поиска доказательств преследуют цель изучить не то, как человек реально изобретает доказательство, но то, как возможен поиск доказательств, каковы методы этого поиска, как соотносятся эти методы по их силе и сложности. То же самое можно сказать и о задачах, относящихся к области искусственного интеллекта. В общем перед этой областью исследований стоят проблемы моделирования ряда процедур, носящих интеллектуальный характер, но сами способы их реализации могут существенным образом отличаться от методов, которые использует естественный интеллект. Все это важно, тем более что история науки и методологии свидетельствует: развитие науки и культуры осуществлялось не за счет совершенствования психики и творческих способностей отдельных личностей, а путем изобретения и совершенствования научных методов.

Наиболее естественным для логической постановки задачи поиска доказательств оказалось секвенциальное исчисление. Несмотря на то, что секвенциальное исчисление, предложенное Г.Генценом в 1934 г., не было предназначено для решения проблем поиска доказательств, возможность построения доказательства снизу вверх, т.е. сведение задачи к подзадачам, имплицитно содержится в его работе. Решающий шаг был сделан в 1955 г. Я.Хинтиккой и Э.В.Бетом. Дело не только в чтении фигур заключения снизу вверх, но и во введении нового объекта - дерева поиска доказательств. Фигуры заключения целесообразно переформулировать так, как это было сделано в конце гл.1. Правило введения импликации слева - сформулировать в форме, предложенной А.Кетоненом:

$$\frac{\Gamma \rightarrow \theta, A \quad B, \Gamma \rightarrow \theta}{A \supset B, \Gamma \rightarrow \theta}.$$

Правила введения слабых кванторов – в форме, предложенной С. Кантером:

$$\frac{\Gamma \rightarrow \exists x A x, \theta, A t}{\Gamma \rightarrow \theta, \exists x A x}, \quad \frac{A t, \Gamma, \forall x A x \rightarrow \theta}{\forall x A x, \Gamma \rightarrow \theta},$$

Основными секвенциями будут секвенции вида  $A, \Gamma \rightarrow \theta, A$ . Структурных правил сокращения и добавления посылок нет. Списки формул мы рассматриваем с точностью до перестановок.

Построение дерева поиска начинается с корня. Вопрос о доказуемости нижней секвенции сводится к вопросу о доказуемости верхних секвенций (фигуры заключения обратимы). Дерево поиска доказательства, вообще говоря, бесконечно. Нить дерева поиска доказательства будем называть замкнутой, если в нее входит секвенция вида  $\Gamma, A \rightarrow \theta, A$ . Замкнутую нить мы не продолжаем. Если каждая нить дерева поиска доказательства замкнута, то это дерево поиска доказательства будет доказательством секвенциального исчисления.

Ряд соглашений о порядке применений фигур заключения позволяет для классического исчисления предикатов строить единственное дерево поиска доказательств. Сначала мы применяем правила для логических связок и сильные правила для кванторов и в последнюю очередь слабые правила для кванторов. Общность не нарушится, если при применении слабых правил мы будем использовать только термы, входящие в формулы нижней секвенции (или первую свободную переменную, если ни один терм не входит в формулы нижней секвенции).

Продемонстрируем для примера поиск доказательства секвенции  $\rightarrow \exists x (\exists y A y \supset A x)$ .

Формула нижней секвенции не содержит свободных переменных. Единственная возможность – применить слабое правило для квантора существования:

$$\frac{\rightarrow \exists x(\exists yAy \supset Ax), \exists yAy \supset Aa}{\rightarrow \exists x(\exists yAy \supset Ax)}.$$

Теперь применяем правило для введения импликации справа (в силу сформулированных выше предпочтений):

$$\frac{\frac{\exists yAy \rightarrow \exists x(\exists yAy \supset Ax), Aa}{\rightarrow \exists x(\exists yAy \supset Ax), \exists yAy \supset Aa}}{\rightarrow \exists x(\exists yAy \supset Ax)}$$

Затем применяем сильное правило для введения квантора существования влево:

$$\frac{\frac{Aw \rightarrow \exists x(\exists yAy \supset Ax), Aa}{\exists yAy \rightarrow \exists x(\exists yAy \supset Ax), Aa}}{\rightarrow \exists x(\exists yAy \supset Ax), \exists yAy \supset Aa}}{\rightarrow \exists x(\exists yAy \supset Ax)}$$

На следующем шаге применяем слабое правило для квантора существования. Какой терм мы должны использовать:  $w$  или  $a$ ? Использование  $a$  ничего нового не дает, поэтому используем  $w$ :

$$\frac{Aw \rightarrow \exists x(\exists yAy \supset Ax), \exists yAy \supset Aw, Aa}{\frac{Aw \rightarrow \exists x(\exists yAy \supset Ax), Aa}{\exists yAy \rightarrow \exists x(\exists yAy \supset Ax), Aa}}}{\rightarrow \exists x(\exists yAy \supset Ax), \exists yAy}}{\rightarrow \exists x(\exists yAy \supset Ax)}$$

Заметим, чтобы ликвидировать неопределенность в выборе искомого терма, мы, не нарушая общности, могли бы переформулировать слабые правила для кванторов следующим образом:

$$\frac{\Gamma \rightarrow \theta, \exists x A x, A t_1, \dots, A t_n}{\Gamma \rightarrow \theta, \exists x A x}, \quad \frac{A t_1, \dots, A t_n, \forall x A x, \Gamma \rightarrow \theta}{\forall x A x, \Gamma \rightarrow \theta},$$

где  $t_1, \dots, t_n$  все термы (свободные переменные), входящие в формулы нижней секвенции. (Более эффективно использовать метод временных переменных, предложенный С. Кангером.)

Однако продолжим построение дерева поиска доказательств. Согласно нашим соглашениям, применяем правила для введения импликации справа и получаем

$$\begin{array}{c} \exists y A y, A w \rightarrow \exists x (\exists y A y \supset A x), A w, A a \\ \hline A w \rightarrow \exists x (\exists y A y \supset A x), \exists y A y \supset A w, A a \\ \hline A w \rightarrow \exists x (\exists y A y \supset A x), A a \\ \hline \exists y A y \rightarrow \exists x (\exists y A y \supset A x), A a \\ \hline \rightarrow \exists x (\exists y A y \supset A x), \exists y A y \supset A a \\ \hline \rightarrow \exists x (\exists y A y \supset A x) \end{array}$$

Видно, что самая верхняя секвенция основная. Формула  $A w$  входит в левую и правую часть секвенции. Этот факт отметим, поставив крестик над формулой:

$$\begin{array}{c} \exists y A y, \overset{x}{A w} \rightarrow \exists x (\exists y A y \supset A x), \overset{x}{A w}, A a \\ \hline A w \rightarrow \exists x (\exists y A y \supset A x), \exists y A y \supset A w, A a \end{array}$$

.....

Заметим, что, прежде чем применять какое-либо правило к секвенции, следует проверить, не является ли она основной.

Описанная процедура поиска доказательства является полной, т.е. секвенция доказуема тогда и только тогда, когда соответствующее ей дерево поиска доказательства замкнуто. Хао Ван специально подчеркивает - и, как мы видели выше, он прав, - что имеется логическая теория поиска доказательств - инференциальный анализ [5].

Описанный выше алгоритм поиска доказательств



является разрешающей процедурой для ряда классов формул. Легко видеть, что проблема разрешения для исчисления высказываний разрешима. Каждое правило вывода, примененное снизу вверх, уменьшает длину формулы. Поскольку число логических знаков в формуле конечно, то процесс поиска доказательств оборвется. Если формула доказуема, то незамкнутые нити дают контрпримеры. Проблема разрешения для формул с приставкой  $\forall^n \exists^m$  разрешима. В одноместном исчислении предикатов каждой формуле мы можем сопоставить ей эквивалентную в минисферной форме, такую, что ни один квантор не будет находиться в области действия другого. Отсюда, применяя алгоритм поиска доказательств к формуле в указанном виде, мы завершаем процедуру поиска. Отметим, что Хао Ван широко использует перед поиском процедуру приведения к минисферной нормальной форме, когда кванторы сдвинуты как можно глубже.

Метод поиска доказательства в секвенциальных исчислениях описывается и в других формах. Это прежде всего семантические таблицы Бета. Если семантические таблицы Бета истолковывать синтаксически, то мы имеем просто другую форму записи дерева поиска доказательств, нежели в секвенциальном исчислении. Во-первых, дерево поиска перевернуто. Это перевернутое дерево будем называть таблицей. Во-вторых, вместо знака секвенции мы пишем горизонтальную черту; формулы, стоящие слева от стрелки, записываем столбцом влево от черты, а стоящие справа — столбцом справа от черты. Например, правило введения импликации справа

$$\frac{A, \Gamma \rightarrow \theta, B}{\Gamma \rightarrow \theta, A \supset B}$$

будет соответствовать переход от фигуры  $\Gamma \quad \left| \begin{array}{l} \theta \\ A \supset B \end{array} \right.$

к фигуре  $\Gamma \mid \begin{array}{c} \Theta \\ A \supset B \\ B \end{array}$ .

Двухпосылочным правилам вывода будет соответствовать разбиение таблицы на две подтаблицы. Правилу

$$\frac{\Gamma \rightarrow \Theta, A \quad B, \Gamma \rightarrow \Theta}{A \supset B, \Gamma \rightarrow \Theta}$$

будет соответствовать переход от таблицы  $\Gamma \mid \begin{array}{c} \Theta \\ A \supset B \end{array}$  к

таблице  $\begin{array}{c} \Gamma \\ A \supset B \end{array} \mid \begin{array}{c} \Theta \\ I \quad II \\ | B \quad A \end{array}$ . Таблица замкнута, если каждая под-

таблица замкнута, т.е. в ее левую и правую часть входят одни и те же формулы.

Более наглядной формой, чем семантические таблицы Бета, являются аналитические таблицы. Вместо секвенции  $A_1, \dots, A_n \rightarrow B_1, \dots, B_m$  мы пишем список отмеченных знаками  $T$  и  $F$  формул. Запись  $TA_i$  означает, что формула  $A_i$  находится в левой части секвенции, а  $FB_j$  — что формула  $B_j$  находится в ее правой части. Тогда вместо написанной выше секвенции мы пишем  $TA_1, \dots, TA_n, FB_1, \dots, FB_m$ .

Дерево поиска и сами правила переворачиваем; для импликации мы имеем

$$\frac{F(A \supset B)}{TA, FB}, \quad \frac{T(A \supset B)}{FA \mid TB},$$

Таблица замкнута, если в каждой подтаблице встречается формула, отмеченная знаком  $T$  и  $F$ .

Можно избежать отмеченных формул, рассматривая только секвенции вида  $A_1, \dots, A_n \rightarrow$ . Тогда надо иметь правило для логических знаков и их отрицаний:

$$\frac{\neg(A \supset B)}{A \mid \neg B}, \quad \frac{A \supset B}{\neg A \mid B}.$$

Можно поступить дуально, рассматривая только

секвенции вида  $\rightarrow B_1, \dots, B_m$ . Тогда правила для импликации:

$$\frac{A \supset B}{\neg A} \quad , \quad \frac{\neg(A \supset B)}{A \mid \neg B} .$$

Однако описанный метод поиска доказательства для сложных формул дает чрезмерно большое дерево поиска и требует большой памяти и времени при реализации на компьютерах. Была проделана значительная работа по совершенствованию этого метода. Существенным оказался переход к консервативному расширению исчисления предикатов, содержащему сколемовские функции. В 60-е годы Робинсоном был предложен метод резолюций, а С.Ю.Масловым – обратный метод поиска доказательства. В дальнейшем они были усовершенствованы и реализованы на компьютерах. Мы не имеем возможности изложить их в этой книге и отсылаем читателя к специальной литературе (см., например, [14, 23]).

#### 4.2. Поиск натурального вывода в режиме диалога

В этом параграфе мы хотим предложить процедуру построения натурального вывода, реализуемую на компьютере.

Сначала сформулируем систему натурального вывода. Посылками и заключением натурального вывода являются замкнутые формулы классического исчисления предикатов, язык является расширением стандартного языка исчисления предикатов. Вводятся дополнительные  $\varepsilon$ -термы: если  $A$  – формула, содержащая  $x$  свободно, то  $\varepsilon x A$  есть  $\varepsilon$ -терм. Вместо  $\varepsilon$ -термов мы будем в качестве сокращений использовать термы, образованные с помощью символов вспомогательных функций. Для удобства язык будет содержать констан-

ту  $f$  – ложь. Правилами вывода являются следующие правила введения и удаления логических символов:

$$\supset_y: \frac{A \quad A \supset B}{B}, \quad \supset_y: \frac{\alpha \quad \alpha}{A \quad A} \Rightarrow \frac{\vdots \quad \vdots}{B \quad B} \quad A \supset B$$

$$\&_y: \frac{A \& B}{A}, \quad \frac{A \& B}{B}, \quad \&_B: \frac{A \quad B}{A \& B},$$

$$\neg_y: \frac{\alpha \quad \alpha}{\neg A \Rightarrow \neg A} \Rightarrow \frac{\vdots \quad \vdots}{f \quad f} \quad A, \quad \neg_B: \frac{\alpha \quad \alpha}{A \Rightarrow A} \Rightarrow \frac{\vdots \quad \vdots}{f \quad f} \quad \neg A,$$

$$f_y: \frac{f}{A}, \quad f_B: \frac{A \quad \neg A}{f},$$

$$\vee_y: \frac{\alpha \quad \alpha}{A \quad B} \Rightarrow \frac{\alpha}{A \quad B} \Rightarrow \frac{\vdots \quad \vdots}{C \quad C} \quad C, \quad \vee_B: \frac{A}{A \vee B} ; \frac{B}{A \vee B},$$

$$\exists_y: \frac{\exists x A \quad x}{A(\exists x A \quad x)}, \quad \exists_B: \frac{A \quad t}{\exists x A \quad x},$$

$$\forall_y: \frac{\forall x A \quad x}{A \quad t}, \quad \forall_B: \frac{A(\forall x \neg A \quad x)}{\forall x A \quad x},$$

Вывод является субординатным выводом, как описано в гл.1. Заметим, что описанное исчисление не

эквивалентно стандартному исчислению предикатов, но является консервативным расширением, т.е. из множества формул  $\Gamma$  выводима формула  $A$  в сформулированном исчислении тогда и только тогда, когда из  $\Gamma$  выводима  $A$  в классическом исчислении, при условии, что формулы  $\Gamma$  и формула  $A$  не содержат  $\varepsilon$ -термов.

**З а м е ч а н и е 1.** Часто правило удаления отрицания называют то, что мы называли правилом введения константы  $f$ . Сформулированное же нами правило  $\neg_y$  чаще не формулируют (вместо него принимается правило снятия двойного отрицания  $\frac{\neg\neg A}{A}$ , которое в нашей системе является производным).

**З а м е ч а н и е 2.** Другим отличием нашей системы является то, что имеется не только прямое правило удаления квантора существования, но и нестандартное правило введения квантора общности.

Вернемся к процедуре построения натурального вывода. Первоначально строится некоторая конструкция, которая затем (в благоприятном случае) превращается в натуральный вывод. Отметим, что правила вывода могут применяться двояким образом: аналитическим и синтетическим. При аналитическом применении мы сводим задачу построения вывода к подзадачам. Например, чтобы вывести формулу  $A \& B$ , надо вывести формулу  $A$  и формулу  $B$ ; это аналитическое применение правила введения конъюнкции. С другой стороны, если уже выведены формулы  $A$  и  $B$ , то отсюда выводима и формула  $A \& B$ ; это синтетическое применение правила введения конъюнкции. Некоторые правила имеют только аналитическое или только синтетическое применения.

Исходная задача построения натурального вывода формулируется для замкнутых формул классического исчисления предикатов. Пусть нам требуется из фор-

мул  $A \supset B$  и  $B \supset C$  вывести формулу  $A \supset C$ . Эта задача записывается следующим образом:

1.  $A \supset B$  посылка;
2.  $B \supset C$  посылка;
- 
3.  $A \supset C$ .

Список посылок и искомое заключение отделены пустой строкой (дыркой, символом ○). Задача состоит в построении конструкции, которая не содержит дырки и является натуральным выводом.

Выбор, какое правило применить и к каким формулам, не автоматизирован и осуществляется человеком, строящим натуральный вывод, хотя в принципе может быть сформулирована тактика применения правил вывода. (Например, сначала осуществлять аналитические применения, а затем синтетические.)

Сформулируем аналитическое правило введения импликации (АВИ):

$$\begin{array}{ccc} \circ & \Rightarrow & \left| \begin{array}{c} A \\ \circ \\ B \end{array} \right. , \\ A \supset B & & A \supset B \end{array}$$

т.е. конфигурацию  $\begin{array}{c} \circ \\ A \supset B \end{array}$  мы можем заменить на конфигурацию

$$\left| \begin{array}{c} A \\ \circ \\ B \end{array} \right. \\ A \supset B.$$

Таким образом, первоначальная задача редуцируется к более простой.

Отметим, что если нам нужно не просто построить натуральный вывод, но и натуральный вывод с анализом, то мы формулируем правило АВИ следующим образом:

$$\begin{array}{ccc} \circ & \begin{array}{c} m+1. \left| \begin{array}{c} A \\ \circ \\ B \end{array} \right. \\ n. A \supset B \Rightarrow m+2. \left| \begin{array}{c} A \\ \circ \\ B \end{array} \right. \\ n. A \supset B \end{array} & \begin{array}{l} \text{допущение} \\ \\ \supset_B; m+1 - m+2, \end{array} \end{array}$$

где  $m$  - наибольший номер формулы конструкции, откуда удаляется  $\bigcirc$ . В принципе, может быть  $n.A > B$

построена автоматическая процедура, приводящая нумерацию в естественный порядок.

Для нашего примера будем иметь:

- |    |  |              |
|----|--|--------------|
| 1. | $A > B$  | посылка      |
| 2. | $B > C$  | посылка      |
| 4. | $\left  \begin{array}{c} A \\ \bigcirc \\ C \end{array} \right.$ | допущение    |
| 5. |  |              |
| 3. | $A > C$  | $>_B$ ; 4,5. |

Теперь можно осуществить синтетическое правило удаления импликации (СУИ):

Если формулы вида  $A$  и  $A > B$  входят в конфигурацию, стоящую выше дырки, то мы заменяем  $\bigcirc$  на

$\bigcirc$

$m+1. B >_y; n, k,$

где  $n$  и  $k$  - номера формул  $A$  и  $A > B$ .

Заметим, что когда говорят, что формулы  $A$  и  $A > B$  входят в конфигурацию, то имеют в виду, что они входят в подвывод, в который входит  $\bigcirc$ , или входят в вывод, в который входит наш подвывод.

В нашем случае команда СУИ; 4,5 приводит к конфигурации

- |    |   |              |
|----|---|--------------|
| 1. | $A > B$   | посылка      |
| 2. | $B > C$   | посылка      |
| 4. | $\left  \begin{array}{c} A \\ B \\ \bigcirc \\ C \end{array} \right.$ | допущение    |
| 6. |   | $>_y$ ; 4,1  |
| 5. |   |              |
| 3. | $A$   | $>_B$ ; 4-5. |

Вновь применяем правило удаления импликации к формулам за номерами 6 и 2: СУИ; 6,2 - и получаем

1.	$A \supset B$	посылка
2.	$B \supset C$	посылка
4.	$A$	допущение
6.	$B$	$\supset_y$ ; 4, 1
7.	$C$	$\supset_y$ ; 6, 2
	$\bigcirc$	
5.	$C$	
3.	$A \supset B$	$\supset_v$ ; 4-5.

Полученная конструкция еще не является натуральным выводом. Мы должны отождествить формулы за номерами 7 и 5 и убрать дырку. Имеются два правила удаления дырки. Первое из них УД 1 имеет вид

$$\begin{array}{c} n.A \\ \bigcirc \Rightarrow n.A \\ m.A \end{array}$$

-анализ с заменой во всей конфигурации цифры  $m$  на цифру  $n$ . В нашем случае будем иметь

1.	$A \supset B$	посылка
2.	$B \supset C$	посылка
4.	$A$	допущение
6.	$B$	$\supset_y$ ; 4, 1
7.	$C$	$\supset_y$ ; 6, 2
3.	$A \supset C$	$\supset_v$ ; 4-7.

Теперь можно осуществить процедуру перенумерации, и в результате получим:

1.	$A \supset B$	посылка
2.	$B \supset C$	посылка
3.	$A$	допущение
4.	$B$	$\supset_y$ ; 3, 1
5.	$C$	$\supset_y$ ; 4, 2
6.	$A \supset C$	$\supset_v$ ; 3-5.

Последняя конфигурация будет стандартным натуральным выводом. В этом выводе мы уже не различаем аналитические и синтетические правила вывода.



Может оказаться, что конфигурация имеет вид  $\alpha$   
 $\bigcirc$   
 $A$   
и в конфигурацию  $\alpha$  входит формула  $A$ . Непосредственно мы не можем применить сформулированное выше правило удаления дырки. Предварительно необходимо применить правило реитерации

$$\begin{array}{ccc} \alpha & m+1.A & \text{реит.}; n, \\ \bigcirc & \rightarrow & \bigcirc \\ m.A & & A \end{array}$$

где формула  $A$  входит в  $\alpha$  за номером  $n$ . После этого правила реитерации можно уже применить правило удаления дырки.

Продemonстрируем изменение правила реитерации на примере. Пусть требуется доказать формулу  $A \supset \supset (B \supset A)$ . Задача формулируется в виде

$$\begin{array}{c} \bigcirc \\ 1. A \supset (B \supset A). \end{array}$$

Применяя АВИ, получаем

$$\begin{array}{lcl} 2. & \left| \begin{array}{c} A \\ \bigcirc \end{array} \right. & \text{допущение} \\ 3. & \left| B \supset A \right. & \\ 1. & A \supset (B \supset A) & \supset_{\text{в}}; 2-3; \end{array}$$

Еще раз применяем АВИ и получаем

$$\begin{array}{lcl} 2. & \left| \begin{array}{c} A \\ 4. \left| \begin{array}{c} B \\ \bigcirc \end{array} \right. \end{array} \right. & \text{допущение} \\ & & \text{допущение} \\ 5. & \left| A \right. & \\ 3. & \left| B \supset A \right. & \supset_{\text{в}}; 4-5 \\ 1. & A \supset (B \supset A) & \supset_{\text{в}}; 2-3 \end{array}$$

Теперь применяем правило реитерации реит.;2 и получаем

$$\begin{array}{lcl} 2. & \left| \begin{array}{c} A \\ 4. \left| \begin{array}{c} B \\ 6. \left| \begin{array}{c} A \\ \bigcirc \end{array} \right. \end{array} \right. \end{array} \right. & \text{допущение} \\ & & \text{допущение} \\ & & \text{реит.}; 2 \end{array}$$

5.			○	
			A	
3			$B \supset A$	$\supset_B; 4-5$
1.			$A \supset (B \supset A)$	$\supset_B; 2-3.$

Остается применить правило удаления дырки:

2.			A	допущение
4.			B	допущение
6.			A	реит.; 2
3.			$B \supset A$	$\supset_B; 4-6$
1.			$A \supset (B \supset A)$	$\supset_B; 2-3$

и после перенумерации имеем

1.			A	допущение
2.			B	допущение
3.			A	реит.; 1
4.			$B \supset A$	$\supset_B; 2-3$
5.			$A \supset (B \supset A)$	$\supset_B; 1-4.$

**З а м е ч а н и е.** Можно не формулировать специального правила реитерации, а в качестве основного правила вывода формулировать правило тождественного перехода  $\frac{A}{A}$ .

В дальнейшем мы сформулируем еще одно правило удаления дырки.

Чтобы обеспечить более систематическое построение вывода для классической логики, нам полезно еще одно правило – аналитическое правило удаления импликации (АУИ):

$n. A \supset B$		$n. A \supset B$	
$\vdots$		$m+1. \mid \neg B$	допущение
$\circ$	$\Rightarrow$	$\mid \circ$	
$s. B$		$m+2. \mid A$	
		$m+3. \mid B$	$\supset_y; m+2, n$
		$m+4. \mid f$	$f_B; m+1, m+3$
		$s. B$	$\neg_y; m+1-m+4.$

При применении этого правила мы указываем номер импликативной формулы: АУИ; *n*. Правило АУИ применяем, когда нельзя применить другие правила для импликации.

Покажем, как это правило полезно для доказательства закона Пирса. Нам требуется доказать закон Пирса, поэтому мы начинаем с конфигурации

○

$$1. ((A \supset B) \supset A) \supset A.$$

На первом шаге применяем АВИ;1 и получаем

$$\begin{array}{ll} 2. & (A \supset B) \supset A \quad \text{допущение} \\ & \quad \quad \quad \circ \\ 3. & A \\ 1. & ((A \supset B) \supset A) \supset A \quad \supset_B; 2-3. \end{array}$$

Далее применяем только что сформулированное правило АУИ;2:

$$\begin{array}{ll} 2. & (A \supset B) \supset A \quad \text{допущение} \\ 4. & \neg A \quad \text{допущение} \\ & \quad \quad \quad \circ \\ 5. & A \supset B \\ 6. & A \quad \supset_y; 5,2 \\ 7. & f \quad f_B; 4,5 \\ 3. & A \quad \neg_y; 4-7 \\ 1. & (A \supset B) \supset A \quad \supset_B; 4-3. \end{array}$$

Теперь применяем АВИ;5 и получаем

$$\begin{array}{ll} 2. & (A \supset B) \supset A \quad \text{допущение} \\ 4. & \neg A \quad \text{допущение} \\ 8. & A \quad \text{допущение} \\ & \quad \quad \quad \circ \\ 9. & B \\ 5. & A \supset B \quad \supset_B; 8-9 \\ 6. & A \quad \supset_y; 5,2 \\ 7. & f \quad f_B; 4,6 \end{array}$$

$$\begin{array}{ll}
3. | A & \neg_y; 4-7 \\
1. ((A \supset B) \supset A) \supset A & \supset_B; 2-3.
\end{array}$$

Чтобы продолжить построения конструкции дальше, необходимо сформулировать синтетическое правило введения константы  $f$  ( $CBf; s, n$ ):

$$\begin{array}{ccc}
s.A & & s.A \\
& & \vdots \\
\vdots & \Rightarrow & n. \neg A \\
& & \vdots \\
n. \neg A & & m+1. f \\
\bigcirc & & \bigcirc
\end{array}$$

В рассматриваемом случае мы имеем

$$\begin{array}{lll}
2. & (A \supset B) \supset A & \text{допущение} \\
4. & | \neg A & \text{допущение} \\
8. & | | A & \text{допущение} \\
10. & | | f & f_B; 4, 8 \\
& | | \bigcirc & \\
9. & | B & \\
5. & A \supset B & \supset_B; 8-9 \\
6. & A & \supset_y; 5, 2 \\
7. & | f & f_B; 4, 6 \\
3. & | A & \neg_y; 4-7 \\
1. & ((A \supset B) \supset A) \supset A & \supset_B; 2-3.
\end{array}$$

Настало время сформулировать второе правило удаления дырки - УД2:

$$\begin{array}{ccc}
s.f & s.f & \\
\bigcirc & \Rightarrow & n.A \quad f_y; s \\
n.A & &
\end{array}$$

В рассматриваемом случае поиска доказательства закона Пирса имеем:

2.	$(A \supset B) \supset A$	допущение
4.	$\neg A$	допущение
8.	$A$	допущение
10.	$f$	$f_B; 4, 8$
9.	$B$	$f_y; 10$
5.	$A \supset B$	$\supset_B; 8-9$
6.	$A$	$\supset_y; 5, 2$
7.	$f$	$f_B; 4, 6$
3.	$A$	$\neg_y; 4-7$
1.	$((A \supset B) \supset A) \supset A$	$\supset_B; 2-3.$

Наконец, после перенумерования имеем

1.	$(A \supset B) \supset A$	допущение
2.	$\neg A$	допущение
3.	$A$	допущение
4.	$f$	$f_B; 2, 3$
5.	$B$	$f_y; 4$
6.	$A \supset B$	$\supset_B; 2-5$
7.	$A$	$\supset_y; 6, 1$
8.	$f$	$f_B; 2, 7$
9.	$A$	$\neg_y; 2-9$
10.	$((A \supset B) \supset A) \supset A$	$\supset_B; 1-9.$

Построенная конфигурация является натуральным выводом закона Пирса из пустого множества посылок.

Чтобы не возникало недоразумений, отметим, что

$\neg_y$  мы называли правило  $\frac{\neg A, \Gamma \vdash f}{\Gamma \vdash A}$  и  $\neg_B$  - правило

$\frac{A, \Gamma \vdash f}{\Gamma \vdash \neg A}$ . Правило, которое нередко называют пра-

виллом удаления отрицания:  $\frac{A, \neg A}{f}$ , мы называем

правилом введения  $f$  и  $\frac{f}{A}$  - правилом удаления  $f$ .

В процедуре построения натурального вывода мы

используем синтетическое правило введения  $f$ :

$$\begin{array}{ccc} A & & A \\ \neg A & \Rightarrow & \neg A \\ \bigcirc & & f \\ & & \bigcirc \end{array}$$

и аналитическое правило удаления  $f$  ( $АУf$ ):

$$\begin{array}{ccc} n.\neg A & & n.\neg A \\ \bigcirc & \Rightarrow & \bigcirc \\ s.f & & m+1.A \\ & & s.f \quad f_B; n, m+1. \end{array}$$

Правилу  $f_y$  соответствует процедура ликвидации дырки УД2.

Мы имеем два аналитических правила для введения и удаления отрицания:

$$\begin{array}{ccc} A \vee \bigcirc: & \bigcirc & \Rightarrow \\ & n.\neg A & \begin{array}{l} m+1. \mid A \\ \bigcirc \\ m+2. \mid f \\ n. \mid \neg A \end{array} \end{array}$$

$$\begin{array}{ccc} A \vee \bigcirc: & \bigcirc & \Rightarrow \\ & n.A & \begin{array}{l} m+1. \mid \neg A \\ m+2. \mid f \\ n. \mid A \end{array} \end{array}$$

В принципе, целесообразно использовать правила снятия и введения двойного отрицания в аналитической и синтетической формах:

$$\begin{array}{ccc} \text{СУДО:} & \begin{array}{ccc} \neg \neg A & \Rightarrow & \neg \neg A \\ \bigcirc & & A \\ & & \bigcirc \end{array} & , \text{СВДО:} \begin{array}{ccc} A & \Rightarrow & A \\ \bigcirc & & \neg \neg A \\ & & \bigcirc \end{array} \\ \text{АУДО:} & \begin{array}{ccc} \bigcirc & \Rightarrow & \bigcirc \\ \neg \neg A & & A \\ & & \neg \neg A \end{array} & , \text{АВДО:} \begin{array}{ccc} \bigcirc & \Rightarrow & \bigcirc \\ A & & \neg \neg A \\ & & A \end{array} \end{array}$$

но, вообще говоря, эти правила производны.

Сформулируем теперь правила для конъюнкции:

$$\begin{array}{lcl}
 \text{СУК:} & \begin{array}{c} n.A \& B \\ \vdots \\ \bigcirc \end{array} \Rightarrow \begin{array}{c} n.A \& B \\ \vdots \\ \bigcirc \end{array} \Rightarrow n+1.A \&_y; n & \text{СВК:} & \begin{array}{c} n.A \\ \vdots \\ \bigcirc \end{array} \Rightarrow \begin{array}{c} n.A \\ \vdots \\ \bigcirc \end{array} \Rightarrow s.B \Rightarrow \begin{array}{c} s.B \\ \vdots \\ \bigcirc \end{array} \\
 & \begin{array}{c} n.A \& B \Rightarrow n.A \& B \\ \vdots \\ \bigcirc \end{array} \Rightarrow n+1.B \&_y; n & & \begin{array}{c} m+1.A \& B \&_B; m, s \\ \vdots \\ \bigcirc \end{array}
 \end{array}$$

$$\begin{array}{c}
 \text{АВК:} \quad n. \begin{array}{c} \bigcirc \\ A \& B \end{array} \Rightarrow m+1. \left| \begin{array}{c} \bigcirc \\ A \end{array} \right. \\
 \qquad \qquad \qquad m+2. \left| \begin{array}{c} \bigcirc \\ B \end{array} \right. \\
 \qquad \qquad \qquad n. \quad A \& B \&_B; m+1, m+2.
 \end{array}$$

Для дизъюнкции мы формулируем три группы правил. Во-первых, два синтетических правила для введения дизъюнкции (СВД):

$$\begin{array}{c} n.A \\ \vdots \\ \bigcirc \end{array} \Rightarrow \begin{array}{c} n.A \\ \vdots \\ \bigcirc \end{array} \Rightarrow m+1.AVB \quad \text{и} \quad \begin{array}{c} n.B \\ \vdots \\ \bigcirc \end{array} \Rightarrow \begin{array}{c} n.B \\ \vdots \\ \bigcirc \end{array} \Rightarrow m+1.AVB.$$

Во-вторых, аналитическое правило удаления дизъюнкции (АУД):

$$\begin{array}{c} n.AVB \\ \vdots \\ \bigcirc \end{array} \Rightarrow \begin{array}{c} n.AVB \\ \vdots \\ \bigcirc \end{array} \Rightarrow \begin{array}{c} m+1. \left| \begin{array}{c} A \\ \bigcirc \end{array} \right. \\ m+2. \left| \begin{array}{c} C \end{array} \right. \\ m+3. \left| \begin{array}{c} B \\ \bigcirc \end{array} \right. \\ m+4. \left| \begin{array}{c} C \end{array} \right. \end{array} \quad \begin{array}{l} \text{допущение} \\ \\ \text{допущение} \end{array} \\
 s.C \vee_y; m+1-m+2, m+3-m+4, n.$$

Необходимо еще одно аналитическое правило введения дизъюнкции. Сформулируем его в следующей форме:

$$\begin{array}{c} \bigcirc \\ s.AVB \end{array} \Rightarrow \begin{array}{c} m+1. \left| \begin{array}{c} \neg(AVB) \\ m+2. \left| \begin{array}{c} A \\ m+3. \left| \begin{array}{c} AVB \\ m+4. \left| \begin{array}{c} f \\ m+5. \left| \begin{array}{c} \neg A \end{array} \right. \end{array} \right. \end{array} \right. \end{array} \right. \end{array} \right. \end{array} \quad \begin{array}{l} \text{допущение} \\ \text{допущение} \\ v_B; m+2 \\ f_B; m+1, m+3 \\ \neg f; m+2-m+4 \end{array}$$

$m+6.$	$B$	допущение
$m+7.$	$AVB$	$\vee_B; m+6$
$m+8.$	$f$	$f_B; m+1, m+7$
$m+9.$	$\neg B$	$\neg_B; m+6-m+8$
	$\bigcirc$	
$m+10.$	$f$	
$z.$	$AVB$	$\neg_y; m+1-m+10.$

Конечно, если бы в натуральном выводе разрешалось пользоваться производным правилом  $\frac{\neg(AVB)}{\neg A \& \neg B}$ , то АВД мы могли бы сформулировать проще:

$\bigcirc$	$\Rightarrow$	$m+1.$	$\neg(AVB)$	допущение
$z.AVB$		$m+2.$	$f$	
		$z.$	$AVB$	$\neg_y; m+1-m+2.$

Правило УВД следует использовать, если нет других возможностей. Например, закон дистрибутивности целесообразно доказывать без использования правила УВД.

Обратимся теперь к кванторным правилам. Все исходные формулы замкнуты, т.е. не содержат свободных вхождений индивидуальных переменных. При построении конструкций будем вводить временные переменные (метапеременные)  $t_1, t_2, \dots$ . Эти переменные вводятся слабыми кванторными правилами; будем считать, что каждое применение слабого правила вводит новую переменную.

Формулировка синтетического правила удаления квантора общности ( $CU\forall; z$ ) проста:

$z. \forall xAx$		$z. \forall xAx$	
$\vdots$		$\vdots$	
$\bigcirc$	$\Rightarrow$	$m+1. At_t$	$\forall_y; z$
		$\bigcirc$	

Здесь  $t_t$  - новая временная переменная.

Правило аналитического введения квантора суще-



ствования нельзя формулировать в столь простой форме  $(AB\exists^+)$ :

$$\begin{array}{c} \circ \\ s.\exists xAx \end{array} \Rightarrow \begin{array}{c} \circ \\ m+1. At_i \\ s.\exists xAx \quad \exists_y; m+1 \end{array}$$

Дело в том, что посылку мы можем использовать многократно, но заключение только однократно. Поэтому надо предусмотреть возможность ее многократного использования в неявном виде. Правило  $AB\exists$  мы формулируем в следующем виде:

$$\begin{array}{c} \circ \\ s.\exists xAx \end{array} \Rightarrow \begin{array}{l|l} m+1. & \exists xAx \quad \text{допущение} \\ \circ & \\ m+2. & At_i \\ m+3. & \exists xAx \quad \exists_y; m+2 \\ m+4. & f \quad f; m+1, m+3 \\ s. & \exists xAx \quad \neg_y; m+1-m+4. \end{array}$$

При вторичном применении, а иногда и в первый раз достаточно использовать частный случай правила  $AB\exists^+$ . Правила для сильных кванторов формулируются просто в терминах  $\varepsilon$ -символа:

$$\begin{array}{l} \text{C}\forall\exists: \quad s.\exists xAx \quad \circ \Rightarrow \begin{array}{l|l} s.\exists xAx & \\ m+1. & A(\varepsilon xAx) \quad \exists_y; s \\ \circ & \end{array} \\ \text{A}\forall\forall: \quad s.\forall xAx \quad \circ \Rightarrow \begin{array}{l|l} m+1. & A(\varepsilon x \neg Ax) \\ s. & \forall xAx \quad \forall_y; m+1. \end{array} \end{array}$$

Естественно,  $\varepsilon$ -термы  $\varepsilon xAx$  и  $\varepsilon x \neg Ax$  могут содержать вхождения временных переменных. Для удобства вместо  $\varepsilon$ -термов мы будем использовать функциональные буквы (нуль- и более местные). Для каждого применения будем использовать новые буквы. По существу, это сколемовские функции. Однако мы не единым разом устраняем положительные кванторы, а в процессе поиска вывода. Приведем простой пример.

Пусть требуется доказать формулу

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

Проведем поиск доказательства:

- 1)  $\bigcirc$ 
  1.  $\exists x \forall y A x, y \supset \forall y \exists x A x, y$
- 2) 2.  $\exists x \forall y A x, y$ 

$$\bigcirc$$
  3.  $\forall y \exists x A x, y$
  1.  $\exists x \forall y A x, y \supset \forall y \exists x A x, y$
- 3) 2.  $\exists x \forall y A x, y$ 
  4.  $\forall y A a, y,$  где  $a = \varepsilon x \forall y A x, y$ 

$$\bigcirc$$
  3.  $\forall y \exists x A x, y$
  1.  $\exists x \forall y A x, y \supset \forall y \exists x A x, y$
- 4) 2.  $\exists x \forall y A x, y$ 
  4.  $\forall y A a, y$
  5.  $A a, t_1$ 

$$\bigcirc$$
  3.  $\forall y \exists x A x, y$
  1.  $\exists x \forall y A x, y \supset \forall y \exists x A x, y$
- 5) 2.  $\exists x \forall y A x, y$ 
  4.  $\forall y A a, y$
  5.  $A a, t_1$ 

$$\bigcirc$$
  6.  $\varepsilon x A x, b,$  где  $b = \varepsilon y \exists x A x, y$
  3.  $\forall y \exists x A x, y$
  1.  $\exists x \forall y A x, y \supset \forall y \exists x A x, y$
- 6) 2.  $\exists x \forall y A x, y$ 
  4.  $\forall y A a, y$
  5.  $A a, t_1$ 

$$\bigcirc$$
  7.  $A t_2, b$
  6.  $\exists x A x, b$

$$3. \mid \forall y \exists x A x, y$$

$$1. \exists x \forall y A x, y \supset \forall y \exists x A x, y.$$

Для простоты мы здесь применили частный случай аналитического правила введения квантора существования. Построенная конструкция еще не является выводом. Мы не можем применить правило УД<sub>1</sub>, так как на аргументных местах стоят разные термы.

Чтобы конструкция превратилась в вывод (после применения правила УД), необходимо во всей конструкции заменить временные переменные на термы. Мы имеем подсказку, какие подстановки надо сделать. Чтобы формулы на местах 5 и 7 были тождественны, надо  $t_2$  заменить на  $a$  и  $t_1$  на  $b$ .

В общем случае мы составляем систему уравнений и ищем ее решение (если таковое существует). В нашем случае мы делаем подстановку  $t_1 := b$ ,  $t_2 := a$ . В результате указанной подстановки мы получаем конструкцию

2.	$\exists x \forall y A x, y$	допущение
4.	$\forall y A a, y$	$\exists y ; 2$
5.	$A a, b$	$\forall_y ; 4$
	○	
7.	$A a, b$	
6.	$\exists x A x, b$	$\exists_b ; 7$
3.	$\forall y \exists x A x, y$	$\forall_b ; 6$
1.	$\exists x \forall y A x, y \supset \forall y \exists x A x, y$	$\supset_b ; 2-3.$

Наконец, применив правило УД<sub>1</sub> и правило перенумерации, имеем

1.	$\exists x \forall y A x, y$	допущение
2.	$\forall y A a, y$	$\exists y ; 1$
3.	$A a, b$	$\forall y ; 2$
4.	$\exists x A x, b$	$\exists_b ; 3$
5.	$\forall y \exists x A x, y$	$\forall_b ; 4$
6.	$\exists x \forall y A x, y \supset \forall y \exists x A x, y$	$\supset_b ; 1-5.$

Отметим, что операция устранения временных переменных может осуществляться не только в конце процедуры. Далее процесс решения системы уравнений между термами может не иметь решения, тогда процесс поиска вывода может быть продолжен.

В качестве примера советуем построить, следуя указанной процедуре, доказательство формулы

$$\exists x \forall y (Ax \supset Ay).$$

## Глава 5. ПРИЛОЖЕНИЕ ЛОГИКИ К ПРОБЛЕМАМ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

### 5.1. Эпистемическая логика и распределенные системы

Эпистемическая логика занимается изучением рассуждений о знании и рассуждений на основании знаний, в состав которых входят такие словосочетания, как "я знаю", "Анна знает, что Петр не знает", "он полагает", "известно" и т.п. Как ветвь одного дерева эпистемическая логика тесно связана с другими областями неклассических логик – модальной, многозначной, релевантной, временной. В последнее время заметно возрос прикладной интерес к логическому анализу знания со стороны компьютерных наук, особенно в связи с такой глобальной задачей, как представление знания в формальных моделях.

#### *Классическая семантика возможных миров*

Имеются давние традиции в изучении предложений мнения и знания как в традиционной, так и в формальной логике XX столетия. Однако оформление эпистемической логики в самостоятельную дисциплину обычно относят к началу 60-х годов – времени разработки и быстрого распространения семантик возмо-

жных миров для неклассических логик. Стратегически исходными стали следующие идеи. Указанный тип семантик предполагает рассмотрение некоторого пространства мыслимых положений дел или возможных миров. Пусть  $W$  есть такое множество возможных миров, а  $i$  — обозначение субъекта. Рассматривая знания субъекта  $i$ , обычно приходят к выделению некоторого подмножества множества  $W$ , на котором субъект как бы концентрирует свое внимание. Обозначим данное подмножество через  $W_1$ . Существенно то, что оно релеативизированно не только относительно субъекта, но и ситуации (возможного мира), в которой находится субъект, т.е. относительно некоторого  $w_0$  — элемента множества  $W$ . Соотнесение подмножества  $W_1$  с субъектом  $i$  и миром  $w_0$  означает введение бинарного отношения  $R$ , определенного на  $W$ . Миры из  $W_1$  представляют собой миры, с которыми  $w_0$  находится в отношении  $R$ .

Возможные миры из выделенного подмножества могут оказаться неразличимыми для субъекта по каким-либо параметрам. К таким параметрам можно отнести знание фактов. Например, можно сказать, что субъект  $i$  знает факт  $\alpha$  в мире  $w_0$ , если и только если  $\alpha$  истинно во всех мирах, достижимых из  $w_0$  по отношению  $R$ , т.е. в элементах  $W_1$ . Миры из  $W_1$  называются эпистемическими альтернативами к  $w_0$ , а отношение  $R$  — отношением альтернативности<sup>34</sup>. Говорят, что эпистемические альтернативы из  $W$  совместимы со всем тем, что субъект  $\alpha$  знает в мире  $w_0$ .

<sup>34</sup> Толкование понятия "возможного мира" значительно расширилось со времени появления первых семантик. Возникли термины, более точно выражающие смысл понятия мыслимого положения дел в рамках каждого конкретного исследования. Так, применительно к знанию в философской логике говорят об эпистемических альтернативах к данному миру, а в теоретических компьютерных науках — о состояниях знания субъекта.

Предположим, что  $W$  состоит из четырех элементов:  $\{w_1, w_2, w_3, w_4\}$ . Пусть  $\alpha$  обозначает предложение "В Москве - туман",  $\beta$  - предложение "В Ленинграде - туман". Миры  $w_1, w_2$  и  $w_3$  связаны отношением  $R$ . Рассмотрим следующую ситуацию, представленную на диаграмме (рис.7).

В мире  $w_1$  субъект знает, что в Москве - туман, но не знает, есть ли туман в Ленинграде. Причем наличие  $\beta$  в  $w_2$  и  $\neg\beta$  в  $w_3$  не противоречит знанию субъектом факта  $\alpha$  в  $w_1$ . Для субъекта миры  $w_1, w_2, w_3$  оказываются неразличимыми относительно параметра  $\alpha$ .

Перейдем к построению модели рассматриваемой ситуации. Будем использовать язык пропозициональной модальной логики с одним оператором. В его словарь входят пропозициональные переменные, логические связки (отрицание  $\neg$ , конъюнкция  $\&$ ) и оператор  $K$ . Если  $\alpha$  и  $\beta$  есть формулы, то  $\neg\alpha$ ,  $(\alpha \& \beta)$  и  $K\alpha$  также есть формулы. В случае с одним субъектом индекс, указывающий на субъекта, обычно опускают. Например, вместо  $K_i$  пишут просто  $K$ , что можно прочитать как "некто знает, что  $\alpha$ ". Импликация и дизъюнкция вводятся по определению обычным образом.

Этот язык можно интерпретировать в реляционных

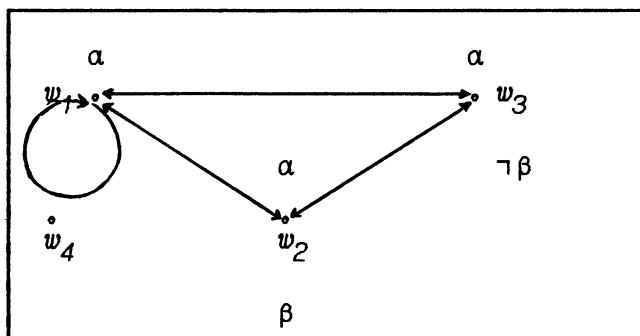


Рис. 7

семантиках (типа Крипке). Под моделью  $\mathbb{M}$  будем понимать упорядоченную тройку  $\langle W, R, \varphi \rangle$ , где  $W$  есть непустое множество возможных миров;  $R$  - бинарное отношение на  $W$  ( $R$  - рефлексивно, симметрично и транзитивно, т.е. является отношением эквивалентности);  $\varphi$  - функция, приписывающая каждой пропозициональной переменной подмножество  $W$ :  $\varphi(p) \subseteq W$ . Неформально  $\varphi(p)$  состоит из тех миров, где имеет место  $p$ .

Определим понятие истины относительно данного мира и данного приписывания. Отношение  $\mathbb{M}, w \models_{\varphi} \alpha$  означает, что  $\alpha$  истинно в мире  $w$  модели  $\mathbb{M}$  при данном приписывании  $\varphi$ . Следующие условия имеют место во всех стандартных реляционных семантиках:

1.  $\mathbb{M}, w \models_{\varphi} p$  для каждой пропозициональной переменной  $p$ , если  $w \in \varphi(p)$ ;
2.  $\mathbb{M}, w \models_{\varphi} \neg \alpha$ , если и только если  $\mathbb{M}, w \not\models_{\varphi} \alpha$ ;
3.  $\mathbb{M}, w \models_{\varphi} (\alpha \& \beta)$ , если и только если  $\mathbb{M}, w \models_{\varphi} \alpha$  и  $\mathbb{M}, w \models_{\varphi} \beta$ ;
4.  $\mathbb{M}, w \models_{\varphi} K\alpha$ , если и только если  $\mathbb{M}, w' \models_{\varphi} \alpha$  для всех  $w'$  таких, что  $wRw'$ .

Формула  $\alpha$  истинна в модели  $\mathbb{M}$ , если и только если  $\mathbb{M}, w \models_{\varphi} \alpha$  для всех  $\varphi$  и для всех  $w \in W$ . Формула  $\alpha$  истинна в классе  $\mathbb{M}$ -моделей, если и только если она истинна в каждой модели данного класса.

Для аксиоматизации могут быть использованы следующие схемы аксиом и правил вывода [37]:

- PL.  $\alpha$ , если  $\alpha$  - пропозициональная тавтология.  
 A1.  $(K\alpha \& K(\alpha \supset \beta)) \supset K\beta$ .  
 A2.  $K\alpha \supset \alpha$ .  
 A3.  $K\alpha \supset KK\alpha$ .  
 A4.  $\neg K\alpha \supset K\neg K\alpha$ .

MP.  $\alpha, \alpha \supset \beta / \beta$  (modus ponens)

RN.  $\alpha / K\alpha$ .

PL и MP составляют пропозициональную часть; A1 утверждает, что знание субъекта замкнуто относительно импликации (выводимости); A2 отражает методологическое требование, предъявляемое к рациональному познанию: "настоящее" знание в отличие от мнения должно содержать истину (согласно A2, субъект знает лишь то, что истинно); A3 и A4 обращают внимание на рефлексивный характер знания: субъект осознает свое знание или незнание и может раскрыть основания своего знания (незнания); A3 часто называют позитивной, а A4 – негативной интроспекцией. Условия {PL, A1, A2, MP, RN} задают систему Т (или М): для Т имеем рефлексивность R в семантике. Условия для Т с аксиомой A3 задают модальную систему S4 (рефлексивность и транзитивность R); наконец, Т с добавлением аксиомной схемы A4 дает систему S5 (рефлексивность, симметричность и транзитивность R).

### *Логическое всеведение и автоэпистемическая логика*

Рассмотренная выше эпистемическая интерпретация стандартных модальных систем неоднократно подвергалась критике. Наибольшее возражение вызвало правило RN и два других более слабых правила – RR:  $\alpha \supset \beta / K\alpha \supset K\beta$  и RE:  $\alpha \equiv \beta / K\alpha \equiv K\beta$ . Все они создают образ слишком "идеальных" субъектов, которые знают все логические истины (RN), все логические следствия своего знания (RR), а также все логически эквивалентные предложения (RE). Такая ситуация была названа *логическим всеведением*. Логическое всеведение неприемлемо не только с философской точки зрения, но и в вычислительных процессах: в условиях огра-



ниченности ресурсов (памяти, времени) вывести все логические следствия невозможно.

Было предложено множество решений, преодолевавших логическое всеведение. Прежде чем привести одно из них, хотелось бы сказать несколько слов в защиту приведенных характеристик знания. Пусть  $K_a$  читается как "я знаю, что  $a$ ". Другими словами, рассмотрим эпистемические контексты первого лица единственного числа. В этом случае  $A_1, A_3$  и  $A_4$  окажутся вполне обоснованными, так как отражают факт осознания субъектом своего личностного знания и его способность к мыслительным операциям на его основе. Относительно  $A_2$  сомнения также рассеются, если иметь в виду социальный характер знания. Субъект не замыкается в своем собственном "Я", а подчеркивает связь своих познаний с нормами и представлениями, признанными в обществе, где он живет.

Системы, в которых субъект как бы размышляет о своем собственном знании на некотором фоне общих знаний или сравнивает свое знание с другими точками зрения, были названы *автоэпистемическими логиками*<sup>35</sup>. В них субъект является одновременно и наблюдателем и наблюдаемым. Очевидно, что логики  $S_4$  и  $S_5$  принадлежат к классу автоэпистемических. Нежелательными в них остаются лишь правила, провоцирующие логическое всеведение. Их пересмотр и привел к новым модификациям этих систем.

Последовательное проведение принципа разграничения горизонта видения субъекта и некоторой другой возможной точки зрения ведет к выделению двух относительно независимых областей в семантике возможных миров. Это можно сделать, например, разли-

---

<sup>35</sup> Термин появился в литературе по искусственному интеллекту. Впервые его употребил Р.Мур, исследуя немонотонные выводы в эпистемической интерпретации [42].

чая полные описания состояний (возможные миры) и частичные. Под полным описанием возможного мира будем понимать множество  $w$  атомарных формул или их отрицаний, такое, что для каждой атомарной формулы  $p$  либо  $p \in w$ , либо  $\neg p \in w$ , но не то и другое вместе. Другими словами, выполняются условия непротиворечивости и полноты описаний состояний. Частичное описание есть непустое множество  $w$  атомарных формул или их отрицаний, такое что ни для какой формулы не верно, что  $p \in w$  и  $\neg p \in w$  вместе. Другими словами, выполняется лишь условие непротиворечивости. На этом принципе построены системы **БрТ**, **Бр4** и **Бр5** [2].

Рассмотрим следующие группы аксиомных схем и правил вывода:

Г р у п п а А:

A2, A3, A4 и

A5.  $K(K\alpha \supset \alpha)$ .

Эту группу составляют схемы-характеристики знания.

Г р у п п а В:

B1.  $K\neg\neg\alpha \equiv \alpha$ .

B2.  $K(\alpha \& \beta) \equiv (K\alpha \& K\beta)$ .

B2a.  $K(\alpha \& (\beta \vee \gamma)) \supset K((\alpha \& \beta) \vee \gamma)$ .

B3.  $K\alpha \supset K(\alpha \vee \beta)$ .

B4.  $K\beta \supset K(\alpha \vee \beta)$ .

B5.  $K\neg(\alpha \& \beta) \equiv K(\neg\alpha \vee \neg\beta)$ .

B6.  $K\neg(\alpha \vee \beta) \equiv K(\neg\alpha \& \neg\beta)$ .

Данную группу составляют схемы, характеризующие особенности возможных миров - эпистемических альтернатив.

П р а в и л а в ы в о д а:

MP, а также

RD. Из  $K\alpha \supset K\gamma$  и  $K\beta \supset K\gamma$  следует  $K(\alpha \vee \beta) \supset K\gamma$ , где посылки являются подстановочными частными случаями

доказуемых формул  $K\alpha' \supset K\gamma'$  и  $K\beta' \supset K\gamma'$  с немодализированными  $\alpha'$ ,  $\beta'$  и  $\gamma'$ .

RN. Если  $\alpha$  полностью модализирована, то из  $\alpha$  следует  $K\alpha$ .

ЕрТ = (PL, группа В, А2, А5, МР, RD и RN).

Ер4 = (ЕрТ, А3 ).

Ер5 = (ЕрТ, А4).

Перейдем к построению семантики для этих систем. Ер-модельной структурой будем называть следующую упорядоченную тройку:  $S = \langle W, H, R \rangle$ , где  $W$  - множество полных описаний состояний (возможных миров);  $H$  - множество частичных описаний состояний;  $R$  - бинарное отношение на  $W \cup H$ , удовлетворяющее условию (K): если  $v_1 R v_2$  и  $v_1 \neq v_2$  или же  $v_1 \in H$ , то  $v_2 \in H$  ( $v_1, v_2 \in W \cup H$ ).

При рефлексивности  $R$  имеем ЕрТ-модельную структуру. При транзитивности  $R$  в ЕрТ-модельной структуре получаем Ер4-модельную структуру. При симметричности  $R$  в Ер4-модельной структуре получаем Ер5-модельную структуру.

Пусть  $X = \text{ЕрТ, Ер4 или Ер5}$ .  $X$ -моделью формул, принадлежащих системе  $X$ , будем называть  $\langle S, \varphi \rangle$ , где  $S$  есть  $X$ -модельная структура, а  $\varphi$  - есть функция приписывания переменным. Для каждой переменной  $p$   $\varphi$  приписывает два подмножества:  $\varphi_T(p) \subseteq W \cup H$  и  $\varphi_F(p) \subseteq W \cup H$ . Подмножество  $\varphi_T(p)$  есть множество миров, где  $p$  истинно, а  $\varphi_F(p)$  - множество миров, где  $p$  ложно. Заметим, что  $W$  состоит из полных миров, которые являются в точности членами одного из множеств  $\varphi_T(p)$  или  $\varphi_F(p)$  для любого  $p$ , а  $H$  состоит из частичных миров, не входящих ни в  $\varphi_T(p)$ , ни в  $\varphi_F(p)$  для не-которого высказывания  $p$ .

Введем понятие истинности (выполнимости) формул в данном мире при данном приписывании. Запись  $X, v \models_{\varphi} \alpha$  означает, что формула  $\alpha$  истинна (выполнима) в мире  $v$  модели  $X$  при приписывании  $\varphi$ , а

$X, v \models_{\varphi} \alpha$  означает, что формула  $\alpha$  ложна (не выполняема) в  $v$  модели  $X$  при приписывании  $\varphi$ . Пусть  $v \in W \cup H$ , тогда имеем:

1.  $X, v \models_{\varphi} p$ , если и только если  $v \in \varphi_T(p)$ ;  
 $X, v \models_{\varphi} p$ , если и только если  $v \in \varphi_F(p)$ .
2.  $X, v \models_{\varphi} \neg \alpha$ , если и только если  $X, v \not\models_{\varphi} \alpha$ ;  
 $X, v \models_{\varphi} \neg \alpha$ , если и только если  $X, v \models_{\varphi} \alpha$ .
3.  $X, v \models_{\varphi} (\alpha \& \beta)$ , если и только если  $X, v \models_{\varphi} \alpha$  и  $X, v \models_{\varphi} \beta$ ;  
 $X, v \models_{\varphi} (\alpha \& \beta)$ , если и только если  $X, v \models_{\varphi} \alpha$  или  $X, v \models_{\varphi} \beta$ .
4.  $X, v \models_{\varphi} K\alpha$ , если и только если  $X, v' \models_{\varphi} \alpha$  для всякого  $v'$  из  $W \cup H$ , такого что  $vRv'$ ;  
 $X, v \models_{\varphi} K\alpha$ , если и только если  $X, v' \not\models_{\varphi} \alpha$  и  $X, v' \models_{\varphi} \alpha$ , или  $X, v' \models_{\varphi} \alpha$  для некоторого  $v' \in W \cup H$ , такого, что  $vRv'$ .

Формула  $\alpha$  истинна в модели, если она истинна в ней для каждого  $w \in W$  при любом  $\varphi$ .

Доказано, что системы **ЕрТ**, **Ер4** и **Ер5** непротиворечивы, полны и разрешимы [2].

### *Знание как обоснованное мнение. Неявное знание*

Рассмотрим оператор знания в третьем лице единственном числе, что соответствует прочтению  $K\alpha$  как "он знает, что  $\alpha$ ". Схема АЗ покажется слишком сильной: не всегда человек может знать основания своего знания, так как рефлексия предполагает знание методологии, т.е. познания более высокого уровня (специальные знания). Вообще говоря, выражение "знает, что знает" имеет много смыслов. В частности, может интерпретироваться и как осознание своего знания и как знание оснований. Если в автоэписте-

мической логике оба глагола "знает" в данном выражении еще можно определить как однопорядковые (субъект как бы просматривает одно состояние своего знания из другого), то в общем случае они оказываются разнопорядковыми (знает<sub>1</sub>, что знает<sub>2</sub>...). Например, знание оснований знания – не одно и то же, что знание фактов.

Разрыв между знанием в первом смысле и знанием во втором смысле усиливается в контекстах третьего лица из-за отделенности наблюдателя от субъекта. Существует определенная разница между информацией, которая известна наблюдателю, и информацией субъекта (в представлении наблюдателя). Еще более это очевидно прослеживается в схеме А4, поскольку незнание может касаться именно той информации, которой обладает исключительно наблюдатель, а субъект о ней даже не подозревает.

В автоэпистемической логике наблюдатель, являясь в то же самое время наблюдаемым, находится *внутри* эпистемической системы. В эпистемической логике третьего лица наблюдатель – *вне* системы личностного знания субъекта. Эта предпосылка наиболее последовательно проводится в двухуровневых системах. Внешний уровень таких систем содержит как рассуждения наблюдателя об объектах, так и оценки субъекта об этих объектах. На внутреннем уровне представлены собственно рассуждения субъекта.

Приведем пример такой двухуровневой системы. В отличие от предыдущих она основана на синтаксическом подходе к модальностям. Это означает, что модальности интерпретируются не как операторы, а как предикаты на предложениях. С каждым эпистемическим выражением связывается некоторое множество предложений. Например, *Ка* означает, что *α* принадлежит множеству предложений – области определения *К*.

Введем новый оператор *ВК* и будем рассматривать

его в качестве базового. Запись  $BK_i\alpha$  можно прочитать как "некто полагает, что знает, что  $\alpha$ ", где "некто" означает неопределенного, но фиксированного субъекта. Словосочетание "полагает, что знает" в языке наблюдателя соответствует выражению "я знаю", где личное местоимение "я" относится к субъекту. Более того, будем рассматривать не один оператор, а некоторое семейство операторов  $\{BK_i; i \in I\}$ . Основанием для этого послужила идея многомерности индивидуального сознания. В семантическом плане она ведет к рассмотрению множественности "областей полаганий" (знаний) субъекта, или, другими словами, множественности относительно законченных областей рассуждений. Например, в область действия оператора знания в высказывании "он знает, что дважды два - четыре" нельзя произвольным образом ввести с помощью дизъюнкции  $p/p \vee q$  "постороннее" высказывание и получить: "он знает, что дважды два - четыре ( $p$ ) или по пятницам Иванов ходит в баню ( $q$ )". Оба высказывания  $p$  и  $q$  принадлежат разным сферам знаний.

Пусть  $L$  - язык пропозициональной логики, а  $L^E$  - эпистемический язык. Множество формул языка  $L^E$  есть наименьшее множество  $Form$  такое, что

1.  $p \in Form$  для любой пропозициональной переменной  $p$ .
2. Если  $\alpha \in Form$ , то  $\neg \alpha \in Form$ .
3. Если  $\alpha, \beta \in Form$ , то  $(\alpha \& \beta) \in Form$ .
4. Если  $\alpha$  - немодальная формула языка  $L$ , то  $BK_i\alpha \in Form$  для каждого  $i \in I$ .

Другие связки определяются обычным образом. Итерированные модальности не рассматриваются.

Пусть  $\alpha$  и  $\beta$  - немодальные пропозициональные формулы,  $\gamma$  и  $\delta$  - формулы эпистемического множества  $Form$ ,  $i$  - произвольный элемент  $I$ . Система  $BKL$  определяется следующими схемами и правилами:

PL.  $\gamma$ , если  $\gamma$  - пропозициональная тавтология.

B1.  $BK_i \neg \neg \alpha \equiv BK_i \alpha$ .

B2.  $BK_i (\alpha \& \beta) \equiv BK_i \alpha \& BK_i \beta$ .

B7.  $BK_i \neg (\alpha \& \beta) \equiv BK_i \neg \alpha \vee BK_i \neg \beta$ .

B8.  $BK_i \alpha \supset \neg BK_i \neg \alpha$ .

MP.  $\gamma, \gamma \supset \delta / \delta$ .

Построим семантику возможных миров для **BEL**:  $\mathbb{M} = \langle W, U, s_i : i \in I, \varphi \rangle$  есть модель для **BEL**, если и только если  $W$  есть непустое (базисное) множество возможных миров;  $U$  - непустое (вспомогательное) множество возможных миров;  $s_i : i \in I$  - семейство функций следующего типа: для каждого  $i \in I$   $s_i$  есть функция из  $W$  в  $U$ ;  $\varphi$  есть функция приписывания оценок. Для каждой пропозициональной переменной  $p$

$$\varphi(p) = \langle \varphi_O(p), \varphi_T(p), \varphi_F(p) \rangle,$$

где  $\varphi_O(p) \subseteq W$ ,  $\varphi_T(p) \subseteq U$ ,  $\varphi_F(p) \subseteq U$ ;  $\varphi_O(p)$  и  $\varphi_T(p)$  есть множество миров, где  $p$  истинно, а  $\varphi_F(p)$  - множество миров, где  $p$  ложно.

Имеет место следующий принцип непротиворечия:

$$\varphi_T(p) \cap \varphi_F(p) = \emptyset.$$

Расширим оценку  $\varphi$  до сложных формул языка:

$$\varphi_T(\alpha \& \beta) = \varphi_T \alpha \cap \varphi_F \beta; \quad \varphi_F(\alpha \& \beta) = \varphi_F \alpha \cup \varphi_F \beta;$$

$$\varphi_T(\neg \alpha) = \varphi_F \alpha; \quad \varphi_F(\neg \alpha) = \varphi_T \alpha.$$

Определим понятия истинности и ложности формул в данном мире из  $U$  при данном приписывании. Для произвольной пропозициональной (немодальной) формулы  $\alpha$  и любого  $u \in U$  имеет место

$$\mathbb{M}, u \models_{\varphi} \alpha, \text{ если и только если } u \in \varphi_T \alpha;$$

$$\mathbb{M}, u \models_{\varphi} \alpha, \text{ если и только если } u \in \varphi_F \alpha.$$

Из определений следует, что функции  $\varphi_T$  и  $\varphi_F$  используются для оценки формул относительно вспомогательного множества  $U$ ,  $\varphi_O$  приписывает оценку переменным в мирах из  $W$ . Неформально  $W$  включает множества полных и непротиворечивых описаний состоя-

ний, а  $U$  состоит из частичных непротиворечивых состояний.

Пусть  $\alpha$  и  $\beta$  – произвольные формулы множества  $Form$  эпистемического языка  $L^e$ ,  $\gamma$  – пропозициональная (немодальная) формула языка  $L$ . Определим понятие истинности формулы в мирах из базового множества  $W$  модели  $\mathbb{M}$  при данном приписывании  $\varphi$ :

$\mathbb{M}, w \models_{\varphi} p$ , если и только если  $w \in \varphi_O(p)$ , в противном случае  $\mathbb{M}, w \not\models_{\varphi} p$ ;

$\mathbb{M}, w \models_{\varphi} (\alpha \& \beta)$ , если и только если  $\mathbb{M}, w \models_{\varphi} \alpha$  и  $\mathbb{M}, w \models_{\varphi} \beta$ , в противном случае  $\mathbb{M}, w \not\models_{\varphi} (\alpha \& \beta)$ ;

$\mathbb{M}, w \models_{\varphi} \neg \alpha$ , если и только если  $\mathbb{M}, w \not\models_{\varphi} \alpha$ , в противном случае  $\mathbb{M}, w \models_{\varphi} \neg \alpha$ ;

$\mathbb{M}, w \models_{\varphi} BK_i \gamma$ , если и только если  $\mathbb{M}, s_i(w) \models_{\varphi} \gamma$ , в противном случае  $\mathbb{M}, w \not\models_{\varphi} BK_i \gamma$ .

Формула  $\alpha \in Form$  истинна в модели  $\mathbb{M} = \langle W, U, s_i : i \in I, \varphi \rangle$ , если и только если  $\mathbb{M}, w \models_{\varphi} \alpha$  для каждого  $w \in W$  при любом  $\varphi$ .

Операторы знания и заблуждения можно ввести по определению:

$$K_i \alpha =_{Df} BK_i \alpha \ \& \ \alpha; \quad E_i \alpha =_{Df} BK_i \alpha \ \& \ \neg \alpha.$$

Можно сказать, что в системе **BKL** описывается знание как обоснованное мнение для контекстов третьего лица: субъект утверждает, что он знает факт  $\alpha$  ( $\alpha$  истинно в некотором состоянии знания субъекта  $u \in U$ , входящем в область знания, определяемую функцией  $s_i$ ), и этот факт  $\alpha$  подтверждается наблюдателем ( $\alpha$  истинно в базовом мире  $w \in W$ , ассоциированном с  $u$  при помощи функции  $s_i$ ). В системе допускается противоречивость мнений-знаний субъекта, принадлежащих разным областям рассуждений, поскольку возможна истинность как  $BK_i \alpha$ , так и  $BK_j \neg \alpha$ , где  $i \neq j$ . Противоречие невозможно внутри области



рассуждения (недопустима одновременная истинность  $BK_1\alpha$  и  $BK_1\neg\alpha$ ), а также для знания. Заметим, что система **BEL** описывает потенциальное знание субъекта – возможные области его знаний.

Доказано, что **BEL** – непротиворечива и полна относительно заданной семантики [7].

В системе **BEL** требование истинности знания, т.е. схема  $A2$ , соблюдается по определению оператора  $K$ . Так же проходит замкнутость относительно импликации (схема  $A1$ ). Однако ни аксиомы интроспекции ( $A3$  и  $A4$ ), ни правила  $RN$ ,  $RR$  и  $RE$  не имеют места. Заметим, что данные правила как бы устанавливают мост между сферой логически истинного (фиксированной точкой зрения наблюдателя) и знаниями субъекта. Полное разрушение этого моста привело бы от идеального "знайки" к идеальному "незнайке" – субъекту, не умеющему получать логические следствия из своего знания. Кроме того, в контекстах третьего лица глупым окажется скорее наблюдатель, неспособный понять наблюдаемого, чем человек, о знании которого идет речь.

Размышления над логическим всеведением привели к созданию систем, где различаются *явное* (*explicit*) и *неявное* (*implicit*) знание. До сих пор речь шла только о явном знании субъекта. Наблюдатель вправе утверждать, что субъект нечто знает лишь после того, как последний проявит свое знание – сообщит о нем. Подобным образом нельзя сказать, что "он думает то-то и то-то", пока нет результата этого процесса, т.е. пока субъект не скажет, о чем он думает. Однако во многих случаях о людях судят по косвенным источникам, а также дополняют имеющиеся сведения игрой собственного воображения. Механизмы подразумеваемого и неявного знания являются неотъемлемой чертой коммуникационных процессов.

Для описания неявного знания можно использовать

измененное правило RN: если из  $\alpha$  выводимо  $\beta$ , то, если субъект явно знает  $\alpha$ , он неявно знает  $\beta$ . Формально:  $\alpha \supset \beta / KE\alpha \supset KI\beta$ .

Имеются различные стратегии описания неявного знания. Допустим, что введены описания состояний явного знания субъекта как частичные, непротиворечивые множества. Их можно расширить за счет информации, известной наблюдателю, и получить, таким образом, новые следствия. Разумеется, вводимая информация не должна противоречить исходной.

Возможен и другой путь, когда неявное знание берется в качестве исходного. В данном случае оно понимается скорее как потенциальное или возможное знание. К каждому эпистемическому описанию состояния добавляется множество предложений, которые "осознаются" субъектом в этом состоянии. Тогда можно сказать, что субъект явно знает  $\alpha$  в мире  $w$ , если  $\alpha$  истинно во всех альтернативах к  $w$  и  $\alpha$  попадает в множество "осознаваемых" субъектом предложений. Предположим, что мы имеем дело с некоторой дистрибутивной системой, где каждый процессор вычисляет, что он "знает" из полученной информации, реализует некий алгоритм. В множество "осознанных" предложений попадут как раз те предложения, истинность которых может быть продемонстрирована с помощью заданного алгоритма [30].

### *Многосубъектные системы. Групповое знание*

Системы, описывающие модальности третьего лица единственного числа, являются неявно двухсубъектными, т.к. предполагают явное указание на действующее лицо, а наблюдатель лишь подразумевается. Явно указав наблюдателя, переходим к многосубъектным системам ("Сидоров знает, что Иванов знает, что Петров ходит по пятницам в баню").

Вернемся к семантикам типа Крипке, рассмотрен-

ным в первом разделе. Многосубъектной моделью можно считать структуру  $\mathbb{M} = \langle W, R_1, \dots, R_n, \varphi \rangle$ , где  $W$  и  $\varphi$  определяются, как прежде, а  $R_i$  есть отношение эквивалентности на  $W$  для любого  $i=1, \dots, n$ . Каждое  $R_i$  можно рассматривать как отношение альтернативности, ассоциированное с субъектом  $i$ . Определение истинности для операторов знания будет выглядеть следующим образом:

$\mathbb{M}, w \models_{\varphi} K_i \alpha$ , если и только если  $\mathbb{M}, w' \models_{\varphi} \alpha$  во всех  $w'$ , таких что  $w R_i w'$  для любого  $i=1, \dots, n$ .

Характерной чертой многосубъектных систем является наличие цепочек итерированных операторов типа  $K_A K_B K_C \neg K_A \alpha$ . Это обстоятельство приводит к чрезвычайному усложнению описаний возможных состояний дел. Нетривиальные модели возникают даже при рассмотрении единственного высказывания и двух субъектов.

Особое место в исследованиях многосубъектных систем занимает проблема *группового знания*, или *разделенного мнения*. Она связана с разработкой сложных коммуникативных систем, в которых субъект должен согласовывать совместные действия, уметь добиваться взаимопонимания, уметь прогнозировать по аналогии с действиями других и т.п. Д.Льюис предложил [40] следующее понимание группового знания, которое затем было использовано в литературе по искусственному интеллекту: группа имеет общее знание о  $\alpha$ , если не только каждый член группы знает, что  $\alpha$ , но и каждый знает, что каждый знает... что  $\alpha$ .

Поясним это определение на примере [45]. Допустим, что в клубе "Звезда" по вечерам в течение недели идет фильм-детектив. А спрашивает В: "Ты когда-либо смотрел фильм, который идет в "Звезде" по вечерам?". Предполагается, что и А, и В знают, о каком фильме идет речь. Также А должен знать, что

В знает об этом фильме, и наоборот. Более того, А должен знать, что В знает, что А знает и т.д.

Пусть  $E\alpha$  обозначает "каждый знает, что  $\alpha$ ", а  $S\alpha$  обозначает " $\alpha$  есть общее знание". Новые операторы можно определить следующим образом в семантиках типа Крипке:

$\mathbb{M}, w \models_{\varphi} E\alpha$ , если  $\mathbb{M}, w' \models_{\varphi} \alpha$  во всех  $w'$ , таких, что

$\langle w, w' \rangle \in R_1 \cup \dots \cup R_n$ . Это означает, что

$\mathbb{M}, w \models_{\varphi} E^l \alpha$ , если  $\mathbb{M}, w \models_{\varphi} K_1 \alpha \& \dots \& K_n \alpha$ ;

$\mathbb{M}, w \models_{\varphi} S\alpha$ , если  $\mathbb{M}, w \models_{\varphi} E^l \alpha$  для  $l=1, 2, \dots$ , где  $E^1 \alpha = E\alpha$  и  $E^{l+1} \alpha = EE^l \alpha$ .

Существенную роль в достижении общего знания играют условия коммуникации. Проиллюстрируем это на примере, вошедшем в фольклор по распределенным системам [45].

Две дивизии одной армии расположились на двух холмах, возвышающихся над общей долиной. В долине находится враг. Ясно, что если обе дивизии атакуют врага одновременно, то они выиграют сражение, тогда как атака одной дивизией приведет к поражению. Плана нанесения первоначального удара по врагу не существует, и командующий первой дивизией генерал А хотел бы скоординировать одновременность нападения в определенное время на следующий день. Ни один генерал не решится атаковать, пока не будет уверен, что другой начнет атаку вместе с ним. Обмениваться информацией генералы могут только через посланника. Обычно посланник добирается из одного лагеря в другой за один час. Однако вполне вероятно, что он может заблудиться в темноте или, еще хуже, его перехватит враг. К счастью, именно в эту ночь все идет благополучно. Сколько понадобится времени генералам для того, чтобы согласовать атаку?

Предположим, что посланник генерала А прибывает

к генералу В с запиской: "Давай атаковать на рассвете". Будет ли генерал В атаковать? Разумеется, нет, так как генерал А не знает, что он получил записку, и, таким образом, не может атаковать. Поэтому генерал В посылает посланника обратно с одобрением предложения генерала А. Предположим, что посланник проделал и это поручение. Будет ли генерал А атаковать? Нет, так как сейчас генерал В не знает, получил ли А записку; поэтому он думает, что генерал А может думать, что он, т.е. В, не получил действительную записку, и, таким образом, не атакует. Генерал А вновь посылает посланника обратно с подтверждением. Но опять-таки этого оказывается недостаточно, даже если действительно посланник приносит записки вовремя. Сумма подтверждений вовсе не гарантирует достижения согласия, так как вновь и вновь возникает возможность невыполнения задания посланником.

Если описать данную ситуацию в терминах знания, то с каждым приходом посланника в лагерь возрастает глубина "генеральских знаний" от "В знает" к "А знает, что В знает, что А знает" и т.д. Из-за этих условий атака не может начаться на рассвете. Исследования показали, что в любой системе, где обмен информацией не гарантирован, общее знание не достижимо. Оно не достижимо и в системах, где коммуникация гарантирована, но имеется неопределенность во времени передачи сигналов [45].

#### *Интерпретация семантик возможных миров в многопроцессорных системах*

Как задача о двух генералах, так и многие другие проблемы эпистемической логики исследовались в многопроцессорных вычислительных системах. Их создание – плод работы многих специалистов по искусственному интеллекту [45-47]. Описание "знания"

процессора в таких системах опирается на многие ключевые идеи семантики возможных миров, поэтому их можно рассматривать как конкретизацию логических семантик.

Многопроцессорная система представляет собой конечную совокупность вычислительных устройств – процессоров, связь между которыми осуществляется при помощи соединений в сети. Каждый процессор выполняет определенные операции и все время находится в некотором состоянии. Это состояние зависит от набора функций, имеющихся в процессоре, получаемой внешней информации, а также от некоторых факторов внутреннего характера, например боя встроенных часов.

Под состоянием процессора можно понимать его кругозор – полное описание информации о процессоре. Последнее предполагает указание времени начала работы, исходного состояния, истории посланий (передач и ответов), протокола (описания действий), которому он следует. В другой интерпретации под состоянием имеют в виду внутреннее состояние процессора в фиксированном процессе. В последнем случае процессор может "забывать" факты, которые он прежде "знал". Глобальное состояние системы есть описание состояний процессоров.

Если так можно выразиться, процессоры выполняют функции субъектов в эпистемической логике. Приписывание процессорам знаний означает тот факт, что наблюдатель использует термин "знает" для оценки определенных состояний процессора. Говорят, что процессор знает, что  $p$ , если состояние процессора таково, что имеет место  $p$ . В каждой конкретной многопроцессорной системе то, что известно процессору, зависит от задач исследования. Например, "знание" может быть информацией, хранящейся в базе данных процессора. В другой модели знание обо-

гащается информацией, которую процессор может вывести (вычислить).

Рассмотрим многопроцессорную систему в эпистемической интерпретации. Предполагается, что язык включает в себя множество базисных фактов – высказываний, не находящихся в области действия эпистемических операторов. Такими высказывания могут быть утверждения типа "переменной  $x$  приписывается значение 0", "процессор  $j$  находится в состоянии  $\delta$ ". Пусть  $W$  есть множество всевозможных глобальных состояний многопроцессорной системы. Другими словами,  $W$  есть множество процессов, происходящих в определенное время. Элементы  $W$  будем называть также точками соотнесения.  $R_i$  есть бинарное отношение на  $W$ ,  $\varphi$  – функция, приписывающая значения базисным фактам:  $\varphi(p) \subseteq W$ . Понятие истины в точке соотнесения  $w \in W$  модели  $\mathbb{M}$  обычное для простых и сложных высказываний, образованных с помощью булевых связок. Истинность высказывания "процессор  $i$  знает, что  $\alpha$ " также определяется обычным образом:

$\mathbb{M}, w \models_{\varphi} K_i \alpha$ , если и только если  $\mathbb{M}, w' \models_{\varphi} \alpha$  для всех  $w'$ , таких, что  $w R_i w'$ .

Тот факт, что пара  $\langle w, w' \rangle$  принадлежит отношению  $R_i$ , означает, что процессор  $i$  находится в одном и том же состоянии в точках  $w$  и  $w'$ . Очевидно, что глобальные состояния, связанные отношением  $R_i$ , неразличимы относительно состояния процессора  $i$ . Другими словами,  $R_i$  есть отношение эквивалентности (рефлексивно, симметрично, транзитивно). Просматривая глобальные состояния, наблюдатель делает утверждение о том, что процессор  $i$  знает, что  $\alpha$ , поскольку  $\alpha$  истинно во всех глобальных состояниях, совместимых с текущим состоянием процессора. Очевидно, что все аксиомы системы S5 имеют место при такой интерпретации.

Пусть  $G$  есть группа, состоящая из  $m$  процессоров. Будем говорить, что  $w'$  достижимо из  $w$  по отношению  $R$ , если существуют точки  $w_1, \dots, w_m$  и процессоры  $f_1, \dots, f_{m-1} \in G$ , такие, что  $w = w_1$ ,  $w' = w_m$ , и процессор  $f_l$  находится в одном и том же состоянии в  $w_l$  и в  $w_{l+1}$  для  $1 \leq l < m$ . Определим операторы группового знания:

$\mathbb{M}, w \models_{\varphi} E_G \alpha$ , если и только если  $\mathbb{M}, w' \models_{\varphi} \alpha$  для всех  $w'$ , таких, что  $w R_i w'$  для некоторого  $i \in G$ ;

$\mathbb{M}, w \models_{\varphi} C_G \alpha$ , если и только если  $\mathbb{M}, w' \models_{\varphi} \alpha$  для всех  $w'$ , достижимых из  $w$  по отношению  $R$ .

Нетрудно проверить, что  $E^m \alpha$  имеет место для всех  $m$  в точке  $w$ , исходя из определения, что  $E_G \alpha$ , если и только если  $\alpha$  имеет место во всех точках  $w'$ , достижимых из  $w$ .

## 5.2. Рассуждения при ограниченных ресурсах.

### Логика с ресурсами

Интересным свойством человеческих рассуждений является то, что некоторый процесс логического вывода может выполняться в течение ограниченного отрезка времени, а потом может быть отброшен или замещен чем-то другим. Пробуя доказать какое-либо утверждение, человек может не успеть сделать это за ограниченное время. Если у человека появляется необходимость совершить какое-либо действие, то он может вывести негативный факт — "Если я не смог до сих пор доказать этого, это должно быть ложным". Вполне возможно, что при меньших ограничениях на время человек сможет доказать это утверждение. В качестве ресурсов может выступать не только время, но и объем расходуемой памяти, число обращений к оракулу (справочникам), энергетические затраты, все то, что как-то расходуется в процессе рассуждений.



Две тысячи лет назад для человека морская вода была просто раствором соли. Он был уверен, что в ней больше ничего нет. Это сейчас мы знаем, что в морской воде можно обнаружить все элементы таблицы Менделеева. Древний человек, кроме соли, ничего извлечь из нее не мог. Если бы мы попросили его верифицировать предложение "Морская вода содержит золото", то он сказал бы нам, что оно ложно, либо сразу, либо после еще одной безуспешной попытки выпарить воду и извлечь золото. Ресурсы не позволяли ему установить факт наличия золота в воде. Эта истина была для него недостижима. Древний человек планировал свои действия из убеждения, что золота и еще многого другого в воде нет. Для него это было прагматически истинно. Как ни странно, но такое огрубление даже приносило определенную пользу. Оно позволило увидеть определенный порядок в первоначальном хаосе.

Казалось бы, практика древнего человека ушла в прошлое. Приведем примеры, которые покажут, что это не так.

Пусть нам даны два действительных числа, представленных в виде бесконечных десятичных дробей:

$$0,a_1a_2a_3 \dots \text{ и } 0,b_1b_2b_3\dots$$

Будем считать, что эти дроби порождаются двумя генераторами. На порождение каждой цифры в этих десятичных представлениях затрачивается определенное время, расходуются и другие ресурсы. Допустим, что мы умеем сравнивать цифры, стоящие на одинаковых местах в этих последовательностях. Легко видеть, что нашего умения недостаточно для того, чтобы установить факт равенства хотя бы одной пары действительных чисел, так как для этого нам потребовалось бы породить и просмотреть две бесконечные последовательности. Самое большее, на что мы

способны, — это установить, что два числа неравны, когда на каком-то шаге сравнения обнаружим, что на одинаковых местах последовательностей стоят различные цифры. Однако бывает так, что с прагматической точки зрения, с точки зрения той цели, ради достижения которой мы интересуемся равенством (неравенством) двух данных чисел, мы можем пойти на компромисс между достигнутой точностью и израсходованными ресурсами, приняв, что два числа равны, если совпадают, скажем, первые десять цифр после запятой. Если даже цифры, стоящие в одиннадцатой позиции, не совпадут, это мало повлияет на практические последствия нашего приравнивания. Исходная частично рекурсивная процедура позволяет перечислять пары неравных действительных чисел. Наложив разумные ограничения на число ее шагов, мы можем получать с ее помощью и практически приемлемые результаты о равенстве чисел.

Уже имеются системы искусственного интеллекта, в которых явным образом производится распределение ресурсов в процессе вывода. В них принимается принцип, который можно выразить следующим образом: "Пробуй доказать  $X$ , пока не исчерпаешь ресурсов, а потом, если доказать  $X$  не удастся, заключи, что  $X$  ложно".

Рассмотрим, как расходуются ресурсы времени в процессе верификации булевых формул. Примем допущение, что система, использующая логику с ресурсами, является системой последовательной обработки информации. Формулы обрабатываются слева направо. Целью такой обработки является их верификация или фальсификация. Время будем считать имеющим дискретную линейную структуру. Всякое вычисление начинается в какой-то момент времени. Таким образом, можно считать, что ресурсы имеют структуру, изомор-

фнкую множеству натуральных чисел с естественным порядком на нем.

Определим язык.

Алфавит:

1.  $p_1, p_2, p_3, \dots \in Var$  - множество пропозициональных переменных.
2.  $\&, \vee, \neg, >$  - логические связки.
3.  $V_1, V_2, V_3, \dots$  - операторы верификации с ресурсами.

Формулы:

1. Всякая пропозициональная переменная является формулой.

2. Если  $A$  и  $B$  - формулы, то и  $\neg A$ ,  $(A \& B)$ ,  $(A \vee B)$ ,  $(A > B)$ ,  $V_n A$  - формулы.

3. Ничто другое формулой не является.

Формулы вида  $V_n A$  читаются: " $A$  верифицируема за время  $n$ ". Связка эквивалентности " $\leftrightarrow$ " вводится по определению обычным образом.

Модель:

Моделью будем называть пару  $\mathbb{M} = \langle v, \alpha \rangle$ , где

$v: Var \rightarrow \{0, 1\}$  - истинностная оценка пропозициональных переменных:  $\alpha: Var \rightarrow \mathbb{N}^+ \cup \{\omega\}$  - оценка ресурсов

Для  $\omega$  выполняются условия  $\omega + n = n + \omega = \omega + \omega = \omega$  и для всякого  $n$  имеет место  $n < \omega$ .

Обе оценки по индукции распространим на множество всех формул следующим образом:

1.  $v(\neg A) = 1 - v(A)$ ,  $\alpha(\neg A) = \alpha(A)$ .
2.  $v(A \& B) = \begin{cases} 0, & \text{если } v(A) = 0, \\ v(B), & \text{если } v(A) = 1, \end{cases}$   
 $\alpha(A \& B) = \begin{cases} \alpha(A), & \text{если } v(A) = 0, \\ \alpha(A) + \alpha(B), & \text{если } v(A) = 1. \end{cases}$
3.  $v(A \vee B) = \begin{cases} 1, & \text{если } v(A) = 1, \\ v(B), & \text{если } v(A) = 0, \end{cases}$   
 $\alpha(A \vee B) = \begin{cases} \alpha(A), & \text{если } v(A) = 1, \\ \alpha(A) + \alpha(B), & \text{если } v(A) = 0. \end{cases}$
4.  $v(A > B) = \begin{cases} 1, & \text{если } v(A) = 0, \\ v(B), & \text{если } v(A) = 1. \end{cases}$

$$\alpha(A \supset B) = \begin{cases} \alpha(A), & \text{если } v(A)=0, \\ \alpha(A)+\alpha(B), & \text{если } v(A)=1. \end{cases}$$

$$5. v(V_n A) = \begin{cases} 1, & \text{если } v(A)=1 \text{ и } \alpha(A) \leq n, \\ 0 & \text{в противном случае,} \end{cases}$$

$$\alpha(V_n A) = \min(n, \alpha(A)).$$

Формула  $A$  истинна в модели  $\mathbb{M} = \langle v, \alpha \rangle$ , если и только если  $v(A)=1$ . Формула  $A$  общезначима, если и только если она истинна во всех моделях. Класс общезначимых формул аксиоматизируем посредством аксиом и правил вывода классической логики высказываний плюс следующих аксиом:

$$A1. \quad V_n \neg \neg A \leftrightarrow V_n A.$$

$$A2. \quad V_n A \supset A.$$

$$A3. \quad V_n A \supset V_{n+1} A.$$

$$A4-1. \quad \neg V_1 (A \& B).$$

$$A4-2. \quad V_n (A \& B) \leftrightarrow \bigwedge_{i=1}^{n-1} (V_i A \& V_{n-i} B), \quad n > 1.$$

$$A5-1. \quad V_1 (A \vee B) \leftrightarrow V_1 A.$$

$$A5-2. \quad V_n (A \vee B) \leftrightarrow \bigwedge_{i=1}^{n-1} (V_i \neg A \& V_{n-i} B) \vee \neg V_n A, \quad n > 1.$$

$$A6. \quad V_n \neg (A \& B) \leftrightarrow V_n (\neg A \vee \neg B),$$

$$A7. \quad V_n \neg (A \vee B) \leftrightarrow V_n (\neg A \& \neg B).$$

$$A8. \quad V_n (A \supset B) \leftrightarrow V_n (\neg A \vee B).$$

$$A9. \quad V_n \neg (A \supset B) \leftrightarrow V_n (A \& \neg B).$$

$$A10. \quad V_n V_m A \leftrightarrow V_n A \& V_m A.$$

$$A11-1. \quad V_n \neg V_m A \leftrightarrow \neg V_m A, \quad n \geq m.$$

$$A11-2. \quad V_n \neg V_m A \leftrightarrow V_n \neg A, \quad n < m.$$

Почти во всех языках программирования имеются логические выражения, которые используются в основном для управления ходом выполнения программ. Построенная логика может быть применена для анализа структур управления, в которых учитывается расход временных ресурсов. Эта логика может быть также

использована при построении новых языков программирования, в которых логические выражения являлись бы выражениями построенной выше логики.

Рассмотрим условный оператор `IF  $\alpha$  THEN S1 ELSE S2`, где  $\alpha$  – логическое выражение, а S1 и S2 – операторы.

В языке программирования Бейсик логические выражения относительно просты и никаких проблем с выполнением условного оператора возникнуть не может. Если же взять язык Паскаль, то ситуация резко меняется. На этом языке мы можем написать программу, вычисляющую функцию `PROVABLE(Formula)`. Данная функция принимает значение `TRUE`, если формула *Formula* классического исчисления предикатов доказуема. Если же *Formula* недоказуема, то в некоторых случаях функция принимает значение `FALSE`, а в некоторых просто неопределена. В соответствии с синтаксисом языка Паскаль мы можем записать условный оператор `IF PROVABLE(Formula) THEN S1 ELSE S2`.

В случае, если функция `PROVABLE` для данного значения переменной *Formula* неопределена, программа "зависнет". Нежелательным может оказаться и случай, когда для вычисления значения функции `PROVABLE` потребуется слишком много времени. Если в язык ввести оператор  $V_n$ , то условный оператор можно переписать в виде `IF  $V_n$ (PROVABLE(Formula)) THEN S1 ELSE S2`.

Теперь уже гарантируется, что время, затраченное на вычисление значения функции, не превзойдет величины  $n$ .

Мы рассмотрели относительно простой случай. Аналогично этому можно рассмотреть случай, когда логическое выражение имеет сложную булеву структуру. При этом может быть поставлена задача нахождения такой логически эквивалентной формулы, для

которой расход ресурсов времени минимален. Это возможно, ибо, как видно из аксиом, логически эквивалентные преобразования не всегда эквивалентны в отношении к ресурсам.

### 5.3. Правдоподобные рассуждения в системах искусственного интеллекта

Область логики, исследующая правдоподобные рассуждения, а именно, индуктивная и вероятностная логика, получила новый мощный стимул к развитию в связи с ее приложениями к искусственному интеллекту. Интерес к индуктивной и вероятностной проблематике со стороны специалистов по искусственному интеллекту вызван тем, что большинство представляющих интерес человеческих рассуждений (те, в которых получаются новые, нетривиальные, пусть и не стопроцентно надежные выводы), в сущности, являются индуктивными. Поэтому в связи с работами в области искусственного интеллекта приобретает большое значение индуктивная проблематика в самом широком смысле: извлечение общих закономерностей из разрозненных данных; поведение в условиях неопределенности, т.е. умение оперировать с неточными, вероятностными правилами и данными; выводы на основании свидетельств "за" и "против". Исследования в этой области имеют большое прикладное значение для построения экспертных систем, где обычно приходится иметь дело с большим количеством данных, неточной информацией и правилами, имеющими приближительный, интуитивный характер.

Для представления правдоподобных рассуждений применяются различные средства. Это немонотонная логика, нечеткая логика, другие логические системы, использующие численные значения степеней правдоподобия, модальная логика. Мы подробнее остановимся на методах, наиболее связанных с традицион-

ной проблематикой индуктивной логики (о других методах см., например, [17, 19]).

Индуктивные рассуждения в логике обычно разделяют на два типа: 1) открытие закономерностей, выдвижение гипотез о закономерностях, 2) исследование того, какие гипотезы лучше подтверждаются имеющимися данными. Рассуждения первого типа исследует логика открытия, второго – логика подтверждения.

Логику открытия обычно связывают с именами Ф.Бэкона и Дж.С.Милля. Так называемые методы Бэкона-Милля самим Дж.С.Миллем, правда, не рассматривались как методы открытия нового знания, а только лишь как способ установления наиболее вероятных причин некоторого явления. Например, метод единственного сходства заключается в следующем: пусть явление, причина которого нас интересует, скажем вспышка эпидемии холеры, наступало в различных ситуациях, единственное сходство которых заключается в том, что во всех случаях имело место и другое явление, например плохая система канализации. Значит, в известной мере правдоподобно, что последнее обстоятельство является причиной (достаточным условием) вспышки холеры.

Очевидно, что методы Бэкона – Милля в качестве модели реальной научной деятельности являются слишком сильным упрощением. С их помощью можно дать некоторое обоснование для наших выводов, но, чтобы автоматизировать процесс выдвижения гипотез, этих правил недостаточно. Для строгого следования этим правилам надо было бы либо рассмотреть бесконечное число случаев, либо принять суждение (которое в свою очередь не может основываться ни на чем, кроме индукции) о круге возможных причин данного явления. Действительно, почему в случае эпидемии надо интересоваться именно канализацией, а не рельефом местности, высотой над уровнем моря,

структурой занятости населения и т.д. и т.п. В XX веке упоминать о методах Милля применительно к методологии науки стало дурным тоном, а построение логики открытия казалось безнадежным предприятием. Неожиданное возрождение интереса к этой проблематике связано с исследованиями в области искусственного интеллекта. Было бы очень хорошо научить робота учиться на опыте и обобщать факты, передав ему хотя бы часть навыков, имеющихся у людей. Несомненно, это соответствовало бы созданию мини-логики открытия, имеющей дело всегда с конечной информацией, с ограниченной сферой действительности, но тем не менее моделирующей какие-то процессы человеческих индуктивных рассуждений. Вопрос заключается только в том, поддаются ли эти процессы автоматизации.

Результаты, полученные в последнее время, позволяют ответить на этот вопрос утвердительно. Созданы программы, умеющие находить закономерности в базе данных, делать обобщения, строить новые понятия и выдвигать гипотезы. Например, программы АМ и Эвриско, разработанные американским исследователем Д.Ленатом. Они содержат правила выдвижения гипотез — эвристики (обращать внимание на повторяющиеся явления и т.п.). Программа Эвриско может также и предписывать себе новые эвристики, если какой-то образ действий оказывается в большинстве случаев удачным. Такое поведение несомненно является индуктивным и часто использует добрые старые методы Милля (подробнее см. в [16]). В нашей стране также были сделаны попытки обучить ЭВМ индуктивному поведению — тем же методам Милля, другим правдоподобным выводам (см. предисловие Д.А. Пospelова к [16]).

Логика подтверждения, как и логика открытия, создабалась в качестве логики научного познания.



Создатель логики подтверждения, известный логик и философ Р.Карнап так понимал смысл индуктивных рассуждений: "Результат парадигматического образа индуктивного рассуждения относительно гипотезы  $H$ , исходящего из свидетельства  $E$ , заключается в приписывании вероятности  $H$ , а именно значения  $C(H/E)$ " [29, с.29]. (Здесь  $C(H/E)$  – степень подтверждения гипотезы  $H$  свидетельством  $E$ .)

Логика подтверждения сталкивается с рядом трудностей – парадоксов подтверждения. Наиболее известным является так называемый парадокс воронов. Обычно принимается, что гипотезу "Все  $A$  суть  $B$ " подтверждает наличие предметов, обладающих свойством  $A$  и свойством  $B$ . Например, чем больше мы в жизни встретим черных воронов, тем больше у нас оснований принять гипотезу "Все вороны черные". Таким же образом гипотезу "Все нечерные предметы не являются воронами" подтверждает наблюдение любого нечерного предмета, например белого ботинка. Но две эти гипотезы логически эквивалентны, значит, то, что подтверждает одну из них, подтверждает и другую. Таким образом, наблюдение белого ботинка подтверждает гипотезу о том, что все вороны черные, а это никак не согласуется с интуицией.

Другой довод против логики подтверждения принадлежит известному философу К.Попперу. Наибольшую степень подтверждения, если отождествлять ее с условной вероятностью, получают гипотезы с наибольшей априорной вероятностью, т.е.самые тривиальные, а не самые проверяемые (см.[18, с.216]). Степень подтверждения, таким образом, не может служить основанием для выбора научной гипотезы. Вообще, возможно такое положение дел, когда единственное имеющееся свидетельство  $E$  подтверждает  $H_1$  и не подтверждает  $H_2$ , но при этом  $C(H_2/E)$  больше  $C(H_1/E)$ . (Пусть, например,  $E$  – высказывание о том, что при

подбрасывании кости выпало четное число,  $H_1$  - что выпала шестерка,  $H_2$  - что выпала любая другая грань.  $P(H_1/E)=1/3$ ,  $P(H_2/E)=2/3$ .) Таким образом, не понятно, выражением чего является степень подтверждения и как можно делать на основании нее выводы о степени обоснованности гипотез.

Чтобы разрешить эти и подобные им трудности, были разработаны многочисленные понятия степени поддержки фактами, использующие функцию вероятностной меры, но отличающиеся от простой условной вероятности. Их определения приведены в книге Г.Кайберга [10, с.240]. Сходство функций, предложенных различными авторами, состоит в том, что степень поддержки фактами всегда пропорциональна разности условной вероятности свидетельства при данной гипотезе и его исходной вероятностью:  $P(E/H)-P(E)$  или его исходной вероятностью при отрицании гипотезы:  $P(E/H)-P(E/\neg H)$ . Кроме того, она обратно пропорциональна исходной вероятности свидетельства  $E$  - чем меньше мы ожидаем встретить такие эмпирические факты, чем меньше их исходная вероятность, тем важнее их наличие. (Так, черные вороны встречаются значительно реже, чем произвольные нечерные предметы, поэтому их вклад в подтверждение гипотезы о том, что все вороны черные, значительно выше.) Анализ различных определений понятия степени поддержки фактами Г.Кайберг заключает следующим знаменательным выводом: "Итак, в любом случае ясно, что, несмотря на полемику по частным вопросам, большинство авторов подразумевали один и тот же критерий принятия (или хотя бы серьезного рассмотрения) научной гипотезы. Теперь становится более понятным несерьезное отношение ко всем этим мерам даже со стороны их авторов. Предложенные функции выявляют интересные факторы, влияющие на беспристрастную оценку гипотез, но они оказываются ско-

рее эмпирическим отражением наших поступков, нежели логическим объяснением, почему мы должны так поступать" [10, с.241].

Таким образом, логика подтверждения немного дает для изучения человеческого познания, в частности научного познания. Ситуация радикальным образом меняется, когда логика подтверждения начинает применяться в экспертных системах. Здесь важна именно способность служить "эмпирическим отражением наших поступков", поскольку экспертная система должна уметь делать такие же выводы, какие сделал бы в том же случае человек, являющийся профессионалом в данной области. Жизненное значение приобретают описанные выше парадоксы подтверждения. Например, если система медицинской диагностики использует то понятие степени подтверждения, которое критиковал Поппер, может получиться, что она сочтет наиболее обоснованным диагноз, опровергаемый всеми имеющимися данными.

Итак, что же представляют собой экспертные системы, на которые мы все время ссылаемся, и для чего они создаются?

Экспертная система - это программное обеспечение, позволяющее вычислительной машине заменить эксперта, т.е. профессионала высокого класса, в какой-либо сложной области: например, в медицинской диагностике или геологоразведке. Экспертные системы должны решать задачи, для которых не существует известного в мельчайших подробностях алгоритма решения, а требуется профессиональный опыт, интуиция. При моделировании профессиональных знаний и методов принятия решений, характерных для эксперта в какой-либо области деятельности, выясняется, что это в основном знания, носящие не полностью надежный, приблизительный характер: сведения о том, что верно "в большинстве случаев", какие дей-

ствия "скорее всего" приведут к успеху. Кроме того, обычно эксперту приходится иметь дело с неполной информацией. Значит, и машина должна научиться рассуждать, выдвигая гипотезы и сравнивая степени их обоснованности.

Экспертная система имеет дело не только с данными о конкретных фактах, но и со знаниями. Эти знания, т.е. правила, относящиеся к сфере профессионального опыта, могут быть выражены, например, в виде правил продукции. Обычно правилам и информации о фактах приписывают характеристики надежности, выражаемые числами, например: "Если  $A$ , то - со степенью надежности  $x$  -  $B$ ". Механизм вывода должен находить логические следствия из имеющейся информации и определять степень их надежности. Пользователь экспертной системы должен иметь возможность узнать, на основании каких правил сделан тот или иной вывод, поэтому обычно в систему включается объясняющая подпрограмма, которая в ответ на запрос выдает список использованных правил.

В качестве классического образца экспертной системы рассмотрим систему МИЦИН. Задачи, которые она решает, типичны для экспертных систем. Это одна из первых систем, осуществляющих правдоподобные выводы, причем методы, положенные в ее основу, до сих пор пользуются успехом. На примере МИЦИНа можно разобрать все основные проблемы, возникающие при моделировании правдоподобных рассуждений в искусственном интеллекте.

МИЦИН - экспертная система, разработанная в 70-е годы в Станфордском университете для диагностики инфекционных заболеваний и поиска лекарственных препаратов, которые лучше всего подходят для лечения данного больного (отсюда название системы - так оканчиваются названия многих лекарств)

(см. [51, с. 37]). Знание, полученное от экспертов медиков, хранится в виде правил, например:

- ЕСЛИ: 1) микроорганизм культивируется на кровяном агаре,  
2) конкретная природа микроорганизма не достоверна,  
3) штамм этого микроорганизма грамотрицателен,  
4) по своей морфологии этот микроорганизм палочка,  
5) у пациента был тяжелый ожог,

ТО: имеются некоторые (0,4) основания считать, что микроорганизм принадлежит к бактериям вида псевдомонас.

Число 0,4 означает степень надежности заключения при истинности посылок, или степень подтверждения  $C(H/E_1 \& \dots \& E_5)$ . Таким образом, проблемы оперирования с правилами лежат отчасти в сфере логики подтверждения.

По ряду причин степень подтверждения в МИЦДне не отождествляется с условной вероятностью  $P(H/E_1 \& \dots \& E_5)$ . Одна из них заключается в том, что обнаружился факт, аналогичный парадоксам подтверждения (см. [50, с. 358]). При работе с экспертами, которых просили оценить степени надежности для правил, выяснилось, что отождествление степени надежности с условной вероятностью не устраивает экспертов по следующим интуитивным соображениям. Если симптомы  $E_1, E_2, E_3$  подтверждают диагноз  $H$  со степенью 0,7, то в соответствии с аксиомами теории вероятностей они подтверждают отрицание  $H$  со степенью 0,3. Эксперты отказывались приписать отрицанию  $H$  даже такую небольшую степень надежности на основании  $E_1, E_2$ , и  $E_3$ , поскольку эти свидетельства *опровергают*  $\neg H$ . Это заставило выбрать другую функцию для представления степени надежности,

более согласующуюся с интуицией. А именно, вводятся две функции: мера доверия  $MB$ , отражающая наличие свидетельств в пользу гипотезы, и мера недоверия  $MD$  для представления опровергающих свидетельств. Число, которое ставится в соответствие каждому правилу, - степень надежности  $CF$  - определяется как разность значений этих функций.

Вычисляются они так:

$$MB(H, E) = \begin{cases} 1, & \text{если } P(H) = 1, \\ \frac{\max(P(H/E), P(H)) - P(H)}{1 - P(H)} & \text{в противном случае,} \end{cases}$$

$$0 \leq MB(H, E) \leq 1,$$

$$MD(H, E) = \begin{cases} 1, & \text{если } P(H) = 0, \\ \frac{P(H) \cdot \min(P(H/E), P(H))}{P(H)} & \text{в противном случае,} \end{cases}$$

$$0 \leq MD(H, E) \leq 1,$$

$$CF(H, E) = MB(H, E) - MD(H, E),$$

$$-1 \leq CF(H, E) \leq 1.$$

Пусть теперь известны степени надежности для всех правил и имеются данные, позволяющие применить несколько правил. Как вычислить результирующую степень надежности (суммировать свидетельства). Это невозможно сделать, пользуясь только теоретико-вероятностными вычислениями. По  $P(H/E_1)$  и  $P(H/E_2)$  нельзя вычислить  $P(H/E_1 \& E_2)$ ; по  $P(H/E)$  и  $P(E/X)$  невозможно найти  $P(H/X)$ . Поэтому невозможно вычислить и соответствующие степени надежности, пользуясь только определениями  $MB$  и  $MD$ . Вводятся следующие приближительные методы оценки:

$$MB(H, E_1 \& E_2) = \begin{cases} 0, & \text{если } MD(H, E_1 \& E_2) = 1, \\ MB(H, E_1) + MB(H, E_2) - MB(H, E_1)MB(H, E_2) & \\ \text{в противном случае,} \end{cases}$$

$$MD(H, E_1 \& E_2) = \begin{cases} 0, & \text{если } MB(H, E_1 \& E_2) = 1, \\ MD(H, E_1) + MD(H, E_2) - MD(H, E_1)MD(H, E_2) & \\ \text{в противном случае,} \end{cases}$$

Эти методы строились как методы нахождения *приближенного* значения степени надежности, когда неизвестны необходимые для вычисления точного значения вероятности. Аналогичным образом для сложных гипотез применяются методы нечеткой логики, также дающие лишь приближенное значение:

$$MB(H_1 \& H_2, E) = \min(MB(H_1, E), MB(H_2, E)),$$

$$MD(H_1 \& H_2, E) = \max(MD(H_1, E), MD(H_2, E)),$$

$$MB(H_1 \vee H_2, E) = \max(MB(H_1, E), MB(H_2, E)),$$

$$MD(H_1 \vee H_2, E) = \min(MD(H_1, E), MD(H_2, E)).$$

Посылка в правиле может в свою очередь обладать лишь определенной степенью надежности. Тогда, чтобы вычислить степень надежности заключения, в МИЦИНе используется следующее правило:

$$MB(H, E) = MB'(H, E) \max(0, CF(E, X)),$$

$$MD(H, E) = MD'(H, E) \max(0, CF(E, X)),$$

где  $X$  — все имеющиеся данные;  $MB'$  и  $MD'$  — мера доверия (недоверия) в случае, если известно, что  $E$  истинно.

Свидетельством в пользу выбранных в МИЦИНе способов подсчета степеней надежности могла бы стать успешная работа этой экспертной системы. Но, хотя МИЦИН и дает успешные рекомендации, показано, что они не изменятся, если значения характеристик надежности поменять в пределах 0,2. Таким образом, вопрос об оправданности использования приближенных методов остается.

Между тем, если мы хотим по-прежнему использовать для моделирования правдоподобных рассуждений понятие вероятности - а это наиболее теоретически разработанное средство для представления правдоподобных рассуждений, - нам очень трудно избежать использования приближительных методов. Дело в том, что относительно вероятности логические связи не обладают свойством, которое в логике называется истинностной функциональностью, т.е. вероятность сложного утверждения нельзя вычислить по вероятностям его составных частей. Например, вероятность  $A \& B$  нельзя определить по вероятностям  $A$  и  $B$ ; надо знать еще вероятность  $A$  при условии, что  $B$  истинно, или еще какую-нибудь меру зависимости  $A$  и  $B$ .

Но, может быть, если мы будем знать вероятности исходных утверждений и степени их зависимости друг от друга, мы окажемся в состоянии вычислить вероятность любого утверждения. Для этого степень зависимости должна обладать свойством истинностной функциональности, т.е. по  $c(A, C)$  (где  $c(A, C)$  - степень зависимости  $A$  и  $C$ ) и  $c(B, C)$  должно быть возможно вычислить  $c(A \& B, C)$ . Условная вероятность, как известно, таким свойством не обладает. Доказано, что не существует функции, выражающей зависимости двух утверждений, которая обладала бы таким свойством [27]. Какова бы ни была исходная информация о вероятностях, на каком-то этапе выяснится, что мы не можем вычислить степень надежности интересующего нас утверждения, пользуясь только законами теории вероятностей.

Решать эту проблему можно по-разному. Мы коротко остановимся на байесовском подходе и теории Демпстера-Шефера, затем опишем систему, которая успешно избегает почти все перечисленные выше сложности, и приведем примеры использования теоре-



тической логики для представления правдоподобных рассуждений.

Четкое и подробное изложение байесовского подхода к построению механизма правдоподобного вывода дано в статье К.Нейлора в книге [24]. Байесовский подход получил название по теореме Байеса:

$$P(H/E_1) = \frac{P(E_1/H) P(H)}{P(E_1)}.$$

Эта теорема позволяет найти новую вероятность  $H$ , если известна ее исходная (априорная) вероятность и вероятность свидетельства  $E_1$  самого по себе и относительно  $H$ . Когда мы получаем новое свидетельство  $E_2$ , в теорему подставляется значение  $P(H/E_1)$  вместо  $P(H)$ , и так далее по мере получения все новых свидетельств. На самом деле такое использование теоремы подразумевает два достаточно сильных допущения: независимости свидетельств самих по себе и при условии  $H$ . Действительно, если в теории вероятностей

$$P(H/E_1 \& E_2) = \frac{P(E_1 \& E_2/H) P(H)}{P(E_1 \& E_2)}$$

то, находя значение  $P(H/E_1 \& E_2)$  по формуле

$$\begin{aligned} P(H/E_1 \& E_2) &= \frac{P(E_2/H) P(H/E_1)}{P(E_2)} = \\ &= \frac{P(E_1/H) P(E_2/H) P(H)}{P(E_1) P(E_2)}, \end{aligned}$$

мы неявно приняли, что

$$\begin{aligned} P(E_1 \& E_2/H) &= P(E_1/H) P(E_2/H), \\ P(E_1 \& E_2) &= P(E_1) P(E_2). \end{aligned}$$

Другое отличие байесовского подхода от традиционной статистики – более смелое использование субъективных оценок вероятностей (когда не хватает статистических данных) и принципа индифферентности: если нет оснований считать одно событие более вероятным, чем другое, считают эти два события равновероятными. Сторонники байесовского подхода обосновывают эти методы, показывая, что после многократного применения теоремы Байеса влияние всех этих допущений на результат становится минимальным. Полученные вероятности могут быть не идеально точными, но порядок их соотношения определяется достаточно надежно, а это, как правило, и нужно знать для принятия решения.

Степени надежности при данном подходе обычно (хотя и не всегда) отождествляются с условными вероятностями, что дает следующие поводы для критики: во-первых, не различаются вероятности, полученные в результате применения принципа индифферентности и основанные на каких-то данных, т.е. "основанные на незнании" и "основанные на знании"; во-вторых, не различаются свидетельства "за" и свидетельства "против" (как уже говорилось, эксперты при разработке МИЦИНа отказывались приписывать положительное значение вероятности на основании опровергающего свидетельства). Но эта критика не является серьезным аргументом против байесовского подхода. Зная вероятности, мы можем построить сколь угодно соответствующую интуиции функцию поддержки фактами, а то и две, как в МИЦИНе. Так делается, например, в ПРОСПЕКТОРе – экспертной системе для геологоразведки. Эксперты и пользователи имеют дело с характеристиками надежности, аналогичными МИЦИНовским, только изменяющимися от -5 до 5, а механизм вывода работает с вероятностями. Нас

сейчас интересует именно механизм вывода, т.е. логика.

Серьезным аргументом является все-таки зависимость от допущений. Доказано, что, если к "стандартным" допущениям

1) гипотезы  $H_1, \dots, H_n$ ,  $n > 2$  - взаимоисключающие и исчерпывающие,

2) для любых  $E_1, E_2, H_i$  ( $1 \leq i \leq n$ )  $P(E_1 \& E_2 / H_i) = P(E_1 / H_i) P(E_2 / H_i)$

добавить еще одно:

3) для любых  $E_1, E_2, H_i$  ( $1 \leq i \leq n$ )  $P(E_1 \& E_2 / \neg H_i) = P(E_1 / \neg H_i) P(E_2 / \neg H_i)$ ,

из совокупности утверждений 1. - 3 следует, что для каждого  $H_i$  существует только одно  $E_j$ , изменяющее его вероятность. Ясно, что такая ситуация просто не может существовать в нормальной экспертной системе. Однако допущение 3 иногда принимают. Если же для области, в которой будет применяться экспертная система, существует достаточная статистическая информация, в том числе о взаимной зависимости симптомов относительно каждого диагноза и т. д., теорема Байеса будет давать наилучшие результаты.

Теория Демпстера-Шефера отличается от байесовского подхода тем, что, во-первых, функции уверенности (вероятности) в этой теории определены не для всех гипотез, а только для тех, относительно которых имеются какие-либо свидетельства. Это позволяет, в частности, избежать применения принципа индифферентности. Во-вторых, математический аппарат несколько отличается от аппарата теории вероятностей (см., например, [19]). Для суммирования свидетельств применяется правило Демпстера, что аналогично принятию допущения условной независимости. Наконец, Шефер предложил определять для

меры уверенности верхнюю и нижнюю границы, а не точечное значение. Интервал, который дает теория Демпстера-Шефера, включается в тот, который был бы получен в классической теории вероятностей.

Идея использования интервалов нашла удачное применение в системе ИНФЕРНО Дж.Куинлена [48]. Это система, осуществляющая правдоподобные выводы в условиях не полностью надежной информации. В отличие от МИЦИНа, ПРОСПЕКТОРА и многих аналогичных систем в ИНФЕРНО не принимаются никакие допущения относительно совместных распределений вероятностей высказываний (условная независимость и т.п.) и не вводятся приближенные методы оценки.

Заключения о том, в каких границах лежит вероятность высказываний, являются доказуемыми в теории вероятностей следствиями исходной информации о вероятностях других высказываний. Поэтому ИНФЕРНО имеет дело не с точечными, а с интервальными значениями вероятностей.

Ранее подобный подход применялся при создании системы ВАНД Ф.Хэйес-Рута и некоторых других систем. В общем случае известно, что вероятность  $A$  лежит в интервале  $[s(A), p(A)]$ , т.е. вероятность  $A$  равна по меньшей мере  $s(A)$ , а вероятность  $\neg A$  равна по меньшей мере  $1 - p(A)$ . Преимущества данного подхода очевидны: в отличие от байесовского он позволяет рассматривать отдельно свидетельства за и против  $A$ ; в отличие от МИЦИНа и аналогичных систем построен на прочном фундаменте теории вероятностей. Хотя результат вывода утверждений о границах вероятностей менее определенный, чем, например, в МИЦИНе, он зато более надежный. Если между высказываниями действительно имеют место отношения независимости или несовместимости, в ИНФЕРНО можно учесть это в явном виде и использовать при вычис-

лении вероятностей. В этом случае результат получается не только более определенный, но и полностью надежный. В систему можно ввести информацию о вероятностях любого множества высказываний и исследовать следствия для всей системы в целом: выявить несогласованности, уточнить границы вероятности для любого высказывания (разделения на свидетельства и гипотезы в ИНФЕРНО нет).

Высказывание  $A$  характеризуют две величины:  $t(A)$  и  $f(A)$ , где  $P(A) \geq t(A)$  и  $P(\neg A) \geq f(A)$ . Каждое высказывание имеет тривиальные границы:  $t(A) = 0$  и  $f(A) = 0$ . С получением новой информации интервал, в котором может лежать вероятность  $A$ , становится все уже. Это выражается в возрастании значений  $t(A)$  и  $f(A)$ . Информация о высказывании  $A$  согласована, если  $t(A) + f(A) \leq 1$ .

В ИНФЕРНО используются отношения типа

$A$  свидетельствует в пользу  $S$  со степенью  $X$ :  
 $P(S/A) \geq X$ ;

$A$  есть конъюнкция независимых  $\{S_1, \dots, S_n\}$ :  
 $A \cong \&_i S_i$  и для всех  $i \neq j$   $P(S_i \& S_j) = P(S_i) P(S_j)$ ,  
 и т.п.

Допустим, известны значения для границ какого-либо высказывания. Следует рассмотреть все отношения в которые оно вовлечено, и, возможно, увеличить значения границ других высказываний таким образом, чтобы выполнялись правила перераспределения вероятностей. Они представляют собой неравенства, которые интерпретируются как правила продукции:

ЕСЛИ: прежнее значение границы в левой части  
 меньше значения в правой части

ТО: граница должна быть увеличена до этого  
 нового значения.

Например,  $A$  свидетельствует в пользу  $S$  со степенью  $X$ :

$$(1.1) \quad t(S) \geq t(A)X$$

$$(1.2) \quad f(A) \geq 1 - [1 - f(S)]/X.$$

К сожалению, если правила выводимы из интерпретации отношений, то обратное неверно. В этом смысле логика ИНФЕРНО неполна. Значения для границ, выведенные по правилам, всегда будут правильными, но иногда более слабыми, чем это следует из интерпретации отношений.

Выше речь шла в основном о практически применяемых методах, многие из которых не имеют достаточного теоретического обоснования и носят характер "к случаю". Сейчас мы остановимся на подходах к представлению правдоподобных рассуждений в искусственном интеллекте с точки зрения теоретической логики.

В статье Н.Нильсона [43] предлагается вероятностная логика, позволяющая решить проблему вероятностного следования: определить вероятность (или границы, в которых лежит вероятность) одних предложений по вероятностям других. Мы опишем суть подхода Н.Нильсона в общем виде (изложение на примерах см. в [19]). Пусть дано множество предложений  $\{S_1, \dots, S_n\}$ . Этим предложениям могут быть приписаны истинностные значения  $k$  ( $k \leq 2^n$ ) различными способами (имеются в виду только согласованные приписывания). Каждое такое приписывание можно отождествить с возможным миром (точнее, классом эквивалентности возможных миров относительно этого приписывания).

Вместо того, чтобы говорить о "вероятности того, что действительный мир принадлежит данному классу эквивалентности", будем говорить о "вероятности этого приписывания". В сумме вероятности приписываний должны давать 1.

Очевидно, что вероятность некоторого утвержде-

ния равна сумме вероятностей приписываний, при которых оно истинно.

Пусть известны вероятности всех предложений, кроме  $S_n$ . Обозначим их  $\pi_1, \dots, \pi_{n-1}$ . Для любой разрешимой логики может быть построена матрица, задающая все возможные согласованные приписывания предложениям  $S_1, \dots, S_n$ . Назовем эту матрицу  $V$ . Каждый столбец этой матрицы соответствует одному из возможных приписываний истинностных значений для данных предложений;  $v_{ij}$  равно 1, если при  $j$ -м приписывании  $S_i$  получает значение "истина", и равно 0 в противном случае:

$$V = \begin{vmatrix} v_{11} & v_{12} & \dots & v_{1k} \\ v_{21} & v_{22} & \dots & v_{2k} \\ \dots & \dots & \dots & \dots \\ v_{n1} & v_{n2} & \dots & v_{nk} \end{vmatrix}.$$

Пусть задан  $k$ -мерный вектор-столбец  $P$ , его  $j$ -я компонента есть  $p_j$  - вероятность  $j$ -го предложения. Вероятности предложений задаются вектором  $\Pi$ :

$$\Pi = VP.$$

Таким образом, вероятность  $\pi_i$  предложения  $S_i$  ( $i$ -я компонента  $\Pi$ ) равна сумме вероятностей тех приписываний, в которых  $S_i$  истинно.

Пусть теперь известны вероятности  $S_1, \dots, S_{n-1}$ , требуется определить вероятность  $S_n$ . Введем  $T$  (тавтологию) как первый элемент множества предложений. Первая строка  $V$  будет состоять из одних единиц. Последняя строка  $V$  пусть содержит истинностные значения  $S_n$ . Значения вероятностей по условию даны для всех предложений, кроме  $S_n$ ; вероятность тавтологии равна 1. Рассмотрим матрицу  $V'$  (без последней строки) и вектор  $\Pi'$  (без вероятности  $S_n$ ). Решив уравнение  $\Pi' = V'P$  и найдя  $P$ , лег-

ко определить вероятность  $S_n$  - она равна произведению последней строки  $V$  на  $P$ . Обычно существует много возможных решений для  $P$ , и для вероятности  $S_n$  можно указать лишь границы, в которых она должна лежать.

В статье [43] дается несколько методов, которые упрощают решение получившейся системы уравнений. Но все они становятся непригодными, когда одновременно рассматривается большое число предложений. Тогда вычисления становятся практически невыполнимыми. Для этого случая (больших матриц) также дается несколько вариантов приближенных вычислений, точность которых зависит от имеющихся вычислительных ресурсов.

Первая работа Н.Нильсона, содержащая описание его вероятностной логики, появилась в 1984 г., а в 1985 г. А.Банди, другой известный специалист по искусственному интеллекту, предложил свое решение проблем, связанных с вероятностными рассуждениями в экспертных системах [27]. Оно заключается в том, чтобы сделать связи вероятностной логики истинностно-функциональными. Для этого в качестве меры надежности формулы А.Банди вводит не численные значения вероятностей, а множества точек или элементарных событий, для которых это предложение истинно. Такие множества он назвал инциденциями, или степенями распространенности предложений.

Пусть  $W$  - множество точек (элементарных событий), исчерпывающих и взаимоисключающих. По вычислительным причинам рассматривается конечное множество  $W$ ;  $i(A)$  - (инциденция  $A$ ) - подмножество  $W$ , включающее все те точки, в которых  $A$  истинно. Очевидно, что

$$i(T) = W,$$

$$i(F) = \emptyset,$$



$$\begin{aligned}
t(\neg A) &= W \setminus t(A), \\
t(A \& B) &= t(A) \cap t(B), \\
t(A \vee B) &= t(A) \cup t(B), \\
t(\forall x A x) &\subseteq t(Az) \subseteq t(\exists x A x),
\end{aligned}$$

где  $z$  – терм, не содержащий переменных, свободных в  $Ax$ ;  $T$  и  $F$  – соответственно тавтология и противоречие. Связки, таким образом, истинностно-функциональны, кванторы – нет. Для кванторов даются верхняя и нижняя границы.

Вероятность  $A$  –  $P(A)$  – равна сумме вероятностей точек, составляющих  $t(A)$ . В статье [27] рассматривается простейший случай, когда все точки равновероятны. Тогда вероятность  $A$  равна отношению числа точек в  $t(A)$  к числу точек в  $W$ .

Как мы видели на примере МИЦИНа, механизм вывода в условиях неопределенности должен включать правило определения степени надежности  $B$  по степеням надежности  $A$  и  $A > B$ . А.Банди называет это свойство по аналогии с истинностной функциональностью доказательственной функциональностью. Оно имеет место в ослабленном виде, т.е. в результате нескольких применений *modus ponens* мы получаем различные значения для нижней границы  $B$ , затем применяем теоретико-множественное свойство

$$l_1 \subseteq t(B) \ \& \ l_2 \subseteq t(B) \Rightarrow l_1 \cup l_2 \subseteq t(B),$$

где  $t(B)$  – инциденция  $B$ ;  $l_1$  и  $l_2$  – нижние границы. Этот способ напоминает суммирование свидетельств в МИЦИНе, но в отличие от мициновского правила, имеющего характер *ad hoc*, основан на теории множеств. Поскольку связки являются истинностно-функциональными, нижняя граница определяется гораздо точнее, чем в логике, использующей вероятности.

В статье [27] дается механизм вывода логики инциденций, имеющий дело с верхними и нижними грани-

цами. Он является полным для пропозициональной логики. Этот алгоритм реализован автором на Прологе. Кроме того, дан механизм выявления несогласованности во введенных данных. Метод, предложенный Нильсоном, требует по сравнению с данным гораздо меньше вычислений.

Правила вывода позволяют переходить от одного приписывания, скажем,  $F$ , к другому —  $G$ , которое дает более узкие границы для  $\sup(A)$  и  $\inf(A)$  — наибольшего и наименьшего значений инцидентности  $A$ . Приведем пример правила для конъюнкции:

$$AND1.\sup_G = [\sup_F(A \& B) \cup W \setminus \inf_F(B)] \cap \sup_F(A).$$

Модальная логика все чаще используется в искусственном интеллекте, в том числе и для представления правдоподобных рассуждений. Прежде всего это относится к тем случаям, когда достаточно качественного понятия вероятности: "вероятно", "более вероятно", "менее вероятно" и т.п. Так, например, Дж.Халперн и М.Рэбин [33] построили логику с оператором  $L$ :  $LA$  читается примерно как " $A$  является правдоподобной относительно имеющегося знания ги — потезой". Количественных оценок степеней правдоподобия нет, но их различие удастся передать за счет итерации операторов:  $LLA$  слабее, чем  $LA$ , и вообще,  $L^n A$ , где  $n$  — число операторов перед  $A$ , слабее, чем  $L^m A$ , если  $m < n$ . Для этой логики дается разрешающая процедура, позволяющая установить, следует ли некоторое утверждение из множества других утверждений, или является ли некоторое утверждение необходимо истинным. Правила продукции примут вид "если  $E$ , то  $L^n H$ ". Непосредственной вероятностной интерпретации оператору  $L$  дать, видимо, нельзя. В логике Халперна и Рэбина такую интерпретацию (например, "вероятность больше  $1/2$ ") можно дать только

более сложной формуле, включающей оператор необходимости.

Использовать модальную логику можно не только в случае качественных понятий. Например, при проверке введенных утверждений о вероятности на согласованность естественно воспользоваться логикой с операторами "вероятность... больше или равна  $x$ " или "вероятность... лежит в интервале от  $x$  до  $y$ ".

Достаточно серьезной проблемой, которой мы не будем касаться в этом параграфе, является приписывание исходных вероятностей (инцидентий, степеней правдоподобия и т.д.) утверждениям и правилам. Эта проблема обсуждается, например, в [24, 27, 33].

## ЛИТЕРАТУРА

1. *Андерсон Р.* Доказательство правильности программ. М.: Мир, 1982. 287 с.
2. *Бежаншвили М.Н.* Некоторые эпистемические пропозициональные системы // Методы логических исследований. Тбилиси: Мецниереба, 1987. С.27-38.
3. *Болтянский В.Г., Ефремович В.А.* Наглядная топология. М.: Наука, 1982. 207 с.
4. *Бочаров В.А.* Булева алгебра в терминах силлогистики // Логические исследования. М.: ИФАН СССР, 1983. С.32-42.
5. *Ван Хао.* На пути к механической математике // Кибернетический сборник. М.: Прогресс, 1962. № 5. С.23-51.
6. *Генцен Г.* Исследования логических выводов // Математическая теория логического вывода. М.: Наука, 1967. С.352-390.
7. *Герасимова И.А.* Логический анализ рассуждений на основании личностных знаний // Синтаксические и семантические исследования неэкстенциональных логик. М.: ИФАН СССР, 1989. С.309-318.
8. *Грис Д.* Наука программирования. М.: Мир, 1984. 289 с.
9. *Зыков А.А.* Основы теории графов. Москва : Наука, 1967. 381 с.
10. *Кайберг Г.* Вероятность и индуктивная логика. М.: Прогресс, 1978. 375 с.
11. *Кудрявцев Л.Д.* Современная математика и ее преподавание. М.: Высшая школа, 1985. 170 с.
12. *Кузичев А.С.* Диаграммы Венна. М.: Мир, 1968. 180 с.
13. *Кэррол Л.* История с узелками. М.: Прогресс, 1973. 176 с.

14. Маслов С.Ю. Теория дедуктивных систем и ее применение. М.: Радио и связь. 1986. 135 с.
15. Минский М. Вычисления и автоматы. М.: Мир, 1971. 364 с.
16. Мичи Д., Джонстон Р. Компьютер - творец. М.: Мир, 1987. 254 с.
17. Нечеткие множества и теория возможностей. Последние достижения. М.: Радио и связь, 1986. 406 с.
18. Поппер К. Логика и рост научного знания. М.: Прогресс, 1983. 606 с.
19. Представление и использование знаний. М.: Мир, 1989. 220 с.
20. Самоорганизующиеся системы. М.: Мир, 1964. 259 с.
21. Смирнов В.А. Логические методы научного познания. М.: Наука, 1987. 220 с.
22. Форсайт Р. Паскаль для всех. М.: Мир, 1986. 321 с.
23. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. М.: Наука, 1983, 193 с.
24. Экспертные системы. Принципы работы и примеры. М.: Радио и связь, 1987. 223 с.
25. Apt K. Ten years of Hoare's logic // ACM TOPLAS. 1981. Vol.3. P.431-483.
26. Beth E. Foundations of mathematics. Amsterdam: Reidel, 1959. 357 p.
27. Bundy A. Incident calculus: a mechanism for probabilistic reasoning // J.of Automated Reasoning. 1985. N 1. P.263-283.
28. Burstall R. Program proving as hand simulation with little induction // Inform. Proc. 1974. Vol.74. P.308-312.
29. Carnap R. Inductive logic and rational decisions // Studies in inductive logic and

- probability. Berkeley: Calif. Univ.Press, 1971. Vol.1. P.5-31.
30. *Fagin R., Halpern J.* Belief, awareness and limited reasoning // Proc. 9th Int. Joint Conf. on Artificial Intelligence. 1985. P. 491-501.
  31. *Gödel K.* Die Vollständigkeit der Axiome des logischen Funktionenkalküls//Monatsh.Math.Phys. 1930. Bd.37. S.349-360.
  32. *Gödel K.* Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme // Ibid. 1931. Bd.38. S.173-198.
  33. *Halpern J., Rabin M.* A logic to reason about likelihood // Artificial Intelligence. 1987. Vol.32. P.379-405.
  34. *Hasenjaeger G.* Eine Bemerkung zu Henkins Beweis für Vollständigkeit des Prädikaten Kalküls der ersten Stufe // J.of Symbolic Logic. 1953. Vol. 18. P.42-48.
  35. *Henkin L.* The completeness of the first-order functional calculus // Ibid. 1949.Vol.14.P.159-166.
  36. *Hintikka J.* Form and content in quantification theory // Acta Philosophia Fennica.1955. Vol.8. P.11-55.
  37. *Hintikka J.* Knowledge and belief: An introduction to the logic of the two notions. Ithaca: Acad. Press. 1962. 450 p.
  38. *Hintikka J.* Notes on the quantification theory // Comment. Phys.-Math. Soc.Sci.Fennica. 1955. Vol.17. P.1-13.
  39. *Jaskowski A.* On the rules of suppositions in formal // Studia Logica.1934. Vol.1. P.37-90.
  40. *Lewis D.* Convention. A philosophical study. Harward: Univ.Press. 1969. 175 p.
  41. *Manna Z., Pnueli A.* Verification of concurrent programs:the temporal framework // The Correct-

- ness Problem in Computer Sci. London: Acad. Press. 1981. P.215-283.
42. *Moor R.C.* Semantical considerations on nonmonotonic logic // Artificial Intelligence. 1985. Vol.25. P.56-89.
  43. *Nilsson N.J.* Probabilistic logic // Ibid. 1986. Vol.28. P.71-87.
  44. *Pruell A.* The temporal logic of programs // Proc. 18th IEEE symp. on the foundations of computer science. San-Francisco, 1977. P.46-57.
  45. Proceedings of the conference on theoretical aspects of reasoning about knowledge. California: Morgan Kaufmann, 1986.
  46. Proceedings of the 3rd ACM conference on principle of distributed computing. 1984 .
  47. Proceedings of the 4th ACM symposium on principle of distributed computing. 1985.
  48. *Quinlan J.R.* Ihferno: a cautious approach to uncertain inference // Computer J.1983. Vol.26. P.255-269.
  49. *Rasiowa H., Sikorsky R.* A proof of completeness of theorem of Gödel // Fundam. Math. 1951. Vol. 37. P.193-200.
  50. *Shortliffe E.H., Buchanan B.G.* A model of inexact reasoning in medicine// Mathematical Biosciences. 1975. Vol.23. P.351-379.
  51. *Shortliffe E.H.* Computer-based medical consultations: MYCIN. N.Y.: Elsevier, 1976. Vol.22. 264 p.
  52. *Venn J.* On the diagrammatic and mechanical representations of propositions and reasoning //Philosophical Magazine and Journal of Science 1980. Vol.10. P.5.

## ОБ АВТОРАХ

*Алешина Наталья Альбертовна*, кандидат философских наук, научный сотрудник Института философии АН СССР.

*Анисов Александр Михайлович*, кандидат философских наук, научный сотрудник Института философии АН СССР.

*Быстров Петр Иванович*, кандидат философских наук, научный сотрудник Института философии АН СССР.

*Герасимова Ирина Алексеевна*, кандидат философских наук, научный сотрудник Института философии АН СССР.

*Месъков Валерий Сергеевич*, кандидат философских наук, главный методист по философии ГУПОН Гособразования СССР, доцент кафедры логики МГУ им.М.В.Ломоносова.

*Нерейцова Николай Николаевич*, доктор физико-математических наук, заведующий отделом Физико-технического института Уральского научного центра.

*Смирнов Владимир Александрович*, доктор философских наук, заведующий отделом Института философии АН СССР, профессор кафедры логики МГУ им.М.В.Ломоносова.

*Стеблецова Вера Николаевна*, кандидат философских наук, научный сотрудник Института философии АН СССР.

*Шалак Владимир Иванович*, кандидат философских наук, научный сотрудник Института программных систем АН СССР.



## ОГЛАВЛЕНИЕ

Предисловие .....	3
Глава 1. Классическая первопорядковая логика.....	9
1.1. Что такое правильное рассуждение? Посыл- ки и заключения.....	9
1.2. Формальный характер рассуждений. Предло- жения и высказывания.....	12
1.3. Язык первопорядковой логики предикатов..	16
1.4. Интерпретация языка логики предикатов...	23
1.5. Способы формализации логического следо- вания и общезначимости.....	27
1.6. Силлогистика и так называемые естествен- ные рассуждения.....	56
Глава 2. Логика в программировании.....	60
2.1. Теория алгоритмов и логика – родители программирования.....	60
2.2. Попытки применения логики в программиро- вании в 50-х – 70-х годах.....	64
2.3. Логика описания программ.....	70
2.4. Логическое описание программирования....	77
2.5. Логика как инструмент анализа понятий программирования.....	95
Глава 3. О доказательствах правильности программ.....	99
3.1. Зачем нужно доказывать правильность программ.....	99
3.2. Понятие правильной программы: первое знакомство.....	104
3.3. Доказательство правильности программ: метод математической индукции.....	121
3.4. Доказательство правильности программ и доказательства теорем на компьютерах....	133
3.5. Доказательство правильности программ с помощью временной логики.....	136

Глава 4. Поиск доказательства.....	158
4.1. Возможны ли логические процедуры поиска выводов и доказательств?.....	158
4.2. Поиск натурального вывода в режиме диалога.....	166
Глава 5. Приложение логики к проблемам искусственного интеллекта.....	183
5.1. Эпистемическая логика и распределенные системы.....	183
5.2. Рассуждения при ограниченных ресурсах. Логика с ресурсами.....	203
5.3. Правдоподобные рассуждения в системах искусственного интеллекта.....	209
Литература.....	231
Об авторах.....	235

# CONTENTS

Preface.....	3
Chapter 1. Classical first-order logic.....	9
1.1. What is correct reasoning? Assumptions and conclusions.....	9
1.2. Formal character of reasoning. Propositions and sentences.....	12
1.3. Language of the first-order logic of predicates.....	16
1.4. Interpretations of the language of first-order logic of predicates.....	23
1.5. Methods of formalization of logical entailment and validity.....	27
1.6. Syllogistics and so called natural reasoning.....	56
Chapter 2. Logic in programming.....	60
2.1. The theory of algorithms and logic - the parents of programming.....	60
2.2. Attempts at applying logic in programming in 50-70s.....	64
2.3. Logics of program descriptions.....	70
2.4. Logical description of programming.....	77
2.5. Logic as an instrument of analysis of programming concepts .....	95
Chapter 3. Verification of programmes.....	99
3.1. Why we have to prove the correctness of programs.....	99
3.2. The notion of correct program: the first acquaintance.....	104
3.3. Verification of programmes: method of mathematical induction.....	121
3.4. Verification of programmes and computer- aided theorem proving .....	133

3.5. Verification of programmes with the help of temporal logic.....	136
Chapter 4. The proof search.....	158
4.1. Are the logical procedures of proof search possible?.....	158
4.2. The search of natural deduction proof in the dialogue systems.....	166
Chapter 5. Applications of logic in artificial intelligence.....	183
5.1. Epistemic logic and distributed systems.	183
5.2. Reasoning with bounded resources. Logic with resources.....	203
5.3. Plausible reasoning in the systems of artificial intelligence.....	209
Bibliography.....	231
About authors.....	235

Научное издание

**Алешина** Наталия Альбертовна

**Анисов** Александр Михайлович

**Быстров** Петр Иванович и др.

## **ЛОГИКА И КОМПЬЮТЕР**

### **Моделирование рассуждений и проверка правильности программ**

Зав. редакцией  
Н.Н. Прокофьева

Редактор издательства  
Л.Е. Кононенко

Художник  
В.А. Смирнов

Художественный редактор  
В.Ю. Яковлев

Технический редактор  
Х.Х. Акмаева

Утверждено к печати  
редакцией серии  
"Кибернетика –  
неограниченные возможности  
и возможные ограничения"

ИБ № 40073

Подписано к печати 04.07.90  
Т – 00452. Формат 84 × 108<sup>1</sup>/<sub>32</sub>  
Бумага типографская № 2

Печать офсетная  
Усл.печ. л. 12,6 Усл. кр. отт. 12,7  
Уч.-изд. л. 9,8  
Тираж 50000 экз. Тип.зак. 294  
Цена 65 коп.

Ордена Трудового Красного Знамени  
издательство "Наука"  
117864, ГСП-7, Москва В-485,  
Профсоюзная ул., 90

4-я типография издательства "Наука"  
630077, Новосибирск, 77,  
Станиславского, 25

Оригинал-макет изготовлен на персон-  
альном компьютере в Институте  
философии АН СССР

65 коп.