Московский государственный технический университет имени Н.Э. Баумана

Факультет «Фундаментальные науки»
Кафедра «Математическое моделирование»

А.Н. Канатников

ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

Конспект лекций

Для студентов кафедры ИУ9

Москва
2020

1. АЛГЕБРА ВЫСКАЗЫВАНИЙ

1.1. Введение

Феномен математического доказательства. Теоремы и аксиомы. Аксиоматический метод и его развитие. Понятие исчисления. Цели формализации в математике.

Математическая логика — математическая дисциплина о законах правильного мышления. Можно выделить два первоисточника этой дисциплины. Первый — аристотелевская логика, представляющая собой науку о правильном построении суждений, о структуре суждений и о построении умозаключений, т.е. последовательности суждений, в которой одни суждения необходимо вытекают из других. Второй первоисточник — феномен математического доказательства. Математика отличается от других естественнонаучных дисциплин тем, что все ее факты требуют доказательств в виде некоторых умозаключений. Ссылки на опыт и наблюдения в математике, как правило, в расчет не берутся. Можно, конечно, не согласиться с тем, что доказательство в виде цепи умозаключений — прерогатива только математики. И можно задаваться вопросом, почему математика так устроена. Но это уже отдельные вопросы.

Математическое доказательство строится как цепь некоторых умозаключений, в которой одни суждения выводятся из других. Разумеется, в каждой такой цепи должны быть первичные суждения, обосновывать которые не требуется. В результате принцип математического доказательства привел к тому, что мы называем аксиоматическим методом. Впервые аксиоматический метод как научный принцип мы наблюдаем в „Началах“ Евклида, в которой все положения сформулированы как теоремы, доказываемые с помощью некоторого набора первичных утверждений — аксиом.

Первоначально считалось, что аксиомами могут быть такие утверждения, истинность которых очевидна. Однако уже Евклиду пришлось столкнуться с тем, что для достижения требуемого результата (обоснования всех имеющихся к тому времени в геометрии фактов) одних очевидных истин недостаточно. Споры вызвал знаменитый пятый постулат Евклида, который в современной формулировке гласит, что через точку вне прямой можно провести, и притом только одну, прямую, не пересекающую данную. Неочевидный характер этого постулата связан с непосредственным использованием понятия бесконечного. Если постулат о том, что через две точки можно провести прямую, в определенном смысле можно проверить на практике, то пятый постулат нет, так как невозможно обеспечить неограниченное продолжение прямой.

Именно исследования пятого постулата (а точнее, борьба с пятым постулатом) привели к созданию неевклидовой геометрии, с одной стороны, и к новому взгляду на понятие аксиомы, с другой. Новый взгляд на понятие аксиомы можно выразить следующим образом: очевидность аксиомы не обязательна, но важно, чтобы из набора выбранных аксиом не вытекало каких-либо противоречий. Неевклидова геометрия как раз и была создана как набор логических выводов из системы аксиом, в которых вместо пятого постулата было взято его отрицание. Пытаясь прийти к противоречию, Лобачевский старался вывести пятый постулат из других аксиом, а взамен получил развитую логически непротиворечивую теорию.

Создание неевклидовой геометрии — это первый поворотный момент в развитии аксиоматического метода (в литературе иногда именно это событие считают моментом создания аксиоматического метода). Следующий поворотный момент на рубеже 19–20 вв. вызван открытием противоречий (их называют антиномиями) в теории множеств. Опять ситуация возникла из-за произвольного использования понятия бесконечности. Интуитивно ясно, что противоречия возникают, когда о том или ином понятии судят, еще не определив его толком. Парадокс Рассела состоит в рассмотрении множества всех множеств, не принадлежащих самому себе. Но

на бытовом уровне должно быть ясно, что для того, чтобы говорить о принадлежности какого-либо объекта множеству, этот объект должен быть однозначно определен. Значит, говорить о принадлежности множества самому себе неправомерно. Но тогда что такое множество, как дать формально строгое определение этого понятия, исключающее подобные парадоксы?

Беда однако не приходит одна. Найденные парадоксы касаются не только множеств, но и некоторых определений. Парадокс Рассела можно квалифицировать как „плохое определение“, при котором объект определяется рекурсивно, через себя самого. Это говорит о том, что и приемы конструирования логических суждений, без которых математика невозможна, также должны быть подвергнуты точному математическому исследованию. Основой для такого исследования явилась символика, подобная алгебраической, в которой математические суждения и связи между ними записываются в виде математических формул. Эти формулы подчиняются определенным правилам преобразований и соединений, а логика может быть превращена в некую алгебру, подобную алгебре арифметических выражений или уравнений.

Эти соображения привели к дальнейшему уточнению понятия „аксиоматический метод“. Аксиоматический метод — это формальное описание математический теорий, при котором все теоремы теории являются логическим следствием аксиом, при этом содержательный смысл аксиом не является существенным: лишь бы такие следствия не были взаимоисключающими. Поскольку насчет правил логического вывода возникли разногласия, сами эти правила тоже должны быть формализованы. С введением символики в логику изложение математической теории превращается в некоторый набор исходных формул, т.е. неких буквенных конструкций, сконструированных по определенным правилам, которых можно преобразовать с помощью фиксированного набора правил преобразования. Содержательный смысл исходных формул, их преобразования и получаемых при этом новых формул игнорируется.

Подобная конструкция называется формальным исчислением. Характерные ее черты:

• фиксированный набор используемых символов, называемый *алфавитом теории*;

• определенный набор правил составления формул из алфавита (совокупность всех правильно составленных формул называется *языком теории*);

• выделенный набор правильно составленных формул, объявляемых истинными (это *система аксиом теории*);

• строго определенный набор правил преобразования формул, с помощью которых из уже имеющихся истинных формул получаются новые истинные формулы — теоремы теории.

Построение исчислений не является высшей целью развития математики. На это указывает положение дел в таких классических областях математики, как дифференциальное и интегральное исчисления, геометрия. Эти разделы, несмотря на их многолетнюю историю, по-прежнему излагаются на содержательном, а не формальном, уровне. Парадоксы теории множеств — до сих пор нерешенная проблема. Однако это не мешает с успехом использовать теорию множеств в самых различных математических дисциплинах.

Дело в том, что формализация математической теории — это метод математического исследования этой теории, который позволяет провести анализ исходных положений теории, выявить скрытые допущения, используемые при построении теории и т.п. Собственно, для любого формального исчисления ставятся три вопроса: непротиворечивость, полнота и разрешимость. Под непротиворечивостью понимается, что в этой теории нет взаимно противоположных теорем A и $\neg A$. Под полнотой понимается то, что любое утверждение A, которое может быть сформулировано на языке теории может быть либо доказано, либо опровергнуто (т.е. доказано противоположное утверждение $\neg A$). Наконец, под разрешимостью понимается существование алгоритма (т.е. некоторого конечного набора процедур), позволяющего для любого утверждения теории проверить, верно оно или нет.

При содержательном анализе формальной теории приходится выходить за ее рамки. Поэтому вокруг формальной теории возникает другая теория: „учение о формальной теории“, которую называют ***метатеорией***. Метатеория формирует свой язык, расширяющий язык формальной теории, который называется ***метаязыком***.

Непротиворечивость конкретной теории можно установить, построив модель этой теории в рамках другой теории. В результате мы придем к заключению: если та, другая теория непротивречива, то и наша теория непротиворечива. В этом случае говорят об относительной непротиворечивости. Абсолютная непротиворечивость означает, что данная теория непротиворечива независимо от того, являются непротиворечивыми другие теории или нет.

В 20 в. практически все математические теории получили свои модели в рамках арифметики. Так, все числовые системы были построены на базе множества натуральных чисел. Все геометрические теории имеют естественную интерпретацию на базе множества действительных чисел. Связь алгебры и геометрии общеизвестна. Однако выяснилось, что непротиворечивость формальной арифметики нельзя доказать средствами самой арифметики и что формальная арифметика не обладает ни свойством полноты, ни свойством разрешимости. Выходит, в математике, как и в других науках абсолютной истины нет.

1.2. Алгебра логики

Высказывания и их истинностные значения. Понятие логической операции. Логические операции $\vee$, $\wedge$, $\to$, $\sim$, $\neg$. Пропозициональные формулы: пропозициональные переменные и шаг индукции ($X \vee Y$, $X \wedge Y$, $X \to Y$, $\neg X$). Истинностные функции и пропозициональные формулы.

Напомню, что под высказыванием понимается любое предложение, относительно которого можно сказать истинно оно или нет, т.е. утвердительное предложение. Строго говоря, сказанное не следует рассматривать как настоящее математическое определение — лишь как указание на те реальные объекты, которые могут служить иллюстрацией к точной математической теории.

Формирование предложений в естественном языке указывает на то, что высказывания могут объединяться с помощью связывающих союзов. С математической точки зрения эти союзы реализуют операции над высказываниями. В математической логике рассматривают только такие операции над высказываниями, для которых истинность результирующего высказывания можно определить исходя из истинности исходных независимо от смысла этих высказываний, т.е. независимо от того, что именно утверждают эти высказывания.

Вообще говоря, под операцией на множестве A понимают любое отображение φ: $A^n \to A$, которое любому упорядоченному набору из n элементов множества A (кортежу) ставит в соответствие элемент того же множества. Натуральный показатель n называют ***арностью*** этой операции. Ясно, что множество всех операций на заданном множестве бесконечно, даже если само множество конечно. Однако на практике ограничиваются операциями небольшой арности: ***унарными*** ($n = 1$), ***бинарными*** ($n = 2$), ***тернарными*** ($n = 3$). Кроме того, практические важные операции обладают определенными свойствами (например, свойство коммутативности для бинарных операций).

Ассоциация „операция — функция“ наталкивает на мысль о введении нульарных операций (функций без аргументов). С точки зрения математики нульарная операция — это постоянная функция или, иными словами, некоторый фиксированный элемент множества A. В качестве таких элементов выступают, например, число 0 или 1, т.е. элементы, чем-то выделяющиеся с точки зрения других операций.

Выделим пять операций над высказываниями:

1) дизъюнкция (соответствует союзу „или“);

2) конъюнкция (соответствует союзу „и“);

3) импликация (соответствует фразе типа „если ..., то „);

4) эквиваленция (соответствует фразе типа „... тогда и только тогда, когда ...);

5) отрицание (соответствует союзу „не“).

Первые четыре из этих операций бинарные, последняя унарная. Относительно указанных операций можно сказать лишь, что они формируют новое высказывание. При этом, зная, истинны или ложны исходные высказывания, можно сказать, является ли истинным вновь

образованное высказывание. Введенные пять операций мы будем называть ***логическими*** или ***пропозициональными***.

Для символической записи высказываний и операций над ними необходимо ввести некоторый набор символов и сформулировать правила записи с помощью этих символов. Набор символов (алфавит) в сочетании с набором правил записи представляет собой некоторый язык — язык алгебры высказываний. Вообще говоря, алфавит позволяет из заданного набора символов составлять цепочки символов (слова, фразы). Множество всех цепочек символов разделяется на две группы: правильные и неправильные. Правильные цепочки можно назвать словами, их множество и называется языком.

На практике язык можно определить как множество цепочек, удовлетворяющих некоторым требованиям, которые в конечном счете сводятся к тому, является ли данная цепочка интепретируемой с точки зрения выполнения операций или нет. Но часто язык определяют, указывая правила, по которым из уже построенных слов можно строить новые. Набор таких правил называется ***грамматикой***. Хотя любой математический язык проще любого естественного, многие закономерности у них схожи. Так слова русского языка собираются из отдельных частей с помощью некоторых правил. Описание языка с помощью грамматики в математике приводит к особому типу определения, так называемому индуктивному определению. При таком определении сначала задаются первичные, элементарные слова (это база индукции), а затем указываются правила, по которым из уже построенных слов формируются новые (шаг индукции).

Алфавит языка алгебры высказываний составляют множество ***пропозициональных переменных***, множество *функциональных символов* (символов операций, или ***логических связок***) $\vee$, $\wedge$, $\to$, $\sim$, $\neg$ и множество служебных символов (две круглые скобки). Формулы языка вводятся индуктивно.

База индукции: пропозициональные переменные представляют собой формулы.

Шаг индукции: если X и Y — формулы, то формулами являются $(X \vee Y)$, $(X \wedge Y)$, $(X \to Y)$, $(X \sim Y)$, $\neg X$.

Договоримся о следующих обозначениях. Будем обозначать: пропозициональные переменные — строчными латинскими буквами конца алфавита (x, y, z и т.д.); какие-либо формулы — прописными латинскими буквами конца алфавита (X, Y, Z и т.д.). Для сокращения количества скобок в формулах с целью улучшить их читаемость договоримся о дополнительных правилах. Это правило приоритета операций. Вместо, например $((X \Delta Y) \Delta Z)$, как положено по определению, будем писать $X \Delta Y \Delta Z$, имея в виду, что операции выполняются слева направо, причем сначала отрицание, затем дизъюнкция и конъюнкция, а потом ипликация и эквиваленция. Соглашение не относится к самому языку и служит лишь для удобства: в любой момент сокращенную запись формулы однозначным образом можно преобразовать в полную запись со всеми нужными скобками. В математический язык эти правила вводить нет смысла, поскольку они ничего не добавляют к математической сути вопроса.

В наших рассуждениях будут встречаться формулы, которые относятся к введенному языку, но, кроме того, придется использовать и дополнительные обозначения и символику, чтобы рассуждать не в рамках языка, а о самом языке и его формулах. Так, обозначения переменных — это элемент языка алгебры высказываний, а обозначения формул или соглашение о приоритете уже выходят за рамки языка. В этом случае говорят о расширении рассматриваемого языка или о ***метаязыке***. На практике язык и метаязык тесно переплетаются и трудно разделимы.

Рассмотрим какую-либо формулу, например $(x \to y) \to z$. Об истинности этой формулы можно судить, если известно, истинны ли элементарные высказывания x, y, z. Например, если x и z ложны, а y истинна, то $x \to y$ является истинной, а $(x \to y) \to z$ ложной. Приписав истинной формуле значение 1, а ложной — 0, мы получаем функцию, которая по значениям истинности пропозициональных переменных (элементарных высказываний) определяет значение истинности всей формулы. Эта функция есть отображение вида h: $\{0, 1\}^n \to \{0, 1\}$ (и аргументы, и функция принимают значения 0 и 1). Она называется ***истинностной функцией*** данной

формулы. Позже мы увидим, что любая двузначная (булева) функция от n аргументов есть истинностная функция некоторой пропозициональной формулы с n переменными.

Замечание. Отметим различия понятий „формула“ и „функция“. Первое понятие связано с языком, на котором описываются свойства математических объектов. Формула имеет некоторый набор переменных, если заданы значения этих переменных, то формула также имеет некоторое значение. В этом смысле формулу можно рассматривать как запись некоего правила, которое заданным значениям переменных ставит в соответствие новое значение. Это похоже на понятие функции, но функция (отображение) есть реальный математический объект, а не запись о нем в некотором языке. Кроме того, исходные значения, по которым отображение формирует значение, должны быть упорядочены. Одна и та же формула задает разные функции. Так, формула $x - t$ определяет и функцию $f(x,t)$, и функцию $f(t,x)$, а также, например, функцию $f(x,y,t)$. Чтобы с формулой связать некоторую функцию n аргументов, необходимо установить связь между аргументами функции и переменными формулы. Это в языках программирования реализуется формальными переменными: $f(x,t) = x - t$.

Понятие истинностной функции позволяет и введенные логические операции рассматривать совершенно формально — как формульную запись некоторых истинностных функций (табл. 1.1). В дальнейшем через $|X|$ мы будем обозначать истинностную функцию формулы X.

Таблица 1.1

x	y	$\neg x$	$x \vee y$	$x \wedge y$	$x \to y$	$x \sim y$
0	0	1	0	0	1	1
0	1		1	0	1	0
1	0	0	1	0	0	0
1	1		1	1	1	1

1.3. Тавтологии и эквивалентность формул

Тавтологии. Формулы выполнимые и опровержимые. Теорема о правиле modus ponens: если X и $X \to Y$ — тавтологии, то и Y — тавтология. Подстановка. Эквивалентность формул алгебры высказываний. Теорема о заменах. Следствие 1: если X — тавтология, то и $\mathbf{S}(z_1, z_2, \ldots, z_n | Y_1, Y_2, \ldots, Y_n | X)$ — тавтология. Следствие 2: инвариантность эквивалентности относительно логических операций. Способы получения эквивалентных формул: на основе свойств логических операций; на основе взаимосвязей операций; на основе двойственности.

Среди формул алгебры высказываний выделяют:

– ***тождественно истинные***, или ***тавтологии***, истинные при любой истинности переменных;

– ***выполнимые***, имеющие значение 1 хотя бы для одного набора значений пропозициональных переменных;

– ***опровержимые***, имеющие значение 0 хотя бы для одного набора значений пропозициональных переменных.

– ***тождественно ложные***, или ***противоречия***, ложные при любой истинности переменных.

Эти четыре понятия связаны между собой. Формула, не являющаяся опровержимой, есть тавтология. Такие формулы описывают универсальные логические законы. Именно с использованием тавтологий проводится любое математическое доказательство. Формула, не являющаяся выполнимой, тождественно ложна, т.е. представляет собой противоречие.

Для тавтологий (противоречий) истинностная функция есть константа 1 (0). Для выполнимых формул истинностная функция не равна постоянной 0, а для опровержимых — постоянной 0.

Пример 1.1. Формула $(x \vee (\neg y)) \to z$ является одновременно выполнимой и опровержимой: она истинна, если переменная z обозначает истинное высказывание, и ложна, если, например, переменные y и z обозначают ложные высказывания. Это можно увидеть, составив истинностную функцию, которая в данном случае описывается вектором 01110101. Формула $(x \vee (\neg x))$ является тавтологией, а формула $(x \wedge (\neg x))$ — противоречием. #

Следующее утверждение отражает часто используемое на практике умозаключение, называемое modus ponens (модус поненс).

Теорема 1.1. Если формулы X и $X \to Y$ являются тавтологиями, то и формула Y есть тавтология.

◀ Пусть формулы X и Y построены из переменных $z_1, \ldots, z_n$. Выберем для этих переменных какие-либо значения. Тогда об истинности формулы $X \to Y$ можно судить на основании истинности формул X и Y. Анализируя истинностную функцию для импликации, видим, что при истинности X и ложности Y формула $X \to Y$ является ложной. Но по условию теоремы эта формула тождественно истинная, как и формула X. Следовательно, формула Y не может быть ложной при выбранных значениях переменных. Поскольку значения переменных выбирались произвольно, заключаем, что формула Y тождественно истинна, т.е. тавтология. ▶

Введем понятие ***эквивалентных формул*** алгебры высказываний — формул, имеющих равные истинностные значения при любых значениях входящих в формулы переменных. Альтернативное определение: формулы X и Y называются эквивалентными, если формула $X \sim Y$ является тавтологией. Нетрудно показать, рассуждая как в последней теореме, что формула $X \sim Y$ является тавтологией тогда и только тогда, когда при любых значениях переменных формулы X и Y одновременно или истинны, или ложны, т.е. истинностные функции этих формул совпадают. Для эквивалентных формул введем обозначение $X \equiv Y$. Итак, $X \equiv Y \Leftrightarrow$ $\Leftrightarrow X \sim Y$ — тавтология. Обратите внимание на три символа эквивалентности в последней фразе. Первый обозначает отношение эквивалентности формул, введенное на множестве формул алгебры высказываний, третий — операцию эквиваленции, а второй — по сути та же эквиваленция, но в утверждении, в котором формулируется свойство самой алгебры высказываний, и ее следует отнести к сфере метаязыка.

Имеются некоторые стандартные приемы, приводящие к эквивалентным формулам. Один из них — подстановка. Если мы в формуле заменим одну из подформул другой, то получим новую цепочку символов. Нетрудно доказать индукцией по построению, что это цепочка будет формулой. Заменяемая подформула может встречаться несколько раз. В этом случае говорят о ***вхождениях подформулы***. Замена может выполняться для одного какого-либо вхождения данной подформулы или для всех.

Под подстановкой будем понимать замену всех вхождений в формулу одной или нескольких переменных некоторыми формулами. Результат подстановки в формулу X вместо переменных $z_1, \ldots, z_n$ формул $Y_1, \ldots, Y_n$ обозначают примерно так: $\mathbf{S}(z_1, \ldots, z_n | Y_1, \ldots, Y_n | X)$. Мы также будем использовать обозначение $X^{z_1,\ldots,z_n}_{Y_1,\ldots,Y_n}$.

Отметим, что замена в формуле X одного или нескольких вхождений подформулы Y формулой Z можно рассматривать как получение самой формулы X и результата замены $\widetilde{X}$ подстановкой в третью формулу $\widehat{X}$ вместо некоторой переменной u формул Y (для получения X) и Z (для получения $\widetilde{X}$). Какие именно вхождения Y меняются, определяется конструкцией формулы $\widehat{X}$. Например, в формуле $X = (x \to y) \vee (x \wedge (x \to y))$ два вхождения формулы $Y = x \to y$. Взяв $\widehat{Z} = u \vee (x \wedge (x \to y))$, в результате подстановки $\widehat{Z}^u_Y$ получим X, а в результате подстановки $\widehat{Z}^u_{x \sim y}$ получим формулу $\widetilde{X} = (x \sim y) \vee (x \wedge (x \to y))$ — результат замены в X первого вхождения Y на формулу $Z = x \sim y$.

Введем также обозначение $X[x_1, x_2, \ldots, x_n]$, имея в виду, что список $x_1, x_2, \ldots, x_n$ содержит все переменные, входящие в формулу X.

Теорема 1.2. Если $X \equiv Y$, то при замене всех вхождений какой-либо переменной u и в формуле X, и в формуле Y какой-либо формулой Z получим эквивалентные формулы, т.е.

$$X \equiv Y \quad \Rightarrow \quad X_Z^u \equiv Y_Z^u.$$

Если в формуле X заменить одно из вхождений подформулы Y эквивалентной формулой Z, то получим формулу, эквивалентную X, т.е.

$$Y \equiv Z \quad \Rightarrow \quad X \equiv X_Z^Y.$$

◀ Объединим списки переменных у формул X, Y, Z в общий упорядоченный список z_1, z_2, …, z_k, u. Тогда условие эквивалентности $X \equiv Y$ можно записать как равенство истинностных функций:

$$f_X(z_1, \ldots, z_k, u) = f_Y(z_1, \ldots, z_k, u). \tag{1.1}$$

Замена всех вхождений переменной u формулой Z приводит к композиции истинностных функций: если $\widetilde{X} = X_Z^u$, $\widetilde{Y} = Y_Z^u$, то

$$f_{\widetilde{X}}(z_1, \ldots, z_k, u) = f_X(z_1, \ldots, z_k, f_Z(z_1, \ldots, z_k, u)) \tag{1.2}$$

и

$$f_{\widetilde{Y}}(z_1, \ldots, z_k, u) = f_Y(z_1, \ldots, z_k, f_Z(z_1, \ldots, z_k, u)) \tag{1.3}$$

Из равенств (1.1)–(1.3) вытекает, что

$$f_{\widetilde{X}}(z_1, \ldots, z_k, u) = f_{\widetilde{Y}}(z_1, \ldots, z_k, u),$$

а это равносильно эквивалентности формул $\widetilde{X}$ и $\widetilde{Y}$.

Пусть дана формула $X[x_1, \ldots, x_n]$, в которую входит подформула $Y[x_1, \ldots, x_n]$ (мы можем считать, что подформула включает все переменные исходной формулы, рассматривая недостающие как фиктивные). Заменим подформулу Y новой переменной z, которая не входит в список x_1, …, x_n. Получим новую формулу $\Gamma[x_1, \ldots, x_n, z]$, связь которой с исходной формулой можно записать в виде $X = \Gamma\big|_Y^z$. Замена подформулы $Y[x_1, \ldots, x_n]$ эквивалентной формулой $Z[x_1, \ldots, x_n]$ приведет к новой формуле $\widetilde{X} = \Gamma\big|_Z^z$.

Задав упорядоченный список переменных x_1, x_2, …, x_n, z, можем для формул рассмотреть их истинностные функции. По условию $f_Y(x_1, x_2, \ldots, x_n) = f_Z(x_1, x_2, \ldots, x_n)$ (так как $Y \equiv Z$), Подстановки вместо z формул Y и Z можно записать как композицию функций:

$$f_X(x_1, x_2, \ldots, x_n) = f_\Gamma(x_1, x_2, \ldots, x_n, f_Y(x_1, x_2, \ldots, x_n)),$$
$$f_{\widetilde{X}}(x_1, x_2, \ldots, x_n) = f_\Gamma(x_1, x_2, \ldots, x_n, f_Z(x_1, x_2, \ldots, x_n)).$$

Из равенства $f_Y(x_1, x_2, \ldots, x_n) = f_Z(x_1, x_2, \ldots, x_n)$ вытекает равенство $f_X(x_1, x_2, \ldots, x_n) = f_{\widetilde{X}}(x_1, x_2, \ldots, x_n)$, что равносильно эквивалентности $X \equiv \widetilde{X}$. ▶

Следствие 1.1. Если X — тавтология, то и $\mathbf{S}(z_1, \ldots, z_n | Y_1, \ldots, Y_n | X)$ — тавтология.

◀ Можно рассуждать так. Тавтологии — это формулы, эквивалентные, например, формуле $W = x \vee \neg x$. В качестве переменной x можно выбрать ту, которая не входит в формулу X. В силу теоремы, заменив в формулах X и W все вхождения переменных z_1, …, z_n формулами Y_1, …, Y_n, мы получим эквивалентные формулы. Но формула W при этом не изменится. Поэтому вновь построенная формула $\mathbf{S}(z_1, \ldots, z_n | Y_1, \ldots, Y_n | X)$ будет эквивалентна формуле W, т.е. будет являться тавтологией. ▶

Следствие 1.2. Пусть $X \equiv Z$ и $Y \equiv W$. Тогда $(X \vee Y) \equiv (Z \vee W)$, $(X \wedge Y) \equiv (Z \wedge W)$, $(X \to Y) \equiv (Z \to W)$, $(X \sim Y) \equiv (Z \sim W)$, $(\neg X) \equiv (\neg Z)$.

◄ Формулу $(Z \circ W)$, где $\circ$ — одна из логических связок, можно рассматривать как результат замены в формуле $(X \circ Y)$ сперва подформулы X эквивалентной формулой Z, а затем подформулы Y эквивалентной формулой W. Согласно доказанной теореме такая замена приводит к эквивалентной формуле. Аналогичны рассуждения и для отрицания. ►

С помощью подстановки можно получать эквивалентные формулы, отталкиваясь от известных свойств логических операций.

Теорема 1.3. Для любых пропозициональных формул X, Y и Z верны следующие эквивалентности:

1) $X \wedge X \equiv X$;
2) $X \wedge Y \equiv Y \wedge X$;
3) $(X \wedge Y) \wedge Z \equiv X \wedge (Y \wedge Z)$;
4) $X \vee X \equiv X$;
5) $X \vee Y \equiv Y \vee X$;
6) $(X \vee Y) \vee Z \equiv X \vee (Y \vee Z)$;
7) $X \wedge (Y \vee Z) \equiv (X \wedge Y) \vee (X \wedge Z)$;
8) $X \vee (Y \wedge Z) \equiv (X \vee Y) \wedge (X \vee Z)$;
9) $X \wedge (Y \vee X) \equiv X$;
10) $X \vee (Y \wedge X) \equiv X$.

◄ Непосредственно из таблицы для истинностной функции вытекает, что $x \wedge x \equiv x$. Подставив вместо всех вхождений переменной x формулу X, получим эквивалентность $X \wedge X \equiv X$. Остальные эквивалентности доказываются аналогично. ►

Еще один способ получения эквивалентностей — замена одних операций другими по соответствующим формулам. Из теории булевых функций вытекает, что верны следующие эквивалентности:

$$(\neg(\neg x)) \equiv x, \quad (\neg(x \vee y)) \equiv (\neg x) \wedge (\neg y), \quad (x \to y) \equiv ((\neg x) \vee y).$$

Отталкиваясь от этих эквивалентностей можно доказать следующее.

Теорема 1.4. Для любых пропозициональных формул X, Y и Z верны следующие эквивалентности:

1) $(\neg(\neg X)) \equiv X$ (закон двойного отрицания);
2) $(\neg(X \vee Y)) \equiv (\neg X) \wedge (\neg Y))$ (перенос отрицания через конъюнкцию);
3) $(\neg(X \wedge Y)) \equiv (\neg X) \vee (\neg Y))$ (перенос отрицания через дизъюнкцию);
4) $(\neg(X \to Y)) \equiv (X \wedge (\neg Y))$ (перенос отрицания через импликацию);
5) $(X \to Y) \equiv ((\neg X) \vee Y)$ (представление импликации через дизъюнкцию);
6) $(X \to (\neg X)) \equiv (\neg X)$ (закон упрощения);
7) $(X \to Y) \equiv ((\neg Y) \to (\neg X)$ (закон контрапозиции).

◄ Доказывается теорема так же, как и предыдущая. Например, на основании простой эквивалентности $(\neg(\neg x)) \equiv x$, устанавливаемой непосредственно, путем подстановки вместо переменной x формулы X получаем эквивалентность $(\neg(\neg X)) \equiv X$. ►

Для формул, содержащих только три операции $\neg$, $\vee$, $\wedge$, имеет место принцип двойственности. Для каждой такой формулы X в результате взаимной замены операций $\vee$ и $\wedge$ получим новую формулу X^*, которую назовем двойственной для X. Отметим, что если X^* двойственна X, то X двойственна X^*, так что отношение двойственности симметрично. Это можно выразить формулой $X^{**} = X$ (знак равенства отражает совпадение формул, а не их эквивалентность). Учитывая эквивалентности

$$\neg(x \vee y) \equiv \neg x \wedge \neg y, \quad \neg(x \wedge y) \equiv \neg x \vee \neg y, \tag{1.4}$$

Приходим к эквивалентности

$$X^* \equiv \neg X^{t_1,\ldots,t_n}_{\neg t_1,\ldots,\neg t_n}, \tag{1.5}$$

где t_1, t_2, …, t_n — полный список переменных, входящих в X. Доказательство проводится индукцией по построению формулы. Действительно, для переменных утверждение очевидно. Пусть $X = Y \vee Z$. Тогда, согласно первой эквивалентности (1.4) $\neg X = \neg(Y \vee Z) \equiv \neg Y \wedge \neg Z$. Предполагая в соответствии с индуктивным предположением, что

$$\neg Y^{t_1,\ldots,t_n}_{\neg t_1,\ldots,\neg t_n} \equiv Y^*, \quad \neg Z^{t_1,\ldots,t_n}_{\neg t_1,\ldots,\neg t_n} \equiv Z^*,$$

заключаем, что

$$X^* = Y^* \wedge Z^* \equiv \neg Y^{t_1,\ldots,t_n}_{\neg t_1,\ldots,\neg t_n} \wedge \neg Z^{t_1,\ldots,t_n}_{\neg t_1,\ldots,\neg t_n} = (\neg Y \wedge \neg Z)\Big|^{t_1,\ldots,t_n}_{\neg t_1,\ldots,\neg t_n} \equiv \neg X^{t_1,\ldots,t_n}_{\neg t_1,\ldots,\neg t_n}.$$

Аналогично доказательство в случае второй операции.

Теорема 1.5. Если формулы X и Y эквивалентны, то и формулы X^*, Y^* эквивалентны.

◀ Действительно, из эквивалентности $X \equiv Y$ получаем $\neg X \equiv \neg Y$ и $\neg X^{t_1,\ldots,t_n}_{\neg t_1,\ldots,\neg t_n} \equiv \neg Y^{t_1,\ldots,t_n}_{\neg t_1,\ldots,\neg t_n}$. Согласно (1.5) делаем вывод $X^* \equiv Y^*$. ▶

Принцип двойственности приводит к еще одному способу получения эквивалентных формул.

1.4. Функции алгебры логики

Понятие булевой функции. Множества $\mathcal{P}_{2,n}$ и $\mathcal{P}_2$. Композиция функций. Множество функций и его замыкание. Замкнутые и полные множества функций. Теорема: если F полно и все функции в F выражаются через функции множества G, то G полно. Понятие базиса. Стандартный базис. Базис Жегалкина. Понятие ДНФ и КНФ. Примеры построения ДНФ и КНФ. Совершенные ДНФ и КНФ. Теорема о существовании СДНФ и СКНФ. Утверждение: каждая истинностная функция соответствует некоторой пропозициональной формуле. Штрих Шеффера и стрелка Пирса.

Под ***функцией алгебры логики***, или ***булевой функцией*** понимают любое отображение $f\colon \{0, 1\}^n \to \{0, 1\}$. Это значит, что каждый аргумент булевой функции, как и сама функция, принимает два значения 0 и 1. Частными случаями булевых функций являются логические операции, которые формально оперируют высказываниями, но по сути своей есть функции, в которых и аргументы, и значение могут принимать лишь два значения: ИСТИНА и ЛОЖЬ. Так, отрицание есть булева функция одного переменного, а дизъюнкция, конъюнкция, импликация, эквиваленция — функции двух переменных.

Булеву функцию можно рассматривать как некое логическое условие, которые по входам — значениям аргументов вырабатывает логическое значение 0 (ложь) или 1 (истина).

Для булевых функций можно ввести операцию ***суперпозиции***. Это вариант обычной композиции отображений, использующий понятие формулы. Пусть заданы функции $f(x_1, \ldots, x_n)$, $g_1(u_{11}, \ldots, u_{1,m_1})$, …, $g_n(u_{n1}, \ldots, u_{n,m_n})$. Тогда можно образовать функцию

$$F(u_{11}, \ldots, u_{1,m_1}, \ldots, u_{n1}, \ldots, u_{n,m_n}) = f(g_1(u_{11}, \ldots, u_{1,m_1}), \ldots, g_n(u_{n1}, \ldots, u_{n,m_n})),$$

которую и назовем суперпозицией функций f, g_1, …, g_n. В действительности при суперпозиции некоторые из переменных u_{ij} могут совпадать. В частном случае все функции g_i могут иметь один и тот же набор переменных, и тогда суперпозиция — это обычная композиция отображений. Здесь нетрудно увидеть, что суперпозиция — это построение некоторой формулы из исходных функций и выбор функции, определяемой этой формулой. Порядок переменных определяется порядком функций и порядком их аргументов.

Мы не будем здесь проводить строгие построения, но поставим следующий вопрос: каково множество булевых функций, которое может быть получено из данного множества функций F с использованием суперпозиции? Ясно, что ответ зависит от множества F. В первую очередь нас будут интересовать условия на множество F, при выполнении которых можно утверждать, что каждая булева функция может быть построена таким способом.

Множество всех булевых функций, которые могут быть получены из данного множества X функций с использованием суперпозиции, назовем ***замыканием*** X и обозначим $[X]$. Множество X булевых функций назовем ***полным***, если его замыкание совпадает с множеством $\mathcal{P}_2$ всех булевых функций. Конечное полное множество называют ***базисом*** в множестве булевых функций.

Операция замыкания сродни, например, замыканию множества элементов группы по операции группы или замыканию множества точек на плоскости, состоящему в присоединении к множеству всех его предельных точек. Замыкание можно рассматривать как унарную операцию на подмножествах множества $\mathcal{P}_2$. Свойства операции замыкания:

1) $[\varnothing] = \varnothing$;
2) $[[X]] = [X]$;
3) $X \subset [X]$;
4) $[X] \cup [Y] \subset [X \cup Y]$.

Первое свойство носит формальный характер. Второе вытекает из конечности процедуры построения любой формулы: достаточно, следуя по дереву синтаксического анализа, последовательно заменять функции из $[X]$ их формулами над X. Третье и четвертое свойства очевидны.

Из четвертого свойства вытекает, что если $X \subset Y$, то $[X] \subset [Y]$. Действительно, включение $X \subset Y$ равносильно равенству $X \cup Y = Y$. Если $X \subset Y$, то в силу свойства 4 заключаем, что $[X] \cup [Y] \subset [X \subset Y] = [Y]$. Следовательно, $[X] \subset [Y]$.

Из указанных свойств замыкания вытекает следующее утверждение.

Теорема 1.6. Если F — полное множество булевых функций, каждая из которых представима формулой над множеством G, то и G — полное множество.

◀ Так как каждая функция из F есть формула над G, то $F \subset [G]$. Из свойств замыкания вытекает, что

$$[F] \subset [[G]] = [G].$$

Но поскольку $[F]$ совпадает с множеством всех булевых функций, то и $[G]$ совпадает с множеством всех булевых функций, т.е. G — полный базис. ▶

Доказанная теорема позволяет строить базисы исходя из уже известных базисов. Один из таких базисов (это станет ясно из дальнейшего) составляют дизъюнкция, конъюнкция и отрицание. Этот базис назовем ***стандартным***. Через эти операции можно выразить другие логические операции:

$$X \to Y \equiv \neg X \vee Y, \quad X \sim Y \equiv (X \wedge Y) \vee (\neg X \wedge \neg Y).$$

Имея в виду эти формулы можно было бы ограничиться только тремя операциями $\neg$, $\vee$, $\wedge$. Более того, можно ограничиться двумя операциями, заменив $\vee$ или $\wedge$.

Среди формул, содержащих дизъюнкцию, конъюнкцию и отрицание можно выделить в некотором роде канонические формулы.

Назовем элементарной конъюнкцией формулу $X_1 \wedge X_2 \wedge \ldots \wedge X_n$, в которой каждая подформула X_i либо элементарная, либо отрицание элементарной. Удобно ввести обозначение $x_i^{\sigma_i}$ с $\sigma_i \in \{0, 1\}$, считая, что $x_i^1 = x_i$ и $x_i^0 = \neg x_i$. Тогда элементарную конъюнкцию можно записать в виде $x_1^{\sigma_1} \wedge x_2^{\sigma_2} \wedge \ldots \wedge x_n^{\sigma_n}$.

Элементарная конъюнкция или дизъюнкция нескольких элементарных конъюнкций называется ***дизъюнктивной нормальной формой***, или ***ДНФ***. Пример: $(x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3)$.

Двойственным к ДНФ понятием является ***конъюнктивная нормальная форма***, или ***КНФ***. Это элементарная дизъюнкция или конъюнкция нескольких элементарных дизъюнкций.

Количество переменных в элементарной конъюнкции называется ее длиной. Если в ДНФ все элементарные конъюнкции имеют одинаковый состав переменных и отличаются только распределением между переменными знаков отрицания, то ДНФ называется ***совершенной*** (сокращенно ***СДНФ***). Аналогично вводится понятие ***совершенной КНФ*** (СКНФ).

Теорема 1.7. Каждая формула алгебры высказываний, не являющаяся противоречием, имеет эквивалентную ей совершенную ДНФ. Каждая формула алгебры высказываний, не являющаяся тавтологией, имеет эквивалентную ей совершенную КНФ. #

◄ Структура самой формулы не имеет значения. Пусть x_1, x_2, …, x_n — список всех переменных, входящих в формулу, и $f(\xi_1, \xi_2, \ldots, \xi_n)$ истинностная функция этой формулы (ξ_i — истинностное значение переменной x_i).

Отметим, что элементарная конъюнкция $x_1^{\sigma_1} \wedge x_2^{\sigma_2} \wedge \ldots \wedge x_n^{\sigma_n}$ истинна при $\xi_1 = \sigma_1$, $x_2 = \sigma_2$, …, $\xi_n = \sigma_n$ и ложна при любом другом варианте истинностных значений переменных. Поэтому в СДНФ при заданном наборе истинностных значений переменных максимум одна элементарная конъюнкция имеет истинностное значение 1.

Для каждого набора σ_1, σ_2, …, σ_n, для которого $f(\sigma_1, \sigma_2, \ldots, \sigma_n) = 1$ составляем элементарную конъюнкцию $x_1^{\sigma_1} \wedge x_2^{\sigma_2} \wedge \ldots \wedge x_n^{\sigma_n}$, а затем из них составляем СДНФ. Получим формулу, для которой $f(\xi_1, \xi_2, \ldots, \xi_n)$ является истинностной функцией. Значит, эта СДНФ эквивалентна исходной формуле. Для построения СДНФ требуется лишь, чтобы истинностная функция хотя бы при одном наборе истинностных значений переменных принимала значение 1, т.е. исходная формула не должна быть противоречием.

Утверждение о КНФ является двойственным утверждением о ДНФ и может быть получено взаимной заменой в рассуждениях дизъюнкции и конъюнкции. ►

Следствие 1.3. Каждая булева функция является истинностной функцией какой-либо пропозициональной формулы.

◄ Действительно, доказательство теоремы построено так, что характер формулы не является существенным. Если $f(\xi_1, \xi_2, \ldots, \xi_n)$ не равна тождественно 0, мы можем для нее построить СДНФ. Если $f(\xi_1, \xi_2, \ldots, \xi_n)$ не равна тождественно 1, мы можем построить СКНФ. Таким образом, любая функция имеет либо СДНФ, либо СКНФ, либо и то, и другое. ►

Исходя из стандартного базиса можно строить и другие базисы. Один из них — ***базис Жегалкина***, состоящий из сложения по модулю 2 „$\oplus$“, умножения (она же конъюнкция) „$\cdot$“, и нульарной операции (постоянной функции) 1. Отметим, что алгебраическая система $(\{0,\ 1\},\ \{\oplus,\ \cdot\})$ есть поле $\mathbb{Z}_2$ вычетов по модулю 2, а формулы, построенные на операциях „$\oplus$“ и „$\cdot$“ представляют собой многочлены в $\mathbb{Z}_2$ (их называют ***многочленами Жегалкина***).

Нетрудно прямой проверкой убедиться в том, что

$$x \oplus y \equiv \overline{x}y + x\overline{y} \equiv (\neg x \wedge y) \vee (x \wedge \neg y), \quad 1 \equiv x + \overline{x} \equiv x \vee \neg x,$$

где первое представление дано в альтернативной символике операций: дизъюнкция как сложение $x + y$, конъюнкция как умножение xy, отрицание как ***дополнение*** $\overline{x}$. Согласно теореме 1.6, базис Жегалкина — полное множество.

Представление булевой функции через базис Жегалкина — это представление ее в виде многочлена в поле $\mathbb{Z}_2$, при этом все переменные в многочлене имеют степень 1, поскольку в $\mathbb{Z}_2$ имеем $x^k = x$ для любого элемента x. Учитывая это, заключаем, что для функции $f(x_1, \ldots, x_n)$ от n аргументов соответствующий многочлен имеет не более 2^n слагаемых (одно слагаемое нулевой степени, n слагаемых 1-й степени, C_n^2 слагаемых 2-й степени и т.д.). Например, для функции трех переменных многочлен Жегалкина имеет вид:

$$f(x_1, x_2, x_3) = a_0 \oplus a_{11}x_1 \oplus a_{12}x_2 \oplus a_{13}x_3 \oplus a_{21}x_1x_2 \oplus a_{22}x_1x_3 \oplus a_{23}x_2x_3 \oplus a_3x_1x_2x_3.$$

Итак, в многочлене Жегалкина 2^n слагаемых и, значит, 2^n коэффициентов. Записывая 2^n значений функции, мы получим систему уравнений, из которой можем найти все коэффициенты многочлена. Этот подход носит название ***метода неопределенных коэффициентов***. В принципе можно показать, что такая система имеет и притом единственное решение (это следует из того, что любая булева функция представима многочленом Жегалкина, причем такой многочлен единственный).

Кроме базиса Жегалкина существуют и другие базисы. Основным способом проверки множества функций на полноту является сведение к стандартному базису. Оказывается, такую проверку можно свести к простой проверке некоторых свойств функций заданного множества.

Введем пять так называемых ***классов Поста***. Класс T_0 содержит все функции, удовлетворяющие условию $f(0,0,\ldots,0)=0$, т.е. функции, принимающие нулевое значение при нулевых значениях всех аргументов. Аналогично T_1 — это класс функций, удовлетворяющих условию $f(1,1,\ldots,1)=1$. Класс S — это класс ***самодвойственных функций***, т.е. функций, удовлетворяющих условию $\overline{f(x_1,\ldots,x_n)}=f(\overline{x}_1,\ldots,\overline{x}_n)$, где $\overline{x}=1-x$ — отрицание. Вектор значений самодвойственной функции удовлетворяет условию $\overline{b_1b_2\ldots b_n}=b_nb_{n-1}\ldots b_1$.

Для двух булевых векторов $a=a_1a_2\ldots a_n$ и $b=b_1b_2\ldots b_n$ полагаем $a\leqslant b$, если $a_i\leqslant b_i$, $i=\overline{1,\,n}$. Булеву функцию $f(x)=f(x_1,x_2,\ldots,x_n)$ назовем ***монотонной***, если $f(x)\leqslant f(y)$ при $x\leqslant y$. Множество всех монотонных функций составляют класс M.

Наконец, класс L ***линейных функций*** составляют функции, у которых полином Жегалкина имеет степень не выше первой.

Каждый из классов Поста является замкнутым множеством. Доказательство можно провести методом индукции по построению формулы. При этом ни один из классов не совпадает с множеством $\mathcal{P}_2$ всех булевых функций. Отсюда вытекает, что если заданное множество F булевых функций включено в один из классов Поста, то оно не является полным, поскольку его замыкание также будет включено в этот класс Поста. Мы получили необходимое условие полноты множества булевых функций. Оказывается, что это условие является и достаточным.

Теорема 1.8 (критерий Поста). Множество F булевых функций полно тогда и только тогда, когда оно не является подмножеством ни одного из классов Поста.

◀ Необходимость сформулированного критерия установлена выше. Поэтому сосредоточим внимание на доказательстве достаточности критерия. Это доказательство сводится к построению на основе множества F функций стандартного базиса, причем можно ограничиться только отрицанием и умножением, поскольку сложение (дизъюнкция) выражается через отрицание и умножение:

$$x+y=\overline{\overline{x}\,\overline{y}}.$$

Пусть F не является подмножеством ни одного из классов Поста. Для каждой функции $f\in F$ рассмотрим формулу $g(x)=f(x,x,\ldots,x)$. Поскольку F не содержится в T_0 и в T_1, в F, во-первых, есть непостоянные функции, а во-вторых есть хотя бы одна функция f_1, для которой $g_1(0)=1$, и есть хотя бы одна функция f_2, для которой $g_2(1)=0$. Это возможно, если одна из функций $g_1(x)$ и $g_2(x)$ есть отрицание, либо обе функции постоянны и представляют константы0 и 1. Рассмотрим оба случая.

Пусть функции $g_1(x)$ и $g_2(x)$ являются константы 0 и 1. Тогда для любой функции $f\in F$ формула $f(\alpha_1,\alpha_2,\ldots,\alpha_{i-1},x,\alpha_{i+1},\ldots,\alpha_n)$, в которой α_j — константы, есть формула над F. Выберем функцию f, не являющуюся монотонной. Тогда существуют два булевых вектора p и q, удовлетворяющие условиям $p<q$ и $f(p)=1$, $f(q)=0$. Векторы p и q можно соединить цепочкой $p=p_0,p_1,\ldots,p_k=q$ непосредственно предшествующих друг другу векторов (соседних). В этой цепочке найдется два соседних вектора p_{j-1} и p_j, которые отличаются только одной компонентой с некоторым номером i и на которых $f(p_{j-1})=1$, $f(p_j)=0$. Пусть α_j, $j\neq i$, одинаковые компоненты этих векторов. Тогда формула $f(\alpha_1,\alpha_2,\ldots,\alpha_{i-1},x,\alpha_{i+1},\ldots,\alpha_n)$ представляет собой операцию отрицания.

Предположим, например, что функция $g_1(x)$ является отрицанием. Тогда мы можем составлять формулы вида $f(x \oplus \sigma^1, \ldots, x \oplus \sigma_n)$, где $x \oplus \sigma$ есть переменная x при $\sigma = 0$ и ее отрицание при $\sigma = 1$. Выберем функцию $f \in F$, не являющуюся самодвойственной. Тогда можно указать такой булев вектор $p \in \mathbb{B}^n$, что $f(\overline{p}) \neq \overline{f(p)}$, откуда $f(\overline{p}) = f(p)$. Пусть σ_1, …, σ_n — компоненты вектора p. Рассмотрим функцию $g(x) = f(x \oplus \sigma^1, \ldots, x \oplus \sigma_n)$, определяемую выбранной несамодвойственной функцией f и вектором p. Так как $0 \oplus \sigma_i = \sigma_i$, $1 \oplus \sigma_i = \overline{\sigma}_i$, то $g(0) = f(p) = f(\overline{p}) = f(\sigma_1 \oplus 1, \ldots, \sigma_n \oplus 1) = g(1)$. Следовательно, функция $g(x)$ является константой. Пусть, например, $g(x) = 1$. Тогда $\overline{g}(x) = 0$ — другая константа.

Итак, мы показали, что множество F, не являющееся подмножеством классов T_0, T_1, S, M, позволяет получить в виде формул отрицание и обе константы. Для построения формулы для произведения выберем функцию $f \in F$, не являющуюся линейной. Составляя формулы вида $f(X_1, X_2, \ldots, X_n)$, где X_i — это либо переменная x, либо переменная y, мы получим функции двух аргументов. Покажем, что среди таких функций есть нелинейные. В полиноме Жегалкина, представляющем функцию f, выберем нелинейное (степени два или выше) слагаемое наименьшей степени. Пусть это слагаемое имеет вид $x_{i_1}x_{i_2}\ldots x_{i_k}$. В формуле $f(X_1, X_2, \ldots, X_n)$ положим $X_{i_1} = x$, $X_{i_j} = y$, $j = \overline{2, k}$, а для остальных переменных, не вошедших в выбранное слагаемое выберем значение 0. Тогда выбранное слагаемое преобразуется в xy, остальные нелинейные слагаемые обнулятся, и мы получим функцию вида $g(x, y) = xy \oplus \alpha x \oplus \beta y \oplus \gamma$.

Так как

$$g(x, y) = xy \oplus \alpha x \oplus \beta y \oplus \gamma = (x \oplus \beta)(y \oplus \alpha) \oplus \alpha\beta \oplus \gamma = (x \oplus \beta)(y \oplus \alpha) \oplus \gamma',$$

заключаем, что $g(x \oplus \beta, y \oplus \alpha) \oplus \gamma' = xy$. Но $x \oplus \sigma$ — это переменная x при $\sigma = 0$ и ее отрицание $\overline{x}$ при $\sigma = 1$. Поэтому, если $g(x, y)$ принадлежит замыканию F, то и функция $g(x \oplus \beta, y \oplus \alpha) \oplus \gamma' = xy$, т.е. конъюнкция, принадлежит замыканию F.

Итак, мы показали, что если множество F не является подмножеством никакого класса Поста, то формулами над F можно представить отрицание и конъюнкцию, а значит, и дизъюнкцию. В этом случае, согласно теореме 1.6, множество F полное. ▶

Пример 1.2. Полное множество составляет единственная функция $x \mid y = \neg(x \wedge y)$, называемая ***штрихом Шеффера***. Проверка критерия Поста здесь элементарна, и мы на этом не будем останавливаться. Нетрудно также с помощью этой функции представить функции стандартного базиса:

$$\overline{x} = \overline{xx} = x \mid x, \quad xy = \overline{x \mid y} = (x \mid y) \mid (x \mid y), \quad x + y = \overline{\overline{x}\,\overline{y}} = \overline{x} \mid \overline{y}$$

Аналогична стрелка Пирса. Она тоже составляет базис.

Пример 1.3. В математической логике ключевой операцией является импликация. Совместно с отрицанием она составляет базис: отрицание не попадает в классы T_0, T_1, M, а импликация оказывается за бортом классов S и L. Впрочем, как и в случае других базисов, можно через импликацию и отрицание представить дизъюнкцию и конъюнкцию, а затем сослаться на теорему 1.6.

2. ИСЧИСЛЕНИЕ ВЫСКАЗЫВАНИЙ

Напомним, что полностью формализованная математическая теория (*исчисление*) выглядит так. Имеется некоторый язык, позволяющий составлять правильные слова-формулы, которые отражают возможные утверждения теории. Есть некоторый набор формул, изначально объявленных истинными (это аксиомы). Кроме того, задан некоторый набор правил преобразования формул, которые позволяют из истинных формул получать новые истинные формулы. Все истинные утверждения теории получаются путем формальных преобразований формул в рамках узаконенных правил преобразований. Цепочка последовательных преобразований называется ***выводом***. В полностью формализованной теории утрачивается содержательный смысл формул, а все построение теории превращается в манипуляции заданными символами.

Алгебра высказываний — это содержательная теория, все ее суждения устанавливаются на базе истинностных функций, которые никак не отражаются в языке алгебры высказываний.

Полная формализация алгебры высказываний устанавливает связь получения тех или иных утверждений с реальным процессом умозаключений, который и представляет собой подлинное математическое доказательство. Кроме того, алгебра высказываний в основном сводится к теории булевых функций, т.е. к исследованию конечных объектов. Поэтому исчисление высказываний — одно из самых простых и в этом смысле удобно как иллюстрация современного подхода к формализации в математике.

Любая теория имеет множество вариантов формализации. В рамках логики высказываний мы остановимся на одном из этих вариантов, который назовем ***теорией N***.

2.1. Основные положения теории N

Язык теории $\boldsymbol{N}$. Аксиомы (их одиннадцать): 1) $X \to (Y \to X)$; 2) $X \to Y \to (X \to (Y \to Z) \to (X \to Z))$; 3) $X \wedge Y \to X$; 4) $X \wedge Y \to Y$; 5) $(X \to Y) \to ((X \to Z) \to (X \to Y \wedge Z))$; 6) $X \to X \vee Y$; 7) $Y \to X \vee Y$; 8) $X \to Z \to (Y \to Z \to (X \vee Y \to Z))$; 9) $X \to Y \to (\neg Y \to \neg X)$; 10) $\neg\neg X \to X$; 11) $X \to \neg\neg X$. Правило вывода $\dfrac{X,\ X \to Y}{Y}$ (modus ponens). Вывод в теории $\boldsymbol{N}$. Вывод из гипотез. Пример: $X \to X$. Дерево вывода.

Язык теории $\boldsymbol{N}$ — это язык алгебры высказываний. В теории $\boldsymbol{N}$ одиннадцать ***схем аксиом***. Схема аксиом отличается от аксиомы тем, что в ней используются не конкретные переменные, а символы подстановки, вместо которых могут подставляться конкретные формулы теории. При выборе вместо символов подстановки конкретных формул мы получаем конкретную аксиому. Отметим, что можно было бы избежать схем аксиом, но тогда придется вводить дополнительные правила вывода, которые позволили бы „размножить“ аксиому.

Итак, сформулируем одиннадцать схем аксиом теории $\boldsymbol{N}$:

1) $X \to (Y \to X)$;
2) $X \to Y \to (X \to (Y \to Z) \to (X \to Z))$;
3) $X \wedge Y \to X$;
4) $X \wedge Y \to Y$;
5) $(X \to Y) \to ((X \to Z) \to (X \to Y \wedge Z))$;
6) $X \to X \vee Y$;
7) $Y \to X \vee Y$;
8) $X \to Z \to (Y \to Z \to (X \vee Y \to Z))$;
9) $X \to Y \to (\neg Y \to \neg X)$;
10) $\neg\neg X \to X$;
11) $X \to \neg\neg X$.

В теории $\boldsymbol{N}$ всего одно правило вывода — правило заключения (modus ponens):

$$\frac{X,\ X \to Y}{Y}.$$

Представленная запись означает, что из двух „правильных“ формул вида X и $X \to Y$ вытекает „правильная“ формула Y.

После введения языка, аксиом и правил вывода формальная теория готова. Дальше можно начинать „игру в слова“ и получать теоремы (т.е. выводимые формулы) этой теории. На этом, собственно, заканчивается формальная часть и начинается неформальная, т.е. метатеория. Основной вопрос: что дает формальная теория в качестве выводимых формул. Отметим, что добросовестный вывод даже простых теорем оказывается весьма трудоемким, и следует прибегать к приему, хорошо известному математикам: использовать уже выведенные теоремы наравне с аксиомами.

Выводом теории $\boldsymbol{N}$ будем называть последовательность формул $X_1, X_2, \ldots, X_n$, в которой каждая формула X_i есть либо аксиома, либо получена из каких-либо предшествующих формул X_k, X_m $(k, m < i)$ по правилу modus ponens. Формула X называется выводимой в теории $\boldsymbol{N}$, если она является конечной формулой некоторого вывода. Этот факт будем обозначать следующим образом: $\vdash X$.

Для нас будет важен условный вывод, или вывод из гипотез, при котором некоторые формулы мы предполагаем истинными и на основании этого строим вывод. Такой условный вывод играет промежуточную роль, позволяя рассматривать отдельные части окончательного вывода. Кроме того, условный вывод можно рассматривать как построение вывода из нелогических аксиом, которые присутствуют во всех математических теориях (они характеризуют основные положения теории). Далее большими греческими буквами будем обозначать списки формул теории $\boldsymbol{N}$.

Выводом из гипотез Γ в теории $\boldsymbol{N}$ будем называть последовательность формул, в которой каждая формула есть либо аксиома, либо формула из списка Γ, либо она получена из предшествующих формул по правилу modus ponens. При этом конечная формула X любого вывода из гипотез Γ называется ***выводимой из гипотез*** Γ, что обозначается следующим образом: $\Gamma \vdash X$.

Пример 2.1. Построим вывод формулы $X \to X$, где X — какая-либо формула теории $\boldsymbol{N}$:

1) из схемы 1 получаем аксиому $X \to (X \to X)$;

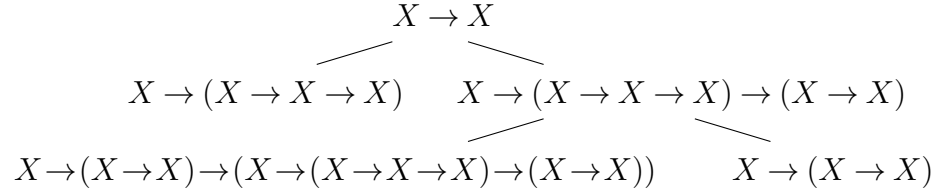
2) из схемы 2, заменяя Z на X и Y на $X \to X$, получаем аксиому $X \to (X \to X) \to (X \to (X \to \to X \to X) \to (X \to X))$,

3) из двух предыдущих формул по правилу вывода $X \to (X \to X \to X) \to (X \to X)$;

4) из схемы 1 при $X \to X$ взамен Y получаем аксиому $X \to (X \to X \to X)$;

5) из двух последних формул по правилу вывода $X \to X$.

Следует заметить, что фактически вывод представляет собой особую структуру — дерево: каждая формула вывода есть либо аксиома или гипотеза (лист дерева, начальный элемент структуры), либо имеет предшественников, из которых она получена по правилу вывода. Для формулы $X \to X$ из последнего примера дерево вывода имеет следующий вид:



2.2. Правила естественного вывода

Теорема о дедукции: если $\Gamma, X \vdash Y$, то $\Gamma \vdash X \to Y$. Структурные правила естественного вывода. Логические правила естественного вывода. Дополнительные правила естественного вывода: присоединение посылки; закон противоречия; двойное отрицание; доказательство от противного; удаление конъюнкции справа. Обобщенное правило введения дизъюнкции.

Одна из целей введения исчисления высказываний — анализ используемой практики построения доказательств. В качестве единственного правила вывода в теории взято правило modus ponens, в то время как построение доказательств на практике использует и многие другие правила: правило исключенного третьего, доказательство от противного и т.п. Оказывается, что все подобные правила можно получить из аксиом исчисления высказываний и правила modus ponens.

Веденный символ $\vdash$ выводимости в исчислении высказываний позволяет строить формулы нового типа $\Gamma \vdash X$, которые на содержательном уровне можно интерпретировать так: „если истинны формулы списка Γ, то истинна формула X. Это формулы метаязыка, позволяющие упростить процесс установления того, выводима данная формула или нет. Само правило modus ponens можно трактовать как формулу $X, X \to Y \vdash Y$. Формулу вида $\Gamma \vdash X$ будем называть ***секвенцией***. Секвенция — это логическая формула, которая может быть истинной или нет.

Чтобы приблизиться к общепринятой практике доказательств, выведем в теории $\boldsymbol{N}$ ряд дополнительных правил, называемых ***правилами естественного вывода***. Следующая теорема дает основополагающее правило естественного вывода, в некотором смысле обращающее правило modus ponens. Фактически эта теорема представляет собой утверждение об эквивалентности символа импликации $\to$ и символа выводимости $\vdash$.

Теорема 2.1 (теорема о дедукции). Если $\Gamma, X \vdash Y$, то $\Gamma \vdash X \to Y$.

◀ Доказательство строится на анализе последовательности вывода. Фактически надо показать, что для любого вывода $Z_1, Z_2, \ldots, Z_n = Y$ из гипотез Γ, X существует вывод из гипотез Γ формулы $X \to Y$. Доказательство проведем индукцией по длине вывода.

При $n = 1$ конечная формула вывода $Z_n = Y$ есть либо аксиома, либо формула из списка Γ, либо формула X (правило modus ponens в данном случае не применялось, так как длина вывода меньше двух). В первом и втором случаях строим последовательность формул Y, $Y \to (X \to Y)$ (аксиома схемы 1), $X \to Y$ (modus ponens), которая является выводом формулы $X \to Y$ из списка гипотез Γ. В третьем случае $X = Y$ и формула $X \to Y$ совпадает с выводимой формулой $X \to X$ (см. пример 2.1). Таким образом, при $n = 1$ утверждение доказано.

Предположим, что утверждение доказано для всех формул Y, имеющих вывод из гипотез Γ, X длины менее n. Рассмотрим произвольный вывод $Z_1, Z_2, \ldots, Z_n = Y$. Формула Y есть либо аксиома, либо формула из списка Γ, либо X, либо получена по правилу modus ponens. В первых трех случаях рассуждения те же, что и при $n = 1$ (в выводе можно оставить только последнюю формулу и свести дело к $n = 1$). Рассмотрим случай, когда Y получена по правилу modus ponens из формул Z_k и Z_j. В этом случае одна из формул, например Z_j, есть импликация $Z_k \to Y$. В соответствии с индукционным предположением из гипотез Γ выводимы формулы $X \to Z_k$ и $X \to (Z_k \to Y)$. Объединим два этих вывода, удалив из списка повторения формул, если они есть*. Дополняем полученное объединение следующими тремя формулами:

$X \to Z_k \to (X \to (Z_k \to Y) \to (X \to Y))$ (аксиома схемы 2);

$X \to (Z_k \to Y) \to (X \to Y)$ (modus ponens);

$X \to Y$ (modus ponens).

В результате получаем вывод формулы $X \to Y$ из гипотез Γ. В соответствии с методом математической индукции утверждение теоремы доказано для конечной формулы любого вывода из гипотез Γ, X. ▶

*По определению в выводе каждая формула есть либо аксиома, либо гипотеза, либо выводится из предыдущих по правилу modus ponens, т.е. формулы в выводе, строго говоря, могут повторяться. Однако ясно, что повторения можно убрать, не нарушая связей в выводе.

Следующая теорема устанавливает пять правил, называемых ***структурными правилами естественного вывода***. Правила сформулированы не для формул языка алгебры высказываний, а для секвенций. Они позволяют из уже известных выводимостей получать новые. При этом соответствующий вывод на самом деле не строится, а лишь формулируется заключение о существовании такого вывода.

Теорема 2.2. Для любых списков формул Γ и Δ и любых формул X, Y, Z истинны следующие секвенции:

1) $\Gamma,\, X,\, \Delta \vdash X$ (закон тождества);
2) $\dfrac{\Gamma \vdash X}{\Gamma,\, \Delta \vdash X}$ (правило добавления гипотез);
3) $\dfrac{\Gamma,\, Y,\, Y,\, \Delta \vdash X}{\Gamma,\, Y,\, \Delta \vdash X}$ (правило сокращения гипотез);
4) $\dfrac{\Gamma,\, Y,\, Z,\, \Delta \vdash X}{\Gamma,\, Z,\, Y,\, \Delta \vdash X}$ (правило перестановки гипотез);
5) $\dfrac{\Delta \vdash Y;\ \Gamma,\, Y \vdash X}{\Gamma,\, \Delta \vdash X}$ (правило сечения).

Правила, обозначенные в теореме, — элементарные следствия из определения понятия „вывод из гипотез“ и лишь фиксируют очевидное.

Отметим также очевидный факт, что мы можем произвольно вводить аксиомы в список Γ гипотез или выводить аксиомы из списка, не меняя сути. первое вытекает из правила добавления гипотез, а второе утверждение можно получить как формальное следствие правила сечения:

$$\begin{array}{ccc} & \Gamma \vdash X & \\ \swarrow & & \searrow \\ \Gamma,\, Y \vdash X & & \vdash Y \end{array}$$

Следующая теорема вводит так называемые ***логические правила естественного вывода***, описывающие некоторые общепринятые приемы проведения математических доказательств.

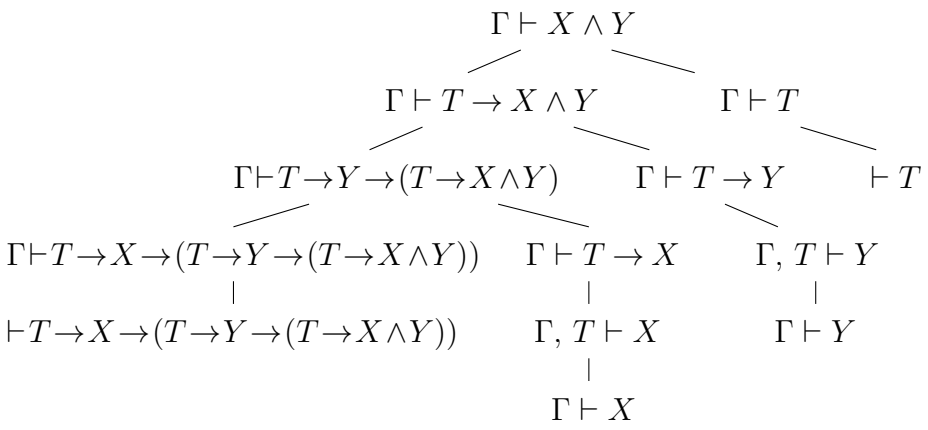
Теорема 2.3. Для любого списка формул Γ и любых формул X, Y, Z верны следующие утверждения:

1) $\dfrac{\Gamma, Y \vdash X}{\Gamma \vdash Y \to X}$ (введение импликации, теорема о дедукции);
2) $\dfrac{\Gamma \vdash X;\ \Gamma \vdash X \to Y}{\Gamma \vdash Y}$ (удаление импликации, modus ponens);
3) $\dfrac{\Gamma \vdash X;\ \Gamma \vdash Y}{\Gamma \vdash X \wedge Y}$ (введение конъюнкции);
4) $\dfrac{\Gamma, X, Y \vdash Z}{\Gamma, X \wedge Y \vdash Z}$ (удаление конъюнкции);
5) $\dfrac{\Gamma \vdash X}{\Gamma \vdash X \vee Y}$ и $\dfrac{\Gamma \vdash X}{\Gamma \vdash Y \vee X}$ (введение дизъюнкции);
6) $\dfrac{\Gamma, X \vdash Z;\ \Gamma, Y \vdash Z}{\Gamma, X \vee Y \vdash Z}$ (удаление дизъюнкции, правило разбора случаев);
7) $\dfrac{\Gamma, X \vdash Y;\ \Gamma, X \vdash \neg Y}{\Gamma \vdash \neg X}$ (введение отрицания);
8) $\dfrac{\Gamma \vdash \neg\neg X}{\Gamma \vdash X}$ (удаление отрицания).

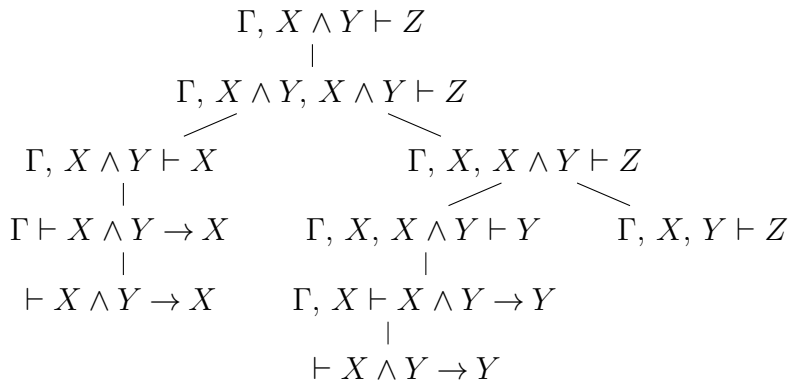
◀ Первые два правила — это теорема о дедукции и правило modus ponens в применении к выводу из гипотез. В самом деле, объединив выводы из Γ формул X и $X \to Y$, получим последовательность, в которой есть эти формулы. Применив modus ponens, получим Y, которую добавим в конец последовательности формул. Получим вывод Y из Γ.

Для доказательства правила 3 для произвольно заданных формул X и Y используем какую-либо аксиому T и аксиому $T \to X \to (T \to Y \to (T \to X \wedge Y))$ схемы 5. Из выводимости формул X

и Y из списка Γ следует выводимость формул $T \to X$ и $T \to Y$. Дважды удаляя импликацию в аксиоме схемы 5, получим выводимость $T \to X \wedge Y$. Еще раз удаляя импликацию (T выводима), получаем выводимость $X \wedge Y$. Указанную последовательность шагов можно представить в виде дерева

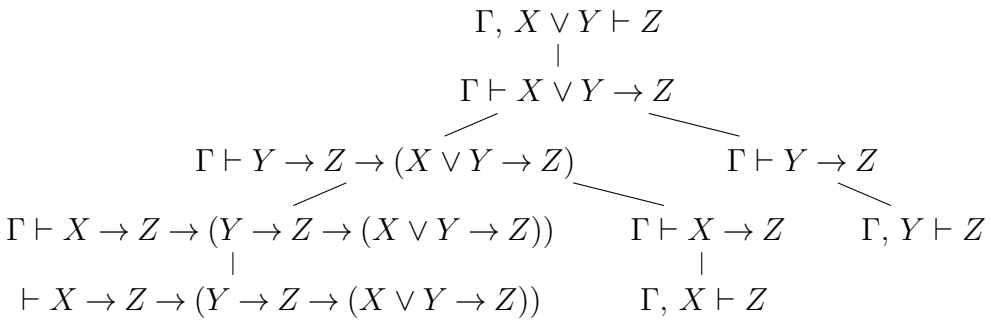


Правило 4 опирается на аксиомы $X \wedge Y \to X$ и $X \wedge Y \to Y$. Из этих аксиом по правилу modus ponens получаем выводимости $X \wedge Y \vdash X$ и $X \wedge Y \vdash Y$. Остается соединить два этих вывода с выводом $\Gamma, X, Y \vdash Z$ для получения нужного вывода $\Gamma, X \wedge Y \vdash Z$. Соответствующее дерево вывода имеет следующий вид:

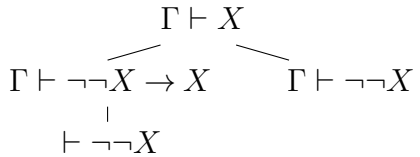


(здесь использовано упрощенное правило 2 вида $\dfrac{\Gamma \vdash X \to Y}{\Gamma, X \vdash Y}$). Аналогично доказывается правило 5, но с использованием аксиом $X \to X \vee Y$ и $X \to Y \vee X$, полученных по схемам 6 и 7.

Правило 6 опирается на аксиому $X \to Z \to (Y \to Z \to (X \vee Y \to Z))$, полученную по схеме 8 и выводится аналогично правилу 5:



Правило 8 опирается на схему аксиом 10 и доказывается следующим деревом:



Отметим, что из схемы 11 вытекает правило, обратное к правилу 8: $\dfrac{\Gamma \vdash X}{\Gamma \vdash \neg\neg X}$.

Остановимся на правиле 7, которое вытекает из схем 9, 10, 11. Непосредственно из схемы аксиом 9 вытекает более слабое правило $\dfrac{\Gamma, X \vdash Y;\ \Gamma \vdash \neg Y}{\Gamma \vdash \neg X}$:

$$\begin{array}{c} \Gamma \vdash \neg X \\ \hline\hline \Gamma \vdash \neg Y \to \neg X \qquad \Gamma \vdash \neg Y \\ \hline\hline \Gamma \vdash X \to Y \to (\neg Y \to \neg X) \qquad \Gamma \vdash X \to Y \\ | \\ \vdash X \to Y \to (\neg Y \to \neg X) \end{array}$$

Чтобы получить нужное правило, необходимо в список Γ добавить аксиому X. Получим правило $\dfrac{\Gamma, X, X \vdash Y;\ \Gamma, X \vdash \neg Y}{\Gamma, X \vdash \neg X}$, или после сокращения одинаковых гипотез: $\dfrac{\Gamma, X \vdash Y;\ \Gamma, X \vdash \neg Y}{\Gamma, X \vdash \neg X}$. Остается обосновать правило $\dfrac{\Gamma, X \vdash \neg X}{\Gamma \vdash \neg X}$, для чего используется схема аксиом 11. Кроме того, как и в случае правила 5 используется какая-либо аксиома T. Эта аксиома вводится в список гипотез, а затем дважды применяется слабый вариант доказываемого правила:

$$\begin{array}{c} \Gamma \vdash \neg X \\ \hline\hline \Gamma, X \vdash \neg T \qquad\qquad \Gamma \vdash \neg\neg T \\ \hline\hline \Gamma, X, T \vdash X \qquad \Gamma, X \vdash \neg X \qquad\qquad | \\ \qquad\qquad\qquad\qquad\qquad\qquad \vdash \neg\neg T \\ \hline\hline \qquad\qquad\qquad\qquad\qquad\qquad \vdash T \qquad \vdash T \to \neg\neg T \end{array} \tag{2.1}$$

Теорема полностью доказана. ▶

Как видно по комментариям в формулировке теоремы, логические правила естественного вывода есть двух видов: правила введения, которые позволяют доказывать формулу с данной логической связкой, и правила удаления, которые показывают, как использовать эту формулу для доказательства других формул.

Правила естественного вывода (в первую очередь, теорема о дедукции) позволяют устанавливать выводимость в исчислении высказываний тех или иных формул, не строя конкретного вывода. Достаточно опираться на установленные правила и, как показано, в доказательстве теоремы, строить вывод, но уже других формул — секвенций. Отметим, что набор правил естественного вывода самодостаточен: в этих правилах учтены все схемы аксиом. При установлении истинности секвенций нет нужды прибегать непосредственно к аксиомам. Как станет ясно из дальнейшего изложения, выводимость формулы X равносильна тому, что секвенция $\vdash X$ выводима из правил естественного вывода.

На практике при установлении истинности конкретных секвенций удобно использовать дополнительные правила, вытекающие из правил естественного вывода. Укажем некоторые простейшие следствия правил естественного вывода.

Пример 2.2. а. Правило $\dfrac{\Gamma \vdash X}{\Gamma \vdash Y \to X}$, называемое присоединением посылки, вытекает из правила введения импликации и правила добавления гипотез:

$$\begin{array}{c} \Gamma \vdash Y \to X \\ | \\ \Gamma, Y \vdash X \\ | \\ \Gamma \vdash X \end{array}$$

б. Правило $\dfrac{\Gamma \vdash X;\ \Gamma \vdash \neg X}{\Gamma \vdash Y}$ (закон противоречия) вытекает из правила введения отрицания добавлением гипотез:

$$\begin{array}{c} \Gamma \vdash Y \\ | \\ \Gamma \vdash \neg\neg Y \\ \diagup \quad \diagdown \\ \Gamma, \neg Y \vdash X \qquad \Gamma, \neg Y \vdash \neg X \\ | \qquad\qquad | \\ \Gamma, \neg Y \vdash X \qquad \Gamma, \neg Y \vdash \neg X \\ | \qquad\qquad | \\ \Gamma \vdash X \qquad\qquad \Gamma \vdash \neg X \end{array}$$

в. Правило введения двойного отрицания $\dfrac{\Gamma \vdash X}{\Gamma \vdash \neg\neg X}$ может быть получено из схемы аксиом 11. Его можно также вывести из правил естественного вывода:

$$\begin{array}{c} \Gamma \vdash \neg\neg X \\ \diagup \quad \diagdown \\ \Gamma, \neg X \vdash X \qquad \Gamma, \neg X \vdash \neg X \\ | \\ \Gamma \vdash X \end{array}$$

г. Полезно правило $\dfrac{\Gamma, X \vdash Y}{\Gamma, \neg Y \vdash \neg X}$ доказательства от противного, которого вытекает из правила введения отрицания:

$$\begin{array}{c} \Gamma, \neg Y \vdash \neg X \\ \diagup \quad \diagdown \\ \Gamma, X, \neg Y \vdash Y \qquad \Gamma, X, \neg Y \vdash \neg Y \\ | \\ \Gamma, X \vdash Y \end{array}$$

д. Правила $\dfrac{\Gamma \vdash X \wedge Y}{\Gamma \vdash X}$ и $\dfrac{\Gamma \vdash X \wedge Y}{\Gamma \vdash Y}$ удаления конъюнкции справа вытекают из правила удаления конъюнкции (но также могут быть получены из схем аксиом 6 и 7). Например, первое из этих правил получается применением правила сечения и правила удаления конъюнкции:

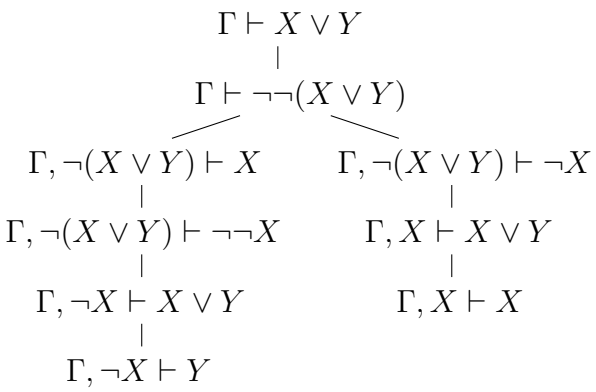
$$\begin{array}{c} \Gamma \vdash X \\ \diagup \quad \diagdown \\ \Gamma, X \wedge Y \vdash X \qquad \Gamma \vdash X \wedge Y \\ | \\ \Gamma, X, Y \vdash X \end{array}$$

Второе правило удаления конъюнкции справа можно получить аналогично. #

Правила естественного вывода и их следствия позволяют доказать выводимость широкого круга секвенций. Однако отметим, что наиболее тяжело устанавливаются выводимости вида $\Gamma \vdash X \vee Y$. Правило введения дизъюнкции сводит задачу к выводу более сильного утверждения $\Gamma \vdash X$ или $\Gamma \vdash Y$.

Пример 2.3. На практике для доказательства утверждения вида $X \vee Y$ можно рассуждать так: „пусть X не верно, т.е. истинно $\neg X$. Докажем, что тогда истинно Y.“ Этому варианту рассуждений соответствуют правила $\dfrac{\Gamma, \neg X \vdash Y}{\Gamma \vdash X \vee Y}$ и $\dfrac{\Gamma, \neg Y \vdash X}{\Gamma \vdash X \vee Y}$, которые можно получить и в исчислении высказываний. Для этого достаточно использовать правило введения отрицания и

правило доказательства от противного:



Это правило назовем ***обобщенным правилом введения дизъюнкции***.

2.3. Глобальные свойства теории N

Выводимые формулы и тавтологии. Непротиворечивость теории $\boldsymbol{N}$. Полнота теории $\boldsymbol{N}$. Разрешимость теории $\boldsymbol{N}$. Независимость аксиом в теории $\boldsymbol{N}$.

Остановимся на описании множества выводимых формул в теории $\boldsymbol{N}$. Нетрудно убедиться в том, что все аксиомы этой теории являются тавтологиями . Например, любая аксиома, полученная по схеме 1, может быть получена подстановкой формул в тавтологию $x \to (y \to x)$, а потому сама является тавтологией (следствие 1.1).

Если формулы X и $X \to Y$ являются тавтологиями, то правило modus ponens приводит к формуле Y, также являющейся тавтологией (теорема 1.1). Это означает, что в любом выводе все формулы являются тавтологиями, поскольку либо являются аксиомами, либо получены из тавтологий по правилу modus ponens (аккуратное доказательство может быть проведено методом математической индукции по длине вывода). Можно также утверждать, что если в списке Γ все формулы — тавтологии, то и все формулы, выводимые из гипотез Γ, также будут тавтологиями.

Теорема 2.4. Любая формула, выводимая в теории $\boldsymbol{N}$, является тавтологией. #

Итак, множество формул, выводимых в теории $\boldsymbol{N}$ содержится в множестве всех тавтологий. На самом деле оба множества совпадают. Чтобы это доказать, установим одно вспомогательное утверждение. Как и в случае булевых функций для произвольной пропозициональной переменной x введем обозначение x^σ, $\sigma \in \mathbb{B}$, полагая, что $x^1 = x$ и $x^0 = \neg x$.

Теорема 2.5. Пусть Y — формула, построенная на переменных $x_1, \ldots, x_n$, $f(\xi_1, \ldots, \xi_n)$ — истинностная функция формулы Y, $\alpha_i \in \mathbb{B}$, $i = \overline{1, n}$, — набор истинностных значений для переменных $x_1, \ldots, x_n$. Если $f(\alpha_1, \ldots, \alpha_n) = 1$, то имеет место выводимость $x_1^{\alpha_1}, \ldots, x_n^{\alpha_n} \vdash Y$. Если $f(\alpha_1, \ldots, \alpha_n) = 0$, то имеет место выводимость $x_1^{\alpha_1}, \ldots, x_n^{\alpha_n} \vdash \neg Y$.

◀ Доказательство строится индукцией по построению формулы. В этом случае базис индукции относится к элементарным формулам, т.е. к переменным. Пусть $Y = x$ и задано истинностное значение α. При $\alpha = 1$ имеем $x^\alpha = x$, $Y = x$, и севенция $x^\alpha \vdash Y$ истинна, поскольку и левая, и правая части секвенции — одна и та же формула. При $\alpha = 0$ имеем $x^\alpha = \neg x$, $\neg Y = \neg x$, и истинной является секвенция $x^\alpha \vdash \neg Y$.

Чтобы доказать шаг индукции, необходимо в предположении, что утверждение верно для формул Z и W, установить его истинность для формул $(Z \vee W)$, $(Z \wedge W)$, $(Z \to W)$ и $(\neg Z)$. Пусть $z_1, \ldots, z_n$ — общий список переменных формул Z и W.

Рассмотрим случай $Y = Z \wedge W$. Для заданного набора α значений $\alpha_1, \ldots, \alpha_n$ введем обозначение $\Gamma = x_1^{\alpha_1}, \ldots, x_n^{\alpha_n}$. Предположим, что на наборе α истинностные функции f_Z и

f_W принимают значение 1. Тогда по индуктивному предположению имеем $\Gamma \vdash Z$ и $\Gamma \vdash W$, откуда по правилу введения конъюнкции получаем выводимость $\Gamma \vdash Y$, так как $Y = Z \wedge W$. Пусть одна из истинностных функций, например f_Z, принимает значение 0. Тогда имеет место выводимость $\Gamma \vdash \neg Z$. Используя правила естественного вывода, получаем

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \neg(Z \wedge W)}}{\Gamma, Z \wedge W \vdash \neg Z}\quad \dfrac{\dfrac{}{\Gamma \vdash \neg(Z \wedge W)}}{\Gamma, Z \wedge W \vdash Z}}{\dfrac{\Gamma, Z, W \vdash \neg Z}{\Gamma \vdash \neg Z}} \qquad \dfrac{\Gamma, Z, W \vdash Z}{}$$

В результате получена выводимость $\Gamma \vdash \neg Y$, поскольку $\neg Y = \neg(Z \wedge W)$.

Рассмотрим случай $Y = Z \vee W$. Опять выберем набор значений α и введем, как и выше, обозначение Γ. Если одна из функций f_Z f_W на наборе α имеет значение 1, например $f_Z(\alpha) = 1$, то по индуктивному предположению имеет место выводимость $\Gamma \vdash Z$. Отсюда сразу получаем выводимость $\Gamma \vdash Z \vee W = Y$ в силу правила введения дизъюнкции. Осталось рассмотреть вариант $f_Z(\alpha) = f_W(\alpha) = 0$. В этом варианте $f_Y(\alpha) = f_Z(\alpha) \vee f_W(\alpha) = 0$ и, значит, нужно установить выводимость $\Gamma \vdash \neg Y$ исходя из индуктивных предположений $\Gamma \vdash \neg Z$ и $\Gamma \vdash \neg W$. Имеем:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \neg(Z \vee W)}}{\Gamma, Z \vee W \vdash U}\quad \dfrac{\dfrac{}{\Gamma \vdash \neg(Z \vee W)}}{\Gamma, Z \vee W \vdash \neg U}}{\Gamma, Z \vdash U \quad \Gamma, W \vdash U} \qquad \Gamma, Z \vdash \neg U \quad \Gamma, W \vdash \neg U$$

В результате требуемая секвенция оказадась сведена к четырем однотипным севенциям, в которых выбор формулы U (соответственно $\neg U$) не имеет значения. Истинность всех четырех севенций устанавливается по единому сценарию с помощью закона противоречия. Рассмотрим, например, первую из них:

$$\dfrac{\dfrac{}{\Gamma, Z \vdash U}}{\Gamma, Z \vdash Z \qquad \dfrac{\Gamma, Z \vdash \neg Z}{\Gamma \vdash \neg Z}}$$

Рассмотрим случай $Y = Z \to W$. Выделим вариант $f_W(\alpha) = 1$. В этом варианте $f_Y(\alpha) = 1$ и нужно установить выводимость $\Gamma \vdash Z \to W$ исходя из $\Gamma \vdash W$. Это верно в силу правила присоединения посылки. Пусть $f_Z(\alpha) = 1$, $f_W(\alpha) = 0$. Тогда $f_Y(\alpha) = 0$, и нужно установить выводимость $\Gamma \vdash \neg(Z \to W)$ исходя из $\Gamma \vdash Z$ и $\Gamma \vdash \neg W$. Это можно сделать следующим образом:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \neg(Z \to W)}}{\Gamma, Z \to W \vdash W}\quad \dfrac{\dfrac{}{\Gamma \vdash \neg(Z \to W)}}{\Gamma, Z \to W \vdash \neg W}}{\Gamma, Z \to W \vdash Z \quad \Gamma, Z \to W \vdash Z \to W} \qquad \dfrac{\Gamma, Z \to W \vdash \neg W}{\Gamma \vdash \neg W}$$
$$\dfrac{\Gamma, Z \to W \vdash Z}{\Gamma \vdash Z}$$

Пусть $f_Z(\alpha) = 0$, $f_W(\alpha) = 0$. Тогда $f_Y(\alpha) = 1$, и требуется установить $\Gamma \vdash Z \to W$ исходя из $\Gamma \vdash \neg Z$ и $\Gamma \vdash \neg W$. Из первой секвенции добавлением гипотезы получаем $\Gamma, Z \vdash \neg Z$, что с очевидной секвенцией $\Gamma, Z \vdash Z$ по закону противоречия приводит к выводимости $\Gamma, Z \vdash W$ и далее по правилу введения импликации к $\Gamma \vdash Z \to W$.

Рассмотрим случай $Y = \neg Z$. Если $f_Z(\alpha) = 1$, то $f_Y(\alpha) = 0$ и $\Gamma \vdash Z$. Введением отрицания получаем $\Gamma \vdash \neg\neg Z = \neg Y$. Аналогично при $f_Z(\alpha) = 0$ имеем $f_Y(\alpha) = 1$ и $\Gamma \vdash \neg Z$, что равносильно $\Gamma \vdash Y$. ▶

Теорема 2.6. Любая тавтология Y выводима в теории $\boldsymbol{N}$.

◀ Пусть формула Y построена из переменных $x_1, \ldots, x_n$. Поскольку Y — тавтология, ее истинностная функция на любом булевом векторе α принимает значение 1. Значит, для любых значений $\alpha_1, \ldots, \alpha_n$ имеет место выводимость $x_1^{\alpha_1}, \ldots, x_n^{\alpha_n} \vdash Y$.

Обозначим $\Gamma_2 = x_2^{\alpha_1}, \ldots, x_n^{\alpha_n}$. Имеют место выводимости $x_1, \Gamma_2 \vdash Y$ и $\neg x_1, \Gamma_2 \vdash Y$. По правилу удаления конъюнкции находим $x_1 \vee \neg x_1, \Gamma_2 \vdash Y$. Используя выводимость $\vdash x_1 \vee \neg x_1$, по правилу сечения получаем $\Gamma_2 \vdash Y$.

Повторяя рассуждения, последовательно убираем из списка гипотез переменные x_2, x_3 и т.д. В конечном счете получим выводимость $\vdash Y$, что равносильно утверждению теоремы. ▶

Полученное описание выводимых в теории $\boldsymbol{N}$ формул позволяет установить такие свойства этой теории, как непротиворечивость, полноту и разрешимость. Установление этих свойств — цель, преследуемая при формализации любой теории.

Под непротиворечивостью формальной теории понимают отсутствие в этой теории такой формулы X, для которой выводимыми являются и сама формула X, и ее отрицание $\neg X$. Иногда дают такое определение непротиворечивости теории: теория непротиворечива, если в ней существует невыводимая формула. Формально утверждение о выводимости любой формулы сильнее, чем утверждение о выводимости какой либо пары X и $\neg X$ (из первого утверждения очевидно вытекает второе). Однако в данном случае оба подхода дают одно и то же: если в теории выводимы и X, и $\neg X$, то по закону противоречия выводима любая формула.

Теорема 2.7. Теория $\boldsymbol{N}$ является непротиворечивой.

◀ Выводимыми в теории $\boldsymbol{N}$ являются только тавтологии. Если формула X выводима в теории $\boldsymbol{N}$, то X — тавтология. Но тогда $\neg X$ является противоречием и не относится к числу выводимых формул. ▶

Под полнотой формальной теории понимают свойство, согласно которому не существует такой невыводимой формулы X, добавление которой к аксиомам приводит к непротиворечивой теории. Другими словами, в такой теории Для каждой формулы X выводима либо сама формула, либо ее отрицание. Наша теория не полна в этом смысле. Действительно, если, добавив в качестве аксиомы некоторую формулу X, мы найдем вывод формулы $\neg X$, то, с иной точки зрения, мы получим вывод $\neg X$ из гипотезы X в теории $\boldsymbol{N}$, т.е. $X \vdash \neg X$. Значит, согласно дереву (2.1), имеет место выводимость $\vdash \neg X$. Однако, как доказано, такое возможно только, если X — противоречие. Следовательно, добавление любой формулы, не являющейся тавтологией или противоречием, дает непротиворечивую теорию.

Впрочем, если в теорию ввести дополнительную схему аксиом, построенную на формуле, не являющейся тавтологией, то подставляя в схему вместо символов подстановки определенные формулы, можно получить противоречие. Это противоречие, с одной стороны, является аксиомой, поскольку получено из схемы аксиом, а с другой стороны, его отрицание есть тавтология, выводимая в теории $\boldsymbol{N}$. Пусть например, схема аксиом построена на формуле, представленной истинностной функцией $f(x_1, \ldots, x_n)$. Так как формула — не тавтология, существует набор значений $\alpha_1, \ldots, \alpha_n$ аргументов функции, на котором она принимает значение 0. Каждую переменную x_j заменим формулой $x_j \vee \neg x_j$, если $\alpha_j = 1$, и формулой $x_j \wedge \neg x_j$. Полученная формула, построенная по схеме аксиом, будет противоречием.

При построении исчисления высказываний (и далее исчисления предикатов) различают полноту в широком смысле, означающую сказанным выше, и полноту в узком смысле, означающую, что выводимыми являются все тавтологии. В узком смысле теория $\boldsymbol{N}$ полна. Впрочем, отсутствие полноты в широком смысле легко устраняется простой модификацией формальной теории: достаточно вместо каждой схемы аксиом записать обычную аксиому, заменив места подстановки переменными и добавить еще одно правило вывода, заключающееся в подстановке вместо переменных произвольных формул. Отметим, что свойством полноты обладают очень

простые математические теории: согласно теореме Геделя о неполноте любая достаточно содержательная теория не является полной, поскольку в рамках такой теории удается построить такую формулу Z, что ни эта формула, ни ее отрицание не являются выводимыми.

Под разрешимостью теории понимают наличие алгоритма, который для любой формулы позволяет установить за конечное число элементарных операций, выводима эта формула или нет. В данном случае в качестве такого алгоритма может рассматривать процедуру построения истинностной функции и проверки ее на наличие значений 0. Отсутствие таких значений означает, что функция является постоянной, а формула — тавтологией, т.е. выводимой в теории $\boldsymbol{N}$. Указанная процедура выполняется за конечное число алгебраических операций и дает однозначный ответ, выводима формула или нет. В этом смысле теория $\boldsymbol{N}$ является разрешимой теорией.

Еще один вопрос, связанный с построением формальной теории — независимость аксиом. Аксиома формальной теории не зависит от остальных аксиом этой теории, если эта аксиома не является выводимой формулой в теории, которая получается из исходной удалением указанной аксиомы. Если ни одна из аксиом формальной теории не является логическим следствием остальных, то говорят о ***независимой системе аксиом***.

Система аксиом теории $\boldsymbol{N}$ является независимой в том смысле, что удалив одну из схем аксиом теории, мы не сможем вывести ни одну из формул, получаемой в рамках этой схемы аксиом. Как можно доказать утверждения подобного рода? Напомним, что в формальной теории формулы теряют всякий содержательный смысл; например, знак импликации может означать какую угодно бинарную операцию на множестве возможных значений переменных, а сами переменные могут быть связаны с любыми математическими объектами. Придание смысла символам формальной теории называют интерпретацией этой теории. В данном случае множество всех высказываний как область изменения переменных и естественный логический смысл символов операций — это одна из возможных интерпретаций исчисления $\boldsymbol{N}$.

Интерпретация формальной теории строится в рамках какой-либо другой математической теории. Надежность интепретации определяется надежностью той теории, в которой интепретируется исчисление. Так, суждения о непротиворечивости, полноте и разрешимости исчисления $\boldsymbol{N}$ получены в рамках так называемой теории булевых функций, т.е. теории функций с областью изменения переменных и функций 0 и 1. В этом смысле подобные суждения носят относительный характер. В математике считают абсолютно надежными теории, связанные с конечными множествами. В этом смысле утверждения о непротиворечивости, полноте и разрешимости исчисления $\boldsymbol{N}$ являются абсолютными. Утверждение о независимости системы аксиом тоже будет абсолютным, если оно будет получено на базе какой-либо конечной интерпретации.

Теорема 2.8. Схемы аксиом исчисления $\boldsymbol{N}$ не зависят друг от друга.

◀ Наиболее просто доказать независимость схем, начиная с 3-й, так как все эти схемы используют, кроме импликации, еще одну операцию, которая участвует лишь в двух других схемах аксиом. Можно ограничиться двухэлементной областью изменения переменных и стандартной интепретацией операций, кроме одной. В табл. 2.1 приведены схемы аксиом и новая интерпретация одной из операций. При этой интерпретации указанная схема аксиом при некоторых значениях входящих в нее формул принимает значение 0, в то время как остальные аксиомы

Таблица 2.1

№	Схема аксиом	Интерпретируемая операция	№	Схема аксиом	Интерпретируемая операция
3	$X \wedge Y \to X$	$x \wedge y = y$	8	$X \to Z \to (Y \to Z \to (X \vee Y \to Z))$	$x \vee y = 1$
4	$X \wedge Y \to Y$	$x \wedge y = x$	9	$X \to Y \to (\neg Y \to \neg X)$	$\neg x = x$
5	$X \to Y \to (X \to Z \to (X \to Y \wedge Z))$	$x \wedge y = 0$	10	$\neg\neg X \to X$	$\neg x = 1$
6	$X \to Y \vee Z$	$x \vee y = z$	11	$X \to \neg\neg X$	$\neg x = 0$
7	$Y \to Y \vee Z$	$x \vee y = y$			

имеют постоянное истинностное значение 1. Изменение интерпретации не затрагивает правила modus ponens, так что при удалении указанной схемы аксиом мы получаем теорию, в которой все выводимые формулы — тавтологии, но некоторые формулы, получаемые из указанной схемы, выводимыми не являются. Это и означает, что указанная схема аксиом не зависит от остальных.

Независимость первых двух схем доказать сложнее, поскольку они касаются импликации, затрагивающей все схемы. Ответ можно найти, рассмотрев в качестве области изменения переменных множество из трех целых чисел $\{0,\ 1,\ 2\}$ и выбрав следующие интерпретации операций: $x \wedge y = \min\{x,\ y\}$, $x \vee y = \max\{x,\ y\}$, $\neg x = 2 - x$. Для импликации потребуем выполнения условия, что $x \to y = 2$ тогда и только тогда, когда $x \leqslant y$. При поставленных условиях схемы 3, 4, 6, 7, 10, 11 будут давать формулы с тождественным значением 2, а в силу условия, наложенного на импликацию, значение Y будет тождественное 2 при $X \equiv 2$ и $X \to Y \equiv 2$. Недоопределенность импликации можно использовать для получения нужных значений первых двух схем аксиом и для обеспечения тождественного значения 2 схем 5, 8, 9.

Положим $x \to y = 0$ при $x > y$. Тогда схема $X \to (Y \to X)$ будет иметь значение 0 при $X = 1$, $Y = 2$, в то время как остальные аксиомы будут давать тождественное значение 2. Действительно, в схеме 2 исключаем случай $X > Y$ или $X \leqslant Z$, так как тогда она имеет вид $0 \to W$ или $W \to 1$. Значит, $Z < X \leqslant Y$ и схема имеет вид $2 \to (X \to 0 \to 0)$. Но $X > Z \geqslant 0$, значит, $X \to 0 = 0$ и $2 \to (X \to 0 \to 0) = 2 \to (0 \to 0) = 2 \to 2 = 2$. В схеме 5 исключаем случай $X > Y$ или $X > Z$, когда она имеет значение 2. Но тогда $X \leqslant \min\{Y,\ Z\}$ и $X \to Y \vee Z = 2$. Поэтому $X \to Y \to (X \to Z \to (X \to Y \wedge Z)) = X \to Y \to (X \to Z \to 2) = 2$. В схеме 8 исключаем случай $X > Z$ или $Y > Z$. Тогда $\max\{X,\ Y\} \leqslant Z$, $X \vee Y \to Z = 2$ и схема 8 имеет тождественное значение 2. В схеме 9 исключаем случай $X > Y$. Но тогда $X \leqslant Y$, $\neg Y \leqslant \neg X$ и формула имеет вид $2 \to 2$.

Положим $x \to y = 1$ при $x > y$. Тогда схема 2 будет иметь значение 1, например, при $X = 1$, $Y = 2$, $Z = 0$. В схеме 1 исключаем случай $Y \leqslant X$. Тогда $X < Y \leqslant 2$, а значит, $X \leqslant 1$ и $X \to (Y \to X) = X \to 1 = 2$. Схемы 5, 8, 9 проверяются так же, как выше. ▶

3. АЛГЕБРА ПРЕДИКАТОВ

3.1. Предикаты и кванторы

Ограниченность исчисления высказываний. Переменные высказывания. Субъект и предикат переменного высказывания. Понятие предиката в математической логике. Арность предиката. Контуры будущей теории. Кванторы как символы агрегирования. Выражения двух видов.

Построенное исчисление высказываний в реальности оказывается довольно бедной теорией. Оно позволяет строить и анализировать различные комбинации высказываний. Однако в математике встречаются предложения, которые, вообще говоря, не являются высказываниями. Например, нельзя сказать, истинно ли утверждение „число x положительное“. Ответ зависит от того, какое это число. Здесь мы сталкиваемся с уточнением того, что такое суждение. В каждом суждении присутствуют два компонента — субъект и предикат. Субъект — это понятие, отражающее предмет, о котором идет речь. В предложении субъект связывается с подлежащим. Предикат же можно уподобить сказуемому, поскольку этот структурный элемент выражает свойства, приписываемые предметам или отрицаемые у них. В суждениях „Роза красная“, „Гитара семиструнная“ субъектами являются „роза“ и „гитара“, а предикатами — „красная“ (все то, что называется красным) и „семиструнная“ (все то, что называется семиструнным).

В математике этой трактовке суждения отвечает „высказывание с параметрами“, т.е. суждение, которое становится высказыванием при конкретизации параметров. Такие суждения называют предикатами. С математической точки зрения предикат можно интерпретировать как некое отображение, которое каждому набору входящих в него параметров ставит в соответствие высказывание. Количество участвующих в предикате параметров называется его ***арностью***.

Понятие предиката не вкладывается в язык исчисления высказываний. Требуется построение более широкой теории, требующей и расширения языка теории. Что должен охватывать такой язык помимо высказываний и логических связок?

Во-первых, если в формулах алгебры высказываний встречаются переменные, областью действия которых является множество высказываний, то в формулах любой математической теории есть переменные, область действия которых связана не с высказываниями, а с объектами, изучаемыми в этой теории. В математическом анализе переменные числовые, в геометрии переменные обозначают точки, прямые, плоскости. Такие переменные называют ***предметными***, а область их действия — ***предметной областью***. В теории могут участвовать предметные переменные разного рода, поскольку описывают объекты разных классов (например, точки и прямые в геометрии).

Во-вторых, в ряде математических теорий есть установленные символы для обозначения некоторых стандартных объектов (например, число π). В отличие от переменных эти символы имеют конкретное значение. Их называют константами.

В третьих, в каждой математической теории используются функции, причем разные, к которым можно отнести и операции над объектами математической теории. Функции можно рассматривать двояко: как отображение и как связь между переменными. С точки зрения формальной записи функция — это символ, который исходя из некоторой совокупности переменных формирует новую переменную (точнее, новый объект, имеющий неопределенное значение).

Наконец, в четвертых, в этом языке должны быть отражены и такие символы, как кванторы. Кванторы — специальные символы (или фразеологические штампы), которые позволяют

из неопределенного предложения построить правильное высказывание, истинность которого устанавливается однозначно. Кванторы можно рассматривать как символы агрегирования, которые из совокупности возможных значений формируют одно значение. Таковы символы предела, интеграла, суммы. В логике используют два квантора: всеобщности $\forall$ и существования $\exists$. Запись $\forall x P(x)$ формирует высказывание, являющееся истинным, если предикат $P(x)$ является истинными для любого возможного значения предметной переменной x. Запись $\exists x P(x)$ формирует высказывание, истинное, если предикат $P(x)$ имеет значение истины хотя бы для одного возможного значения предметной переменной x.

Введение в язык символов агрегирования приводит к тому, что вхождение переменных в формулы может играть две разных роли. В формуле $x + y$ переменным x и y можно присвоить конкретные значения, в результате чего конкретное значение получит все выражение. В формуле $\forall x P(x)$ переменной x нельзя присвоить конкретное значение, поскольку формула подразумевает использование целого множества значений переменной, а не отдельно взятого значения (поэтому и квантор — символ агрегирования). В первом случае говорят о ***свободном вхождении переменной***, а во втором — о ***связанном вхождении***. Одна и та же переменная в выражение может входить и свободно, и связанно. Например, в выражении $n + \sum_{n=1}^{3} a_n$ переменная n входит свободно (первое вхождение) и связанно (второе и третье).

В дальнейшем изложении мы не будем использовать предметных символов агрегирования, ограничившись только двумя кванторами, так как предметные символы агрегирования усложняют язык, но не добавляют чего-то существенного в формальную теорию как таковую. Отметим, что „за бортом“ остаются обозначения множеств и связи множеств и элементов. Если в формальной теории нет средств для обозначения множеств, ее называют теорией 1-го порядка. При введении множеств отдельных элементов получаем теорию 2-го порядка, множества множеств приводят к теории 3-го порядка. И так далее.

Отметим, что новая теория будет содержать выражения двух видов. Первые не означают какое-то высказывание, они описывают лишь действия, выполняемые над объектами теории. Например, выражение $x + y$ играет роль новой предметной переменной, значения которой однозначно определяются значениями x и y. Напротив, выражение $x + y = 0$ выражает суждение, которое может быть истинным или ложным при заданных значениях неизвестных. Разделяя два типа выражений, первые будем называть ***термами***, а вторые ***формулами***. Таким образом, значением терма является некоторый объект теории (например, число), в то время как значением формулы является высказывание.

3.2. Логико-математические языки

Язык как совокупность четырех множеств (V, C, F, P). Предметные переменные, константы и их сортность. Функциональные и предикатные символы, их тип. Предикатные символы арности 0 как пропозициональные переменные. Индуктивное построение термов и формул. Функциональная сложность терма, логическая сложность формулы. Выражения. Правильные формулы: $A(x)$, $B(x, y)$, C, $(\forall x A(x))$. Комментарии по вариантам построения логико-математических языков. Односортные языки. Свободные и связанные вхождения переменных. Предложения (замкнутые термы и формулы). Соглашения о сокращенной записи формул.

Под логико-математическим языком мы будем понимать совокупность $(V,\ C,\ F,\ P)$ из четырех не более чем счетных множеств, элементы которых называют ***символами***. Первое множество — это множество V ***предметных переменных***. Предметные переменные могут иметь разные ***сорта***. Конкретно мы считаем, что есть еще пятое множество S — ***множество сортов*** и задано отображение $\mathrm{sort}_V\colon V \to S$. Значение отображения sort_V на переменной $x \in V$ и есть сорт этой переменной. Каждому сорту $\pi \in S$ соответствует множество V_π предметных переменных этого сорта, являющееся полным прообразом $\mathrm{sort}_V^{-1}(\pi)$ элемента $\pi \in S$.

Элементы множества C называются ***константами***. Каждая константа относится к определенному сорту $\pi \in S$, т.е. задано отображение $\mathrm{sort}_C\colon C \to S$ множества констант в множество сортов.

Элементы множества F — это ***функциональные символы***, каждый из которых характеризуется своим типом, определяемым кортежем $(\pi_0, \pi_1, \ldots, \pi_n) \in S^{n+1}$ сортов (целое число $n > 0$ называется ***арностью функционального символа***).

Элементы множества P — это ***предикатные символы***, каждый из которых характеризуется своим типом, определяемым кортежем сортов $(\pi_1, \ldots, \pi_n) \in S^n$ (и в этом случае $n \geqslant 0$ — ***арность предикатного символа***).

Названия элементов четырех множеств указывают на роль, которые они будут играть в содержательной интерпретации языка. Однако этот содержательный смысл не играет роли в формальной теории. Главное, каким образом из символов составляются правильные слова языка.

В логико-математическом языке два вида правильных слов. Это термы и формулы. Начнем с индуктивного определения термов.

Б а з а и н д у к ц и и. Любая переменная или константа сорта π является термом сорта π (элементарные термы).

Ш а г и н д у к ц и и. Если $t_1, t_2, \ldots, t_n$ — термы сортов $\pi_1, \ldots, \pi_n$ соответственно, f — функциональный символ типа $(\pi_0, \pi_1, \ldots, \pi_n)$, то $f(t_1, t_2, \ldots, t_n)$ — терм сорта π_0.

Из этого определения нетрудно понять, что термы служат для записи выражений в предметной области (например, многочленов в полях).

Функциональной сложностью терма называется количество функциональных символов в нем. Индуктивно эту характеристику можно определить так: сложность элементарного терма равна 0; сложность терма $f(t_1, t_2, \ldots, t_n)$ равна сумме сложностей аргументов плюс один, т.е.

$$|f(t_1, t_2, \ldots, t_n)| = |t_1| + t_2| + \ldots + |t_n| + 1,$$

где знак модуля обозначает функциональную сложность терма.

Подобное индуктивное определение может использоваться не только для определения понятий, но и для доказательства различных утверждений. Например, пусть все элементарные термы обладают свойством P; из того, что термы $t_1, t_2, \ldots, t_n$ обладают свойством P, следует, что терм $f(t_1, t_2, \ldots, t_n)$ также обладает свойством P. Тогда все термы обладают свойством P.

Определим теперь формулы.

Б а з а и н д у к ц и и. Если $t_1, t_2, \ldots, t_n$ — термы сортов $\pi_1, \ldots, \pi_n$ соответственно, p — предикатный символ типа $(\pi_1, \ldots, \pi_n)$, то $p(t_1, t_2, \ldots, t_n)$ — формула (такую формулу будем называть ***элементарной***).

Ш а г и н д у к ц и и. Если X_1, X_2 — формулы, то $(X_1 \to X_2)$, $(X_1 \wedge X_2)$, $(X_1 \vee X_2)$, $\neg X_1$ — формулы. Если X — формула, x — предметная переменная, то $\forall x X$ и $\exists x X$ — формулы.

В то время как термы представляют собой записи последовательностей операций над объектами предметной области, формулы обозначают те или иные утверждения о свойствах этих объектов. Совокупность всех термов в рассматриваемом языке обозначим Tm, совокупность всех формул — Fm, а совокупность и тех, и других — Expr. Элементы Expr будем называть ***выражениями***.

Отметим, что допускаются предикатные символы арности 0, которые являются элементарными формулами. Множество таких символов в сочетании с логическими связками образует подмножество языка, эквивалентное языку алгебры высказываний. Поэтому предикатные символы арности 0 можно рассматривать как пропозициональные переменные.

Для формул может быть введена характеристика, называемая ***логической сложностью***, аналогичная функциональной сложности термов, но определяемая количеством предикатных символов в формуле.

Замечание. 1. На самом деле понятие логико-математического языка можно вводить различными способами, не меняя содержательной сути. Так, предикатные символы можно

рассматривать как особый род функциональных символов, для которых сорт π_0 есть „высказывание“. Аналогично можно было бы отказаться от переменных языка, рассматривая их как функциональные символы арности 0. В результате дело можно было бы свести к двум множествам: C и F.

Отметим также, что множество V на самом деле можно трактовать как семейство множеств, в котором каждое множество — это множество переменных одного сорта. Чтобы обозначить сорт переменной, достаточно указать множество, которому эта переменная принадлежит.

Выбор конкретного варианта определения диктуется субъективными факторами, а также тем, что должно присутствовать нечто, трактуемое как сорт высказываний.

2. Совокупность всех переменных, констант, функциональных и предикатных символов плюс логические связки, кванторы, скобки и запятая составляет алфавит логико-математического языка. В данном случае алфавит оказывается счетным множеством. Однако нетрудно дело свести к конечному алфавиту, если считать, что элементы логико-математического языка на самом деле являются словами другого языка (языка нижнего уровня). Так, для всех натуральных чисел можно получить обозначения с помощью двух символов 0 и 1, используя позиционную форму записи. Однако этот механизм находится за пределами интересов математической логики. Поэтому мы и считаем идентификаторы переменных, функций и т.п. простейшими неделимыми элементами языка — символами. #

Среди логико-математических языков выделим ***односортные языки***, в которых все предметные переменные имеют один сорт. Таков язык, например, арифметики. Напротив, язык линейной алгебры является двусортным (числа и векторы).

В формулах $\forall x\, X$ или $\exists x\, X$ первый символ называется ***квантором*** (далее, если несущественно, какой именно квантор, мы будем использовать обозначение ∇). Комбинация из квантора и переменной называется ***кванторной приставкой***, а формула X, следующая за кванторной приставкой — ***областью действия квантора***. Допускается, что переменная кванторной приставки (***кванторная переменная***) вообще не имеет вхождений в области действия этой приставки.

Появление кванторов приводит к тому, что предметные переменные формулы по своему положению могут иметь разные свойства. В языке высказываний была введена такая операция, как подстановка вместо переменных других переменных или формул. Однако при наличии кванторов произвольная подстановка может привести к неправильной формуле. Например, В формуле $\forall x\,(x > 0)$ нельзя переменную x заменить произвольной формулой: слово $\forall(x+1)\,((x+1) > 0)$ не является формулой, потому что по индуктивному правилу образования формул сразу за квантором должна следовать переменная, а не служебный символ или константа. Содержательный смысл формулы при такой подстановке также теряется.

Будем говорить, что вхождение предметной переменной x в формулу X является ***связанным***, если оно есть часть кванторной приставки или находится в области действия кванторной приставки с той же переменной. В противном случае вхождение переменной называется ***свободным***. В терме все вхождения каждой переменной свободные.

В одной и той же формуле переменная x может иметь и свободные, и связанные вхождения. Например, в формуле $p_1(x) \vee \forall x\, p_2(x)$ три вхождения переменной x, причем первое вхождение свободное, а два других связанные. Если переменная x имеет свободные вхождения в формулу X, то ее называют ***свободной переменной*** этой формулы. Отметим, что определенное вхождение переменной может быть связанным в формуле и свободным в некоторой подформуле. Например, предикатный символ, в один из аргументов которого входит переменная, является подформулой, а в элементарной формуле вхождение любой переменной свободное. Еще один пример. В формуле

$$(\forall \boldsymbol{x} p(\boldsymbol{x}, y) \wedge q(x)) \to (\exists \boldsymbol{x} r(\boldsymbol{x}, \boldsymbol{x}))$$

полужирным шрифтом помечены связанные вхождения переменных, а светлым — свободные.

Свободные переменные, входящие в терм или формулу, называются ***параметрами*** этого терма или формулы. Если формула не имеет свободных переменных, то она называется ***замкнутой*** или ***предложением***.

При записи формул будем использовать сокращения скобок на основе приоритета операций. Наивысшим приоритетом обладают кванторные приставки. Затем идет отрицание, далее дизъюнкция и конъюнкция. Наинизшим приоритетом обладает импликация. Мы будем также использовать сокращение $(X \sim Y)$ для формулы $((X \to Y) \wedge (Y \to X))$, причем символу $\sim$ (эквиваленция) припишем самый низкий приоритет (ниже, чем у импликации). Кроме того, некоторые из функциональных символов арности 2, будем употреблять в инфиксной форме (например, символы алгебраических операций). Все эти соглашения используются для более удобной записи и с теоретическими построениями не связаны. От подобных соглашений всегда можно избавиться, модфицировав соответствующим образом терм или формулу. Разумеется, эти элементы можно было бы включить в формальный язык, но это в теоретическом плане ничего существенного не добавит, хотя заметно усложнит изложение.

3.3. Переименования и подстановки

Роль свободных и связанных переменных. Переименование. Коллизия переменных при переименовании. Отношение конгруэнтности ($\approx$). Подстановка, свободная для данного выражения. Сохранение конгруэнтности при подстановке. Свойство чистоты переменных. Лемма о чистоте переменных.

С понятием связанной переменной мы встречаемся уже в курсе математического анализа Обозначения предела, определенного интеграла дают примеры связывания переменных. Из того же курса известно, что связанную переменную можно изменять, не меняя смысла выражения. Однако это в общем верное правило может приводить к неверным результатам. Например, если в выражении $\lim\limits_{x\to 0}\cos(x^2+y^2)$ заменить связанную переменную x переменной y, получим формулу $\lim\limits_{y\to 0}\cos(y^2+y^2)$, имеющую совсем другой смысл. В этом случае говорят, что при подстановке произошла коллизия переменных.

Одновременная замена переменной x в кванторной приставке* ∇x и всех свободных вхождений переменной x в области действия этой приставки на переменную y того же сорта называется ***переименованием***. ***Коллизия переменных при переименовании*** в формуле $\nabla x\, X$ — ситуация, когда при переименовании переменной x в переменную y в X или в одной из ее подформул есть переменная, которая из свободной превращается в связанную. Например, в формуле $\forall x\, p(x,y,z)$ переменные y и z являются свободными. Но при переименовании x в y получим $\forall y\, p(y,y,z)$, где только одна переменная z осталась свободной. Может быть и другая ситуация. В формуле $\forall x\, p_1(x,(\forall y\, p_2(x,y)),z)$ оба вхождения переменной y связанные. Переменная x в формулу входит связанно (раз есть квантор), но в подформулу $\forall y\, p_2(x,y)$ эта переменная уже входит свободно. Но переименование x в y переводит y в разряд связанной переменной в этой подформуле: $\forall y\, p_1(y,(\forall y\, p_2(y,y)),z)$. В этом случае также произошла коллизия переменных.

Можно сказать по-другому. Назовем переменную x в формуле Z ***свободной переменной*** для переменной y, если никакие вхождения y в Z, являющиеся свободными для Z или одной из ее подформул, не попадают в область действия кванторных приставок с переменной x. Коллизия переменных при переименовании — попытка заменить кванторную переменную x на y в то время как x не является свободной для y.

Мы будем говорить, что две формулы X и X' ***конгруэнтны*** (являются вариантами одна другой, обозначение $X \approx X'$), если одна получена из другой с помощью правильного, т.е. без коллизии переменных, переименования. Точное определение этого понятия можно дать индукцией по построению формулы:

1) элементарная формула конгруэнтна только себе самой;

*Напомним, что символ ∇ обозначает любую из двух кванторных приставок.

2) если $Y \approx Y'$, $Z \approx Z'$, то $\neg Y \approx \neg Y'$ и $Y \nabla Z \approx Y' \nabla Z'$, где ∇ — одна из трех двуместных логических связок;

3) если $X = \nabla x\, Y$, $X' = \nabla y\, Z$, то $X \approx X'$, если x и y одного сорта и для любой переменной z того же сорта, вообще не входящей ни в Y, ни Z, имеем $Y_z^x \approx Z_z^y$, где W_u^v есть результат замены в W всех свободных вхождений переменной v на переменную u.

Отметим, что конгруэнтность — это отношение эквивалентности (симметричное, рефлексивное и транзитивное). Таким образом, множество всех формул разделяется на классы попарно конгруэнтных формул. Хотя формально конгруэнтные формулы различаются, нет оснований такое различие учитывать. С математической точки зрения здесь осуществляется переход от понятия „формула“ к более общему понятию „класс конгруэнтных формул“.

Под подстановкой θ мы будем понимать символическую запись вида

$$\theta = \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ t_1 & t_2 & \dots & t_n \end{pmatrix},$$

в которой $x_1, x_2, \dots, x_n$ — конечный набор (предметных) переменных, а $t_1, t_2, \dots, t_n$ — набор термов, причем сорт терма t_i совпадает с сортом переменной x_i. Такую запись мы всегда можем трактовать как запись отображения, определенного на конечном подмножестве множества всех предметных переменных языка, в множество Tm всех термов языка, которое сохраняет сортность термов. Допускается, что приведенная таблица может быть пустой.

Далее через $T\theta$ будем обозначать результат подстановки θ в выражение (т.е. терм или формулу) T, т.е. результат замены в T всех свободных вхождений каждой переменной x_i из подстановки θ соответствующим термом t_i. Отметим, что при этом некоторые переменные x_i могут не входить свободно в T. В этом случае в отношении x_i не происходит никаких замен, соответствующий столбец в подстановке θ можно опустить.

В алгебре высказываний подстановка — один из способов преобразования формулы в эквивалентную. При этом на подстановку не накладывалось никаких ограничений. Ситуация меняется с появлением кванторов. При подстановке в формуле с кванторами смысл формулы может заметно измениться. Поэтому не все подстановки пригодны с точки зрения логики. Например, формула[**] $x + y = z$ с натуральными переменными x, y, z есть предикат от трех переменных. Формула $\exists y\,(x + y = z)$ есть предикат уже от двух переменных x и z, логическим эквивалентом которого является формула $x < z$. Заменим в этой формуле свободную переменную x термом $x \cdot y$, надеясь на то, что получим предикат $x \cdot y < z$. Однако формула $\exists y\,(x \cdot y + y = z)$ имеет совсем другой смысл. Здесь возникла та же проблема, что и при переименовании — коллизия переменных: при замене переменной x термом $x \cdot y$ переменная y, входящая в терм свободно, оказалась в области действия кванторной приставки и стала связанной. Чтобы избежать искажения, достаточно было предварительно „убрать“ из формулы связанную переменную y с помощью переименован ия а уж затем выполнить подстановку: $\exists u\,(x \cdot y + u = z)$. При таком порядке действий действительно получим предикат $x \cdot y < z$.

Подстановку θ назовем ***свободной для выражения*** T, если для любой переменной x_i из области определения $\operatorname{dom}\theta$ подстановки θ (т.е. встречающейся в первой строке матрицы), никакое свободное вхождение x_i в T не попадает в область действия никаких кванторных приставок с переменными, свободно входящими в $t_i = \theta(x_i)$.

В частности, если ни один из параметров формул $\theta(x)$ не является кванторной переменной в T, то θ свободна для T. Это будет так в случае, когда каждая из формул $\vartheta(x)$ вовсе не имеет параметров, т.е. является замкнутой. Такая подстановка называется ***константной***.

Если подстановка θ не является свободной для T, то, как показывает рассмотренный выше пример, можно, не меняя логической сути формулы, выполнить переименование переменных

[**]Отмечу, что любое алгебраическое уравнение является не термом, а формулой: его можно рассматривать как утверждение, касающееся указанных переменных, при этом решение уравнения представляет собой выявление области истинности этой формулы.

в T, переходя к конгруэнтной формуле T', для которой θ уже будет свободной. Этот подход позволяет обеспечить универсальность подстановки, но, правда, пожертвовав однозначностью: результат будет лишь с точностью до когруэнтности.

Лемма 3.1. Если $X' \approx X''$ и переменная u входит в X' свободно, то и в X'' переменная u входит свободно.

◀ Это интуитивно ясное утверждение (переименование не затрагивает свободных переменных) строго доказывается методом индукции по построению формулы. Утверждение очевидно для элементарных формул (конгруэнтность в этом случае равносильна совпадению). Если $X' = U' \vee W'$, $X'' = U'' \vee W''$, то конгруэнтность $X' \approx X''$ означает, что $U' \approx U''$ и $W' \approx W''$. Если, например, u входит свободно в U', то, согласно индуктивному предположению, u входит свободно и в U'', а значит, и в $X'' = U'' \vee W''$. Аналогичны рассуждения для остальных логических связок, включая отрицание.

Пусть $X' = \nabla x U'$, $x'' = \nabla y U''$, где ∇ — один из кванторов. Условие $X' \approx X''$ в данном случае означает, что для переменной z, не входящей в U' и U'', имеем $(U')^x_z \approx (U'')^y_z$. Отметим, что в этом случае $u \neq x$, поскольку x не входит свободно в X'. Следовательно, u входит свободно в $(U')^x_z$. По индуктивному предположению u входит свободно в $(U'')^y_z$, откуда вытекает, что $u \neq y$, так как y не входит свободно в $(U'')^y_z$. Но тогда u входит свободно и в $X'' = \nabla y U''$. ▶

Теорема 3.1. Если $X' \approx X''$, а θ свободна для X' и X'', то $X'\theta \approx X''\theta$.

◀ Доказательство базируется на индукции по построению формулы и на индуктивном определении конгруэнтности. Если $X' \approx X''$, то формулы X либо обе элементарные, либо обе получены с помощью логической связки, либо обе построены присоединением кванторной приставки. В первом случае X' и X'' совпадают, а значит, формулы $X'\theta$ и $X''\theta$ также элементарны и совпадают, т.е. $X'\theta \approx X''\theta$.

Пусть, например, $X' = U' \vee W'$ и $X'' = U'' \vee W''$, причем $U' \approx U''$, $W' \approx W''$. По индуктивному предположению $U'\theta \approx U''\theta$ и $W'\theta \approx W''\theta$. Следовательно, $U'\theta \vee W'\theta \approx U''\theta \vee W''\theta$ и

$$X'\theta = (U' \vee W')\theta = U'\theta \vee W'\theta \approx U''\theta \vee W''\theta = (U'' \vee W'')\theta = X''\theta.$$

Аналогичны рассуждения для других логических связок.

Пусть $X' = \nabla x U'$, $X'' = \nabla y U''$. Выберем переменную z, не входящую ни в формулы U', U'', ни в формулы подстановки θ. Тогда, согласно условию $X' \approx X''$, имеем $(U')^x_z \approx (U'')^y_z$. Согласно индуктивному предположению, $(U')^x_z\theta \approx (U'')^y_z\theta$. Отметим, что переменные x и y не входят свободно в формулы $(U')^x_z$ и $(U'')^y_z$, так как эти формулы конгруэнтны, x не входит свободно в первую из этих формул, y — во вторую. Рассмотрим подстановку $\overline{\theta}$, полученную из θ удалением столбцов с переменными x и y, если они есть. Применение подстановок θ и $\overline{\theta}$ к формулам $(U')^x_z$ и $(U'')^y_z$ дает один и тот же результат, поскольку переменные x и y не входят свободно в эти формулы. Следовательно, $(U')^x_z\overline{\theta} \approx (U'')^y_z\overline{\theta}$. Так как $(U')^x_z\overline{\theta} = (U'\overline{\theta})^x_z$ и $(U'')^y_z\overline{\theta} = (U''\overline{\theta})^y_z$, то $(U'\overline{\theta})^x_z \approx (U''\overline{\theta})^y_z$. В соответствии с определением конгруэнтности заключаем, что $\nabla x(U'\overline{\theta}) \approx \nabla y(U''\overline{\theta})$. Наконец, с учетом равенств $\nabla x(U'\overline{\theta}) = (\nabla x U')\overline{\theta} = (\nabla x U')\theta$ и $\nabla y(U''\overline{\theta}) = (\nabla y U'')\overline{\theta} = (\nabla y U'')\theta$ заключаем, что $X'\theta = (\nabla x U')\theta = (\nabla y U'')\theta = X''\theta$. ▶

Будем говорить, что формула X обладает ***свойством чистоты переменных***, если, во-первых, никакая переменная кванторной приставки не входит в X свободно, и во-вторых, разные кванторные приставки содержат разные переменные.

Следующая лемма позволяет утверждать, что в любой формуле T можно устроить переименование, в результате которого данная подстановка θ станет свободной для переименованной формулы.

Лемма 3.2 (о чистоте переменных). Для любой формулы X и любого множества $S \subset V$ переменных существует формула Y, когруэнтная X, обладающая свойством чистоты переменных, в которой нет связанных переменных, принадлежащих S.

◀ Доказательство проводится индукцией по построению формулы. Для элементарных формул утверждение тривиально, поскольку такие формулы вообще не имеют связанных переменных. Пусть задано множество S и формула $X = Y \nabla Z$, где ∇ — одна из двуместных логических связок. Предположим, что относительно формул Y и Z уже доказано утверждение леммы с произвольным множеством S. Пусть S_X — множество всех переменных (и свободных, и связанных), входящих в X. Существует формула Y', конгруэнтная Y, обладающая свойством чистоты переменных и не имеющая связанных переменных из множества $S \cup S_X$. Обозначим через S_Y множество всех переменных, входящих в Y'. Существует формула X', конгруэнтная X, обладающая свойством чистоты переменных и не имеющая связанных переменных из множества $S \cup S_Y$. Формула $X' \nabla Y'$ обладает свойством чистоты переменных (любая кванторная переменная, например, в X' не совпадает со „своими“ переменными, т.е. другими кванторными переменными формулы X', а по построению и с „чужими“ переменными, составляющими множество S_Y). Кроме того, в построенной формуле нет связанных переменных, попадающих в S. Для одноместной связки $\neg$ утверждение теоремы очевидно.

Пусть $X = \nabla x\, Y$, где ∇ — один из кванторов. Сперва надо „вывести“ новую кванторную переменную за пределы S. Для этого выбираем новую переменную u, не принадлежащую S и не входящую никаким образом в Y. Подстановка $\binom{x}{u}$ свободна и мы получаем формулу $\nabla u\, Y_u^x$, конгруэнтную X. Если мы теперь заменим Y_u^x конгруэнтной формулой Y' со свойством чистоты переменных и без переменных из S, то получим конгруэнтную Y формулу $\nabla u\, Y'$ со свойством чистоты переменных и без переменных из S. ▶

В соответствии с доказанной леммой, чтобы выполнить в X правильную подстановку θ, необходимо переименовать X в X' так, что среди переменных X' не будут встречаться переменные термов $\theta(x)$. Тогда подстановка θ будет свободной для X', и мы можем реализовать эту подстановку.

3.4. Семантика логико-математического языка

Интерпретация логико-математического языка. Предметная область. Интерпретация предметных констант. Интерпретация функциональных и предикатных символов. Интерпретация логических связок и кванторов. Расширение языка введением символов реальных объектов. Оценка и оцененные выражения.

Сами по себе термы и формулы логико-математического языка ничего не значат. Это лишь средство для обозначения или записи реальных объектов. Чтобы термы и формулы приобрели какой-то содержательный смысл, необходимо связать их с реальными объектами из какой-либо области математики, т.е. определить ***интерпретацию логико-математического языка***, или построить ***модель языка***. Для этого мы должны предусмотреть несколько вещей.

Во-первых, необходимо придать смысл предметным переменным, определив их возможные значения. Мы назовем ***носителем*** (или ***предметной областью***) языка Ω отображение, которое каждому сорту $\pi \in S$ ставит в соответствие некоторое непустое множество D_π. Множество D_π представляет собой носитель сорта π или, другими словами, область изменения переменных сорта π.

Во-вторых, следует придать смысл предметным константам. Естественно каждую константу $c \in C$ сорта π связать с некоторым элементом $\widetilde{c} \in D_\pi$ из носителя сорта π, т.е. задать отображение $\widetilde{C}\colon C \to \bigcup\limits_{\pi \in S} D_\pi$, сохраняющее сортность.

В-третьих, нужно каждому функциональному символу f типа $(\pi_0, \ldots, \pi_n)$ поставить в соответствие конкретное отображение $\widetilde{f}\colon D_{\pi_1} \times \ldots \times D_{\pi_n} \to D_{\pi_0}$, т.е. задать некоторое отображение $\widetilde{F}\colon f \mapsto \widetilde{f}$.

В-четвертых, следует каждому предикатному символу p типа $(\pi_1, \ldots, \pi_n)$ поставить в соответствие n-местный предикат того же вида, т.е. отображение, которое каждому элементу

множества $D_{\pi_1} \times \ldots \times D_{\pi_n}$ ставит в соответствие высказывание. Впрочем, поскольку нас интересует лишь истинность формул, можно ограничиться булевым вариантом и символу p поставить в соответствие отображение $\widetilde{p}\colon D_{\pi_1} \times \ldots \times D_{\pi_n} \to \mathbb{B}$. Отдельно упомянем предикатные символы арности 0. Это пропозициональные переменные, которые в случае булевой интерпретации можно заменить двумя булевыми константами.

В-пятых, следует придать смысл всем логическим связкам и кванторам. Мы будем считать, что они имеют общепринятый смысл и что их интерпретация фиксирована.

Все сказанное означает, что интерпретация — предписание, которое символам языка ставит в соответствие настоящие объекты. Говорят, что модель языка определяет его ***семантику***.

Если задана интерпретация M языка Ω, то каждый терм сорта π определяет отображение, которое каждому набору значений переменных ставит в соответствие объект сорта π, т.е. элемент множества D_π. Другими словами, терм получает значение, если известны значения всех его параметров. Значение терма легко определить индукцией по построению.

Обычно придание переменным терма некоторых значений рассматривается как некоторый формальный акт подстановки в терм вместо свободных переменных других объектов. Вообще говоря, использовать в выражении конкретные объекты не очень удобно. Во-первых, это выводит за пределы рассматриваемого языка. Во-вторых, подстановка в выражение „неизвестно чего“ может привести к неоднозначностям или другим казусам, поскольку так называемые „реальные“ объекты вполне могут оказаться словами другого языка.

Указанный формальный подход можно реализовать следующим образом. Введем в обиход дополнительный набор констант, каждая из которых соответствует реальному объекту. Более точно, выберем множество $\underline{C}$ символов, равномощное $D = \bigcup\limits_{\pi\in S} D_\pi$ и биективное отображение $\widehat{C}\colon D \to \underline{C}$, устанавливающее взаимно однозначное соответствие между объектами $a \in D_\pi$ и символами $\underline{a}$ того же сорта. В формулах будем использовать не сами объекты, о сопоставленные им символы. В результате получим некоторое расширение $\Omega(D)$ исходного языка Ω.

Теперь мы можем в рамках расширенного языка выполнить формальную подстановку вида

$$\theta = \begin{pmatrix} x_1 & x_2 & \ldots x_n \\ \underline{a_1} & \underline{a_2} & \ldots \underline{a_n} \end{pmatrix}.$$

Это константная подстановка, называемая ***оценкой языка*** Ω. Если θ содержит все переменные терма T, то терм $T\theta$ оказывается замкнутым выражением (без параметров) и будет иметь конкретное значение. В этом случае θ называется ***оценкой для выражения*** T, а само выражение $T\theta$ называется ***оцененным***. В общем оцененное выражение — это любое замкнутое выражение языка $\Omega(D)$.

Все сказанное переносится и на формулы языка Ω. Значением оцененной формулы является ложь или истина. При этом говорят, что оцененная формула истинна (ложна) в модели M. Подчеркнем, что сами по себе формулы не имеют какого-либо значения. Истинность возникает, когда конкретную интерпретацию получают все функциональные и предикатные символы, а формула не содержит параметров (или все они заменены константами в результате оценки).

Пример 3.1. В качестве примера логико-математического языка рассмотрим язык элементарной арифметики. Этот язык имеет лишь один сорт переменных, которые мы будем обозначать x, y, z, … Единственную константу обозначим 0. Три функциональных символа f, g, h, из которых первый одноместный, два остальных двуместные. Единственный двуместный предикатный символ обозначим p.

Для функциональных и предикатного символов введем альтернативные бинарные нотации:
$f(x) \equiv x^+$ (переход к следующему натуральному числу);
$g(x,y) \equiv (x + y)$ (операция сложения);
$h(x,y) \equiv (x \cdot y)$ (операция умножения);
$p(x,y) \equiv (x = y)$ (формальное равенство).

Для введенного языка рассмотрим две модели. Первую обозначим ω. Носителем этой модели является множество $\mathbb{N}_0$ целых неотрицательных чисел. Символу 0 поставим в соответствие число нуль. Функциональным символам припишем соответствующие операции над целыми числами. Наконец, равенство $(x = y)$ будет истинным тогда и только тогда, когда два числа x и y совпадают.

Рассмотрим терм $(x + y) \cdot z + x^+$. Этому терму мы можем давать различные оценки. Например, при оценке $\begin{pmatrix} x & y & z \\ \underline{1} & \underline{2} & \underline{3} \end{pmatrix}$ получим терм $(\underline{1} + \underline{5}) \cdot \underline{3} + \underline{1}^+$, который в ω имеет значение 20. Формула $\exists y\,(x + y = z)$ при оценке $\begin{pmatrix} x & z \\ \underline{3} & \underline{5} \end{pmatrix}$ дает истинную оцененную формулу $\exists y\,(\underline{3} + y = \underline{5})$. В то же время при оценке $\begin{pmatrix} x & z \\ \underline{5} & \underline{3} \end{pmatrix}$ получим ложную формулу $\exists y\,(\underline{5} + y = \underline{3})$.

Носителем второй модели, которую обозначим R, является множество $\mathbb{R}$ действительных чисел. Функциональным символам опять сопоставим сложение, умножение и добавление единицы. Формула $x = y$ выражает совпадение действительных чисел.

Вообще говоря, в качестве интерпретации рассматриваемого языка можно выбрать любое множество D, два любых отображения типа f, g: $D^2 \to D$, отображение типа h: $D \to D$ и отображение p: $D \to \mathbb{B}$. Все возможно. Но при этом меняется смысл термов и формул. Например, в модели ω истинна формула $x + y = y + x$, отражающая арифметический закон. Она сохраняется в модели R, но теряет силу в других моделях.

3.5. Логические законы

Формулы общезначимые, выполнимые, опровержимые, тождественно ложные. Пропозициональные тавтологии. Тавтологии с кванторами. Пример: $\neg\exists x \neg X \to \forall x\, X$ — тавтология; $\forall x\, \exists y\, p(x, y) \to \exists y\, \forall x\, p(x, y)$ — нет. Основные логические законы: а) де Моргана (пронесение через отрицание); б) пронесение кванторов через бинарные связки (8 законов); в) обобщенное пронесение кванторов через дизъюнкцию и конъюнкцию (4 закона); г) добавление кванторов или изменение порядка (5 законов); д) законы конгруэнтности.

Среди формул языка Ω есть такие, которые остаются истинными при любой интерпретации и при любой оценке. Такие формулы называют ***тавтологиями*** или ***общезначимыми формулами***. То, что X — тавтология, будем записывать так: $\models X$. Например, формула $p \to (q \to p)$ истинна в любой модели, поскольку вообще не имеет параметров и в любой модели автоматически становится оцененной, тождественно истинной формулой. В то же время есть формулы, которые являются тождественно истинными для данной модели, но не являются тождественно истинными в другой модели. Например, формула $x \cdot y = y \cdot x$ в модели ω тождественно истинна. Однако, если в качестве модели взять множество квадратных матриц (операции x^+ можно поставить в соответствие добавление единичной матрицы) с естественной интерпретацией двуместных функциональных символов, формула $x \cdot y = y \cdot x$ уже не будет тождественно истинной. Если формула X является истинной в данной модели M при любой оценке (т.е. тождественно истинна в M), но не является истинной в некоторой другой модели при некоторой оценке, то эта формула выражает ***закон в модели*** M, истинность X в модели M будем обозначать $M \models X$.

Поскольку нульарные предикатные символы не имеют параметров, их значение не меняется в рамках заданной модели. Поэтому множество всех нульарных предикатных символов в сочетании с логическими связками образует подмножество рассматриваемого языка, идентичное языку алгебры высказываний. Следовательно, любой тавтологии из алгебры высказываний соответствует тавтология языка Ω. Например, $\models \neg A \vee B \to \neg(A \wedge \neg B)$, так как это калька с тавтологии из алгебры высказываний. Такие тавтологии будем называть ***пропозициональными***. В них нет ни предметных переменных, ни кванторов.

Кроме пропозициональных тавтологий в языке Ω есть и другие тавтологии, порожденные кванторами. Например $\neg\exists x\neg X \to \forall x\, X$ — тавтология. Чтобы это показать, выберем произвольную модель M и в ней произвольную оценку θ. Поскольку переменная x не входит свободно в формулу $\neg\exists x\neg X \to \forall x\, X$, можно считать, что x не входит и в верхнюю строку θ. Поэтому $(\neg\exists x\neg X \to \forall x\, X)\theta = \neg\exists x\neg X\theta \to \forall x\, X\theta$. Таким образом, необходимо установить истинность $M \models \neg\exists x\neg(X\theta) \to \forall x\,(X\theta)$. Для доказательства этого предположим, что левая часть импликации истинна, т.е. $M \models \neg\exists x\neg(X\theta)$. Надо доказать, что тогда $M \models \forall x\,(X\theta)$, т.е. для всякого объекта $a \in D_\pi$ верно $M \models (X\theta)^x_{\underline{a}}$. Предполагая противное, заключаем, что существует такой объект $a \in D_\pi$, что формула $(X\theta)^x_{\underline{a}}$ в M ложна. Но тогда $M \models \neg(X\theta)^x_{\underline{a}}$. Следовательно $M \models \exists x\,\neg(X\theta)$. Но это противоречит предположению $M \models \neg\exists x\neg(X\theta)$.

Пример 3.2. Покажем, что формула $\forall x\,\exists y\, p(x,y) \to \exists y\,\forall x\, p(x,y)$ не является логическим законом. Для этого необходимо придумать такую модель M, что $M \models \forall x\,\exists y\, p(x,y)$, но формула $\exists y\,\forall x\, p(x,y)$ в модели M не является истинной (отметим, что рассматриваемые формулы замкнуты, а потому в любой модели автоматически являются оцененными).

В качестве модели выберем ω — модель элементарной арифметики. Предикатному символу $p(x,y)$ поставим в соответствие отношение $x < y$ на множестве натуральных чисел. Тогда формула $\forall x\,\exists y\, p(x,y)$ будет означать, что в множестве натуральных чисел с отношением порядка $x < y$ нет максимального элемента, а формула $\exists y\,\forall x\, p(x,y)$ будет иметь противоположный смысл, что в множестве натуральных чисел есть наибольший элемент. Ясно, что первая формула истинна, а вторая нет.

Две формулы X и Y называются ***логически эквивалентными***, $X \equiv Y$, если формула $X \sim Y$ есть логический закон (напомним, что запись $X \sim Y$ — это сокращение формулы $(X \to Y) \wedge (Y \to X)$). Можно сказать иначе: две формулы логически эквивалентны, если в любой модели и при любой оценке они обе или истинны, или ложны.

Теорема 3.2. Если $X \equiv Y$, то $\forall xX \equiv \forall xY$ и $\exists xX \equiv \exists xY$.

◀ Пусть $x,\ x_1,\ x_2,\ \ldots,\ x_n$ — полный набор всех параметров формул X и Y, в котором в качестве первой выбрана переменная x. При произвольно выбранной интепретации языка и при указанном порядке переменных формулы X и Y порождают булевы функции $f_X(x, x_1, x_2, \ldots, x_n)$ и $f_Y(x, x_1, x_2, \ldots, x_n)$. Эквивалентность формул означает, что эти функции совпадают. Но тогда

$$\min_x f_X(x, x_1, \ldots, x_n) = \min_x f_Y(x, x_1, \ldots, x_n) \quad \text{и} \quad \max_x f_X(x, x_1, \ldots, x_n) = \max_x f_Y(x, x_1, \ldots, x_n).$$

Вычисление минимума или максимума функций f_X, f_Y равносильно навешиванию квантора $\forall$ или $\exists$ с переменной x на формулы X, Y. Это указывает на то, что пары формул $\forall xX$ и $\forall xY$, $\exists xX$ и $\exists xY$ эквивалентны в выбранной модели. Так как модель выбиралась произвольно, заключаем, что эти пары формул логически эквивалентны. ▶

Приведем некоторые логические законы. Остановимся лишь на тех, которые выходят за рамки пропозициональных.

Законы де Моргана:

1. $\neg(\forall x\, X) \sim \exists x\,(\neg X)$.
2. $\neg(\exists x\, X) \sim \forall x\,(\neg X)$.

Доказательства общезначимости этих формул аналогичны доказательству общезначимости формулы $\neg\exists x\neg X \to \forall x\, X$, рассмотренной выше.

Одностороннее пронесение кванторов (здесь X не содержит свободно переменной x):

3. $X \wedge \forall x\, Y(x) \sim \forall x\,(X \wedge Y(x))$.
4. $X \vee \forall x\, Y(x) \sim \forall x\,(X \vee Y(x))$.
5. $X \wedge \exists x\, Y(x) \sim \exists x\,(X \wedge Y(x))$.
6. $X \vee \exists x\, Y(x) \sim \exists x\,(X \vee Y(x))$.
7. $X \to \forall x\, Y(x) \sim \forall x\,(X \to Y(x))$.
8. $X \to \exists x\, Y(x) \sim \exists x\,(X \to Y(x))$.
9. $\forall x\, Y(x) \to X \sim \exists x\,(Y(x) \to X)$.
10. $\exists x\, Y(x) \to X \sim \forall x\,(Y(x) \to X)$.

Рассмотрим, например, формулу 3. Выберем произвольную модель и в ней произвольную оценку θ. Поскольку x в формулу не входит свободно, считаем, что и в θ переменная x не выходит. Получим оцененную формулу $X\theta \wedge \forall x\, Y\theta \sim \forall x\,(X\theta \wedge Y\theta)$, которая является истинной в том и только в том случае, когда обе части эквиваленции имеют одно и то же истинностное значение. Если левая часть импликации истинна, то формулы $X\theta$ и $\forall x\, Y\theta$ обе истинны. Значит, при любом $\underline{a}$ формула $(Y\theta)^x_{\underline{a}}$ является истинной. Следовательно, истинна формула $X\theta \wedge (Y\theta)^x_{\underline{a}}$. Отсюда вытекает, что формула $\forall x(X\theta \wedge (Y\theta))$ в правой части эквиваленции истинна. Нетрудно увидеть, что приведенные рассуждения можно провести в обратном порядке и тем самым показать, что из истинности правой части эвиваленции вытекает истинность левой части.

Отметим, что доказанная формула 3 имеет естественную интрпретацию при выборе конкретной модели:

$$\min\left\{f_X,\ \min_x f_Y\right\} = \min_x \min\left\{f_X,\ f_Y\right\}.$$

Это верно при условии, что функция f_X не зависит от переменной x.

Дополнительные законы пронесения кванторов:

11. $\forall x\, X(x) \wedge \forall x\, Y(x) \sim \forall x\,(X(x) \wedge Y(x))$.
12. $\exists x\, X(x) \vee \exists x\, Y(x) \sim \exists x\,(X(x) \vee Y(x))$.
13. $\exists x\,(X(x) \wedge Y(x)) \to \exists x\, X(x) \wedge \exists x\, Y(x)$.
14. $\forall x\, X(x) \vee \forall x\, Y(x) \to \forall x\,(X(x) \vee Y(x))$.

Рассмотрим, например, формулу 11. Выбрав произвольную модель и в ней произвольную оценку θ (считаем, что она не содержит x), получим оцененную формулу $\forall x\, X\theta \wedge \forall x\, Y\theta \sim$ $\sim \forall x\,(X\theta \wedge Y\theta)$. Истинность левой части означает, что истинны формулы $\forall x\, X\theta$ и $\forall x\, Y\theta$. Следовательно, для любой константы $\underline{a}$ истинны $(X\theta)^x_{\underline{a}}$ и $(Y\theta)^x_{\underline{a}}$, откуда заключаем, что истинна формула $(X\theta)^x_{\underline{a}} \wedge (Y\theta)^x_{\underline{a}}$. Поэтому истинна $\forall x\,(X\theta \wedge Y\theta)$. Наоборот, если формула $\forall x\,(X\theta \wedge Y\theta)$ истинна, то для любой константы $\underline{a}$ истинна формула $(X\theta)^x_{\underline{a}} \wedge (Y\theta)^x_{\underline{a}}$. Делаем вывод, что для любой $\underline{a}$ истинна формула $(X\theta)^x_{\underline{a}}$ и для любой $\underline{a}$ истинна формула $(Y\theta)^x_{\underline{a}}$. Поэтому истинны формулы $\forall x\, X\theta$ и $\forall x\, Y\theta$, а значит, и формула $\forall x\, X\theta \wedge \forall x\, Y\theta$.

Логические законы, связанные добавлением кванторов или изменением их порядка:

15. $\forall x\, X \to X$.
16. $X \to \exists x\, X$.
17. $\exists x \exists y X \sim \exists y \exists x X$.
18. $\forall x \forall y X \sim \forall y \forall x X$.
19. $\exists x \forall y X \to \forall y \exists x X$.

Формулы 15 и 16 можно интерпретировать как утверждение, что для функции $u(x)$ имеет место неравенство

$$\min_x u(x) \leqslant u(x) \leqslant \max_x u(x). \tag{3.1}$$

Формулы 17 и 18 ассоциируются с правилом перестановки максимумов и минимумов по разным переменным, например:

$$\max_x \max_y u(x,y) = \max_y \max_x u(x,y),$$

а формула 19 — отражение неравенства

$$\max_x \min_y u(x,y) \leqslant \min_y \max_x u(x,y).$$

И в самом деле, $u(x,y) \leqslant \max_x u(x,y)$. Следовательно, $\min_y u(x,y) \leqslant \min_y \max_x u(x,y)$. В результате справа — константа, а слева — функция переменного x, т.е. $\min_y \max_x u(x,y)$ есть верхняя грань функции $\min_y u(x,y)$. Значит, имеет место неравенство (3.1).

Логические законы, связанные с конгруэнтностью (формула X не имеет свободных вхождений переменной y):

20. $\forall x\, X \sim \forall y\, X^x_y$.
21. $\exists x\, X \sim \exists y\, X^x_y$.

Эти логические законы указывают на то, что конгруэнтные формулы логически эквивалентны. Впрочем, эквивалентность конгруэнтных формул лучше доказывать непосредственно, опираясь на индуктивное определение конгруэнтности. Иначе придется доказывать, что конгруэнтная формула получается последовательным переименованием, что интуитивно ясно, но требует формальных рассуждений.

Что касается сформулированных логических законов, то можно рассуждать так (на пример квантора всеобщности). Выбираем произвольную модель и в ней произвольную оценку θ. При этом, поскольку ни x, ни y в формулу $\forall x X$ не входят свободно, можно считать, что и в оценку θ эти переменные не входят. Если переменная x не входит в X свободно, то наличие квантора не играет роли: формула $X\theta$ оказывается оцененной. Если же x входит в X свободно, то формула $X\theta$ будет содержать единственный параметр x. Истинность $\forall x(X\theta)$ означает, что для любого объекта $\underline{a}$ формула $(X\theta)^x_{\underline{a}}$ оказывается истинной. Но если мы предварительно выполним подстановку $(X\theta)^x_y)$, а затем подстановку $\binom{y}{\underline{a}}$, то получим ту же формулу $(X\theta)^x_{\underline{a}})$. Она окажется истинной. Аналогичны рассуждения при ложности формулы $\forall x(X\theta)$. Тем самым доказано, что в выбранной модели формулы $\forall x X$ и $\forall y X^x_y$ при любой оценке одновременно истинны или ложны. Значит, они эквивалентны, а формула $\forall x X \sim \forall y X^x_y$ есть логический закон.

3.6. Замены

Понятие формального предиката. Замена элементарной формулы формальным предикатом (индуктивное определение). Теорема о замене элементарных подформул. Правило замены эквивалентным. Пример: эквивалентность $r(x)\vee\forall z\,(\neg\forall x\,q(x,z))$ и $r(x)\vee\forall z(\exists x\,\neg q(x,z))$. Принцип двойственности. Нильпотентность. Эквивалентность двойственных формул.

Речь идет о получении аналога теоремы 1.2. У нас нет самостоятельных пропозициональных переменных. Их роль играют предикатные символы арности нуль. Разумеется, теорема 1.2 остается верной при замене предикатного символа арности нуль какой-либо формулой. Но как быть, если нужно заменить предикатный символ, имеющий параметры? Таким образом, речь идет об обобщении теоремы 1.2 на случай замены в формуле произвольной элементарной подформулы. Механизм замены должен сохранять эквивалентность формул.

При замене какого-либо вхождения элементарной подформулы $p(t_1,\ldots,t_n)$ (аргументы в этой подформуле — какие-либо термы) некоторой формулой X следует установить соответствие между переменными, входящими в элементарную подформулу, и переменными, входящими в X. Такое соответствие можно организовать, вводя понятие ***формального предиката***. Под формальным предикатом мы будем понимать слово вида $x_1x_2\ldots x_nX$. Такое слово означает лишь, что задана какая-то формула X и упорядоченный набор переменных x_1, x_2, …, x_n — больше ничего. Эти переменные и все их вхождения в X в рамках формального предиката будем считать связанными. Кортеж сортов $(\pi_1,\ \pi_2,\ \ldots,\ \pi_n)$ „объявленных“ переменных x_1, x_2, …, x_n будем называть типом формального предиката.

Чтобы в формуле Y заменить элементарную подформулу $p(t_1,\ldots,t_n)$, необходимо построить формальный предикат $U = x_1x_2\ldots x_nX$ того же типа, что и предикатный символ p, затем в формуле X выполнить подстановку $X^{x_1,\ldots,x_n}_{t_1,\ldots,t_n}$ и результат подстановки подставить в формулу Y взамен подформулы $p(t_1,\ldots,p_n)$.

Подобный подход будет более понятным, если учесть, что в любой модели формула задает предикат, роль аргументов которого играют переменные формулы. Задавая формальный предикат, мы выбираем некоторую совокупность переменных (записываем их перед формулой) и упорядочиваем. Эти отобранные переменные фиксируются, остальные остаются свободными. При подстановке вместо элементарной формулы мы должны на место „объявленных аргументов“ предиката подставить аргументы элементарной формулы и в таком виде подставить формулу вместо элементарной. Например, пусть элементарная формула имеет вид $p(t_1,t_2,\ldots,t_n)$. Формальный предикат $x_1x_2\ldots x_nX$ следует в любой модели интерпретировать как предикат $\widetilde{p}(x_1,\ldots,x_n,z_1,\ldots,z_m)$, в котором первые n аргументов по типам соответствуют аргументам предиката p. В результате мы меняем в исходной формуле элементарную подформулу $p(t_1,t_2,\ldots,t_n)$ на подформулу $\widetilde{p}(t_1,\ldots,t_n,z_1,\ldots,z_m)$. Отобранные элементы формального предиката играют роль подстановочных мест и заменяются аргументами элементарной подформулы. Оставшиеся переменные формулы X без изменений переходят в изменяемую формулу.

Указанный алгоритм подстановки можно описать с помощью индукции по построению формулы. Пусть задан формальный предикат $U = x_1x_2\dots x_nX$ и Y обозначает исходную формулу. Результат $Y(p \,||\, U)$ подстановки в Y формального предиката U типа $(\pi_1,\ \pi_2,\ \dots,\ \pi_n)$ вместо элементарной подформулы $p(t_1,\dots,t_n)$ с предикатным символом p того же типа определим следующим образом.

1. Если Y — атомарная формула $q(r_1,\dots,r_m)$, то при q, отличном от p полагаем $Y(p\,||\,U) = Y$, а если $Y = p(t_1,\dots,t_n)$, то полагаем $Y(p \,||\, U) = X\theta$, где $\theta = \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ t_1 & t_2 & \dots & t_n \end{pmatrix}$.

2. Если $Y = V\nabla W$, где ∇ — одна из трех двуместных логических связок, то $Y(p \,||\, U) = V(p \,||\, U)\nabla W(p \,||\, U)$. Если $Y = \neg V$, то $Y(p \,||\, U) = \neg V(p \,||\, U)$.

3. Если $Y = \nabla z\, V$, где ∇ — один из кванторов, то возможны два случая. В первом случае, когда U не содержит свободных вхождений z или V не содержит вхождений предикатного символа p, полагаем $Y(p\,||\,U) = \nabla z\, V(p\,||\,U)$. Во втором случае, когда z имеет свободные вхождения в U, а V имеет вхождения p, выбираем новую переменную u (т.е. не встречающуюся ни в Y, ни в U) и полагаем $Y(p \,||\, U) = \nabla u\, (V_u^z)(p \,||\, U)$.

Обратим внимание на то, что в соответствии с описанным индуктивным правилом при замене меняются все вхождения данного предикатного символа. Последний пункт регулирует вхождение свободных переменных формального предиката. Снова действует правило: никакое свободное вхождение переменной после преобразования не должно оказаться связанным. Если такое получается, соответствующую кванторную переменную надо изменить.

Теорема 3.3. Если $X \equiv Y$, то и $X(p \,||\, x_1\dots x_kZ) \equiv Y(p \,||\, x_1\dots x_kZ)$. Если $Y \equiv Z$, то и $X(p \,||\, x_1\dots x_kY) \equiv X(p \,||\, x_1\dots x_kZ)$.

◄ Первое утверждение сводится к следующему: если U — тавтология, то и $U(p \,||\, x_1\dots x_kZ)$ — тавтология (достаточно в качестве U взять формулу $X \sim Y$). То, что U — тавтология, означает, что при любой интерпретации и любой оценке формула U истинна. Интерпретация, в частности, обеспечивает конкретное содержание предикатного символа p, а формальный предикат $x_1x_2\dots x_kZ$ превращается в некоторый предикат $q(x_1,\dots,x_k,y_1,\dots,y_l)$. При замене происходит замена одного предиката другим, при этом появляются свободные переменные $y_1,\ \dots,\ y_l$, которые мы можем зафиксировать с помощью оценки. В результате дело фактически сводится к замене предикатной буквы p на предикатную букву q. Такую замену можно интепретировать как изменение интерпретации буквы p. При изменении интерпретации формула остается истинной. Значит, и после замены формула остается истинной.

Доказательство второго утверждения аналогично. Замена предикатного символа p одним из формальных предикатов можно интерпретировать как выбор интепретации символа p. Так как формулы Y и Z эквивалентны, то и первая, и вторая замены дают одну и ту же интерпретацию символу p. Следовательно и истинностное значение двух формул будет одинаковым. ►

Следствие 3.1. Если $Y \equiv Z$, то после замены в формуле X одного из вхождений подформулы Y формулой Z получим формулу X', эквивалентную X.

◄ Пусть $x_1,\ x_2,\ \dots,\ x_n$ — полный список переменных, входящих в формулы Y и Z. Рассмотрим формулу Γ, полученную заменой подформулы X элементарной формулой $p(x_1,x_2,\dots,x_n)$. Тогда $X = \Gamma(p \,||\, x_1\dots x_nY)$ и $X' = \Gamma(p \,||\, x_1\dots x_nZ)$. Согласно теореме эти формулы эквивалентны. ►

Утверждение следствия известно как ***правило замены эквивалентным***.

Пример 3.3. Покажем, что формулы $r(x) \vee \forall z\,(\neg\forall x\, q(x,z))$ и $r(x) \vee \forall z(\exists x\, \neg q(x,z))$ эквивалентны.

Обе формулы можно рассматривать как две подстановки в формулу $X = r(x) \vee \forall z\, p(z)$ двух эквивалентных формул $V = \neg\forall x\, q(x,z)$ и $W = \exists x\, \neg q(x,z)$. Сформируем два формальных предиката zV и zW. В результате получим $X(p \,||\, zV) = r(x) \vee \forall z\,(\neg\forall x\, q(x,z))$ и $X(p \,||\, zW) = r(x) \vee \forall z(\exists x\, \neg q(x,z))$. В соответствии с последним логическим законом эти формулы эквивалентны.

Отметим, что было бы неверно в качестве шаблона использовать, например, формулу $X_1 = r(x) \vee \forall z\, p$, опуская в заменяемом символе связанную переменную z. В этом случае в соответствии с индуктивным правилом мы получили бы формулы $X_1(p \,||\, zV) = r(x) \vee \forall z_1\, (\neg \forall x\, q(x, z))$ и $X_1(p \,||\, zW) = r(x) \vee \forall z_1 (\exists x\, \neg q(x, z))$, которые тоже эквивалентны, но отличаются от того, что мы хотели бы получить.

На формулы алгебры предикатов, не содержащие импликации, распространяется принцип двойственности. Формулу X^*, двойственную данной формуле X определим индуктивно:

1) если X — элементарная формула, то $X^* = X$;
2) если $X = \neg U$, то $X^* = \neg U^*$;
3) если $X = U \wedge V$, то $X^* = U^* \vee V^*$;
4) если $X = U \vee V$, то $X^* = U^* \wedge V^*$;
5) если $X = \forall x U$, то $X^* = \exists x U^*$;
6) если $X = \exists x U$, то $X^* = \forall x U^*$.

Непосредственно из определения следует, что преобразование двойственности нильпотентно: $X^{**} = X$. Следующее утверждение связывает преобразование двойственности с отношением эквивалентности. Кроме того, верно следующее утверждение. Назовем внутренним отрицанием $\check{X}$ формулы X ее преобразование, при котором каждая элементарная подформула $p(t_1, t_2, \ldots, t_n)$ заменяется формальным предикатом $x_1 x_2 \ldots x_n \neg p(x_1, x_2, \ldots, x_n)$.

Теорема 3.4. Формула X^*, двойственная формуле X, эквивалентна формуле $\neg\check{X}$.

◀ Доказательство проводится индукцией по построению формулы. Для элементарных формул внутреннее отрицание совпадает с обычным отрицанием $\neg$, так что $\neg\check{X} = X$, но в то же время по определению $X^* = X$.

Если $X = U \wedge V$ и уже доказано, что $U^* \equiv \neg\check{U}$, $V^* \equiv \neg\check{V}$, то согласно определению двойственной формулы и правилу замены эквивалентным

$$X^* = U^* \vee V^* \equiv \neg\check{U} \vee \neg\check{V} \equiv \neg(\check{U} \wedge \check{V}) = \neg\check{X}.$$

Аналогично утверждение доказывается для дизъюнкции и отрицания.

Если $X = \forall x U$ и уже доказано, что $U^* \equiv \neg\check{U}$, то

$$X^* = \exists x U^* \equiv \exists x \neg\check{U} = \neg\forall x \check{U} = \neg\check{X}.$$

Аналогичны рассуждения для кванторной приставки $\exists x$. ▶

Теорема 3.5. Эквивалентность $X \equiv Y$ равносильна эквивалентности $X^* \equiv Y^*$.

◀ Доказательство в силу нильпотентности достаточно провести в одну сторону. При этом достаточно заметить, что если $X \equiv Y$, то $\check{X} \equiv \check{Y}$ и $\neg X \equiv \neg Y$. Поэтому в силу предыдущей теоремы

$$X^* \equiv \neg\check{X} \equiv \neg\check{Y} \equiv Y^*. \qquad ▶$$

3.7. Упрощение формул

Приведение бескванторной формулы к ДНФ и КНФ. Предваренная нормальная форма. Теорема о приведении. Пример: $X = \forall x\, \neg\exists y\, p(x, y) \to \forall x\, (q(x) \to \neg\exists y\, p(x, y))$. Замечание о порядке вынесения кванторов.

На основании логических законов можно преобразовывать формулы логико-математического языка, упрощая или добиваясь получения формул определенной структуры. Например, если в формуле отсутствуют кванторы (бескванторная формула), то ее можно привести к ДНФ или КНФ. Такое преобразование всегда можно проводить в рамках языка логики высказываний, рассматривая любую элементарную подформулу как пропозициональную переменную.

Теорема 3.6. Всякая бескванторная формула эквивалентна некоторой ДНФ и некоторой КНФ.

Разумеется, подобные преобразования возможны и для формул, содержащих кванторы. Наличие кванторов лишь усложняет сам процесс, но не отменяет его.

Кванторы не мешают преобразованиям, если они в формуле собраны вместе. Формулу вида $\nabla x_1\, \nabla x_2\, \ldots\, \nabla x_n\, X$, где X — бескванторная формула, называют ***предваренной*** или ***пренексной***. В частности, любая бескванторная формула является предваренной. Предваренной (пренексной) формой данной формулы X называют любую предваренную формулу Y, логически эквивалентную X. Часть формулы Y без кванторных приставок называют ***матрицей предваренной формы***.

Теорема 3.7 (о предваренной форме). Любая форма имеет предваренную форму.

◀ Для доказательства нужно исходную формулу X заменить формулой X' со свойством чистоты переменных, а затем, применяя логические законы одностороннего пронесения кванторов, „вытащить“ все кванторы за пределы формулы.

Аккуратно доказательство следует проводить методом индукции по построению формулы. В самом деле, утверждение очевидно для элементарных формул (они все бескванторные). Пусть $X = V \circ W$, где $\circ$ — одна из двуместных логических связок, и формулы V и W имеют предваренные формы: $V \equiv \nabla v_1 \ldots \nabla v_m\, V'$, $W \equiv \nabla w_1 \ldots \nabla w_k\, W'$, где V' и W' бескванторные. Полагая, что X обладает свойством чистоты переменных, заключаем, что все переменные v_i и w_j различны, V' не содержит переменных w_j, а W' — переменных v_i. Тогда с помощью законов одностороннего пронесения кванторов получаем

$$X \equiv (\nabla v_1 \ldots \nabla v_m\, V') \circ (\nabla w_1 \ldots \nabla w_k\, W') \equiv \nabla v_1 \ldots \nabla v_m\, \nabla w_1 \ldots \nabla w_k\, (V' \circ W').$$

Здесь мы воспользовались также правилом замены подформулы эквивалентной.

Если формула X имеет вид $X = \neg V$, рассуждения аналогичны. Наконец, в случае $X = \nabla z\, V$ достаточно заменить V предваренной формой, чтобы получить предваренную форму X. ▶

Пример 3.4. Рассмотрим формулу $X = \forall x\, \neg\exists y\, p(x,y) \to \forall x\, (q(x) \to \neg\exists y\, p(x,y))$. Она не обладает свойством чистоты. Используя переименование с помощью двух новых переменных u и v, получим конгруэнтную (а потому эквивалентную) формулу

$$X' = \forall x\, \neg\exists y\, p(x,y) \to \forall u\, (q(u) \to \neg\exists v\, p(u,v)).$$

Применяя одностороннее пронесение кванторов, последовательно получаем:

$$\begin{aligned}
&\forall x\, \neg\exists y\, p(x,y) \to \forall u\, (q(u) \to \neg\exists v\, p(u,v)) \equiv \\
&\equiv \forall x \forall y\, \neg p(x,y) \to \forall u\, (q(u) \to \forall v\, \neg p(u,v)) \equiv \\
&\equiv \forall x \forall y\, \neg p(x,y) \to \forall u \forall v\, (q(u) \to \neg p(u,v)) \equiv \\
&\equiv \exists x \exists y\, \big(\neg p(x,y) \to \forall u \forall v\, (q(u) \to \neg p(u,v))\big) \equiv \\
&\equiv \exists x \exists y \forall u \forall v\, \big(\neg p(x,y) \to (q(u) \to \neg p(u,v))\big).
\end{aligned}$$

Замечание. Вынесение кванторов может проходить в разном порядке. Следовательно, и в предваренной форме кванторные приставки могут идти в разном порядке: предваренная форма определена неоднозначно. В матрице предваренной формы все связки идут в том же порядке и без изменений. Она получается из исходной формулы удалением всех кванторных приставок.

4. ИСЧИСЛЕНИЕ ПРЕДИКАТОВ

4.1. Построение теории P

Язык алгебры предикатов как расширение языка алгебры высказываний. Аксиомы исчисления предикатов: схемы аксиом исчисления высказываний плюс 4 аксиомы с кванторами: 11) $\forall x\, X \to X_t^x$; 12) $\forall x\,(Y \to X(x)) \to (Y \to \forall x\, X(x))$; 13) $X_t^x \to \exists x\, X$; 14) $\forall x\,(X(x) \to Y) \to (\exists x\, X(x) \to Y)$. Правила вывода: 1^* $\frac{X,\ X \to Y}{Y}$; 2^* $\frac{X}{\forall x\, X}$. Вывод и вывод из гипотез. Структурное ограничение. Секвенции.

Исчисление предикатов строится как расширение исчисления высказываний. Основная цель та же: формализовать доказательство различного рода утверждений. Исчисление предикатов — составная часть любой формальной теории, отвечающая за логическую часть теории.

Мы будем строить теорию $\mathcal{P}$, языком Ω которой является произвольный логико-математический язык. Фактически мы строим целое семейство формальных теорий. Однако все эти теории имеют единую логическую составляющую, отличаясь лишь способом формирования предметных выражений — термов. Любую из построенных формальных теорий можно трансформировать так, что предметная составляющая будет простейшей: один сорт переменных при отсутствии констант и функциональных символов. Упрощается и интерпретация простейшего языка. Достаточно задать единую предметную область для всех предметных переменных и указать отображение, которое n-арному предикатному символу ставит в соответствие булеву функцию от n переменных, а в случае 0-арного предикатного символа — символ 0 или 1.

Зададим аксиомы исчисления предикатов в виде некоторого набора схем аксиом. Первые одиннадцать схем повторяют схемы исчисления высказываний:

1) $X \to (Y \to X)$;
2) $X \to Y \to (X \to (Y \to Z) \to (X \to Z))$;
3) $X \wedge Y \to X$;
4) $X \wedge Y \to Y$;
5) $(X \to Y) \to ((X \to Z) \to (X \to Y \wedge Z))$;
6) $X \to X \vee Y$;
7) $Y \to X \vee Y$;
8) $X \to Z \to (Y \to Z \to (X \vee Y \to Z))$;
9) $X \to Y \to (\neg Y \to \neg X)$;
10) $\neg\neg X \to X$;
11) $X \to \neg\neg X$.

Еще четыре схемы связаны с кванторами (ниже X_t^x — правильная подстановка терма t вместо переменной x, Y не содержит свободных вхождений переменной x):

12) $\forall x\, X \to X_t^x$;
13) $\forall x\,(Y \to X(x)) \to (Y \to \forall x\, X(x))$;
14) $X_t^x \to \exists x\, X$;
15) $\forall x\,(X(x) \to Y) \to (\exists x\, X(x) \to Y)$.

В дополнение к правилу заключения (modus ponens) добавляется правило обобщения:

1^* $\dfrac{X,\ X \to Y}{Y}$;

2^* $\dfrac{X}{\forall x\, X}$.

Как и в исчислении высказываний, выводом называем последовательность формул языка, в которой каждая формула либо аксиома, либо получена по одному из правил вывода из предшествующих формул. Также вводим понятие вывода из гипотез Γ (частного вывода), в котором

в последовательности формул могут находиться формулы из списка Γ. Однако для вывода из гипотез введем дополнительное структурное требование: если формула $\forall x\, X$ получена по правилу обобщения, то во всех гипотезах, используемых для вывода этой формулы, переменная x не должна иметь свободных вхождений. Указанное структурное требование не касается аксиом. Поэтому в выводе без гипотез его можно не учитывать.

Смысл сформулированного структурного требования становится понятным, если учесть, что вывод должен сохранять истинность формул. Правило заключения сохраняет истинность формул при фиксированной модели и фиксированной оценке, а именно: если θ — оценка формулы $X \to Y$ в данной модели M, то из истинности $X\theta$ и $(X \to Y)\theta$ следует истинность $Y\theta$. Правило обобщения в отсутствие структурного требования нарушает фиксированность оценки при выводе.

Впрочем, структурное требование касается лишь частного вывода. Мы могли бы это ограничение не учитывать, но тогда нам придется считаться с ним при „сшивке“ частных выводов в окончательный вывод без гипотез. Поясним сказанное. Выводимость из X формулы $\forall x\, X$ следует увязать с выводимостью формулы $X \to \forall x\, X$, т.е. с тем, что эта формула есть тавтология. Но если переменная x входит в X свободно, то при некоторой интерпретации для некоторого объекта a формула $X^x_{\underline{a}}$ будет ложной, а для некоторого объекта b формула $X^x_{\underline{b}}$ — истинной. В этом случае $\forall x\, X$ в соответствии с интерпретацией квантора всеобщности будет ложной формулой при любой оценке, а это приведет к ложности $X \to \forall x\, X$. Однако эта формула будет тавтологией, если X не имеет свободных вхождений x.

Мы сохраняем запись $\Gamma \vdash X$, означающую, что формула X выводима из списка гипотез Γ. Эту запись называем ***секвенцией***.

Пример 4.1. Пусть переменная y не имеет свободных вхождений в X. Тогда $\vdash \forall x\, X \to \to \forall y\, (X^x_y)$. Действительно:

1) $\forall x\, X \to X^x_y$ (схема аксиом 12);
2) $\forall y\, (\forall x\, X \to X^x_y)$ (по правилу обобщения из 1);
3) $\forall y\, (\forall x\, X \to X^x_y) \to (\forall x\, X \to \forall y\, X^x_y)$ (схема аксиом 13);
4) $\forall x\, X \to \forall y\, X^x_y$ (по правилу заключения из 2 и 3);

4.2. Правила естественного вывода

Теорема о дедукции. Правила естественного вывода исчисления высказываний. Новые правила:

р1) $\frac{\Gamma \vdash X}{\Gamma \vdash \forall y\, X^x_y}$ (введение общности);

р2) $\frac{\Gamma \vdash \forall x\, X}{\Gamma \vdash X^x_t}$ (удаление общности);

р3) $\frac{\Gamma \vdash X^x_t}{\Gamma \vdash \exists x\, X}$ (введение существования);

р4) $\frac{\Gamma, X \vdash Y}{\Gamma, \exists y\, X^x_y \vdash Y}$ (удаление существования).

Пример: $\exists x\, X \to \neg\forall x(\neg X)$.

Как и в случае исчисления высказываний, на практике вывод строится с помощью правил естественного вывода. Сохраняются структурные правила естественного вывода, так как они не связаны с сутью самого вывода. Сохраняется теорема о дедукции. Однако в доказательство этой теоремы необходимо внести некоторые коррективы.

Теорема 4.1. Если $\Gamma, X \vdash Y$, то $\Gamma \vdash X \to Y$.

◀ Доказательство теоремы, как и в исчислении высказываний, проводится индукцией по построению формулы. Рассуждения в основном повторяют доказательство теоремы из исчисления высказываний. Дополнительно надо лишь рассмотреть случай, когда конечная формула вывода получена по правилу обобщения.

Итак, пусть в выводе $X_1, X_2, \ldots, X_i = X, \ldots, X_j, \ldots, X_n = Y$ конечная формула Y имеет вид $\forall x X_j$. Этот вывод, как свою часть, содержит и вывод формулы X_j. Если в выводе формулы X_j формула X не используется, то можно считать, что ее нет и в выводе $X_1, X_2, \ldots, X_n = Y$. Добавляя к этому выводу аксиому $\forall x\, X_j \to (X \to \forall x\, X_j)$ (схема аксиом 1), а затем на

основании правила заключения $X \to \forall x\, X_j$, получим вывод формулы $X \to Y \equiv X \to \forall x\, X_j$ из гипотез Γ. Пусть формула X используется в выводе формулы X_j. Тогда, согласно структурному требованию, формула X не имеет свободных вхождений переменной x. Согласно индуктивному предположению из вывода $\Gamma, X \vdash X_j$ вытекает вывод $\Gamma \vdash X \to X_j$, откуда по правилу обобщения (ни X, ни гипотезы, использованные в выводе X_j) не содержат свободных вхождений x) получаем $\Gamma \vdash \forall x\,(X \to X_j)$. Присоединив аксиому $\forall x\,(X \to X_j) \to (X \to \forall x\, X_j)$, получаем вывод формулы $X \to \forall x\, X_j$, т.е. формулы $X \to Y$. ▶

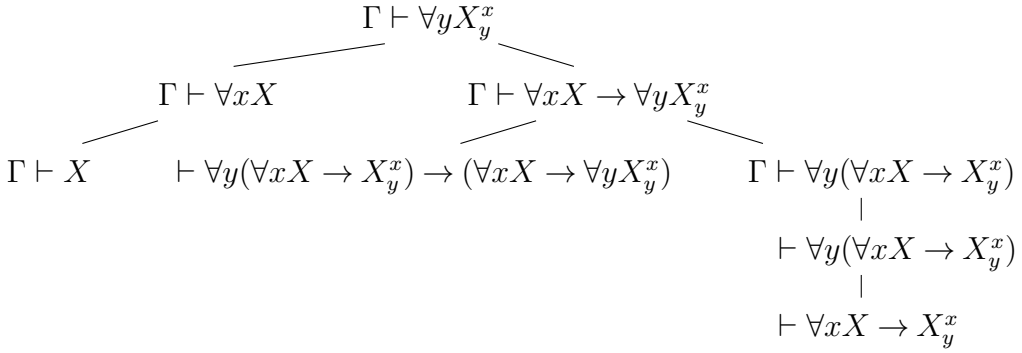
Теорема о дедукции позволяет без изменений перенести из исчисления высказываний в исчисление предикатов все восемь логических правил естественного вывода, а вслед за ними и дополнительные правила естественного вывода: доказательства этих правил не используют правила обобщения. Кроме того, добавляются следующие четыре правила с кванторами:

р1) $\dfrac{\Gamma \vdash X}{\Gamma \vdash \forall y\, X_y^x}$ (введение общности); р3) $\dfrac{\Gamma \vdash X_t^x}{\Gamma \vdash \exists x\, X}$ (введение существования);

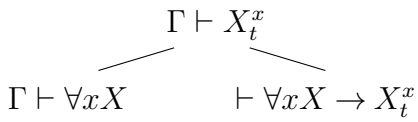
р2) $\dfrac{\Gamma \vdash \forall x\, X}{\Gamma \vdash X_t^x}$ (удаление общности); р4) $\dfrac{\Gamma, X \vdash Y}{\Gamma, \exists y\, X_y^x \vdash Y}$ (удаление существования).

В правилах р1 и р4 переменная x не имеет свободных вхождений в формулы списка Γ и формулу Y. Кроме того, если y отлично от x, то y не входит свободно в X. В правилах р2 и р3 подстановка $\binom{x}{t}$ должна быть свободной для формулы X.

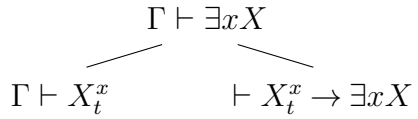
Докажем четыре новых правила естественного вывода. При выводе формулы р1 учтем, что формулы списка Γ не содержат свободных вхождений переменной x:



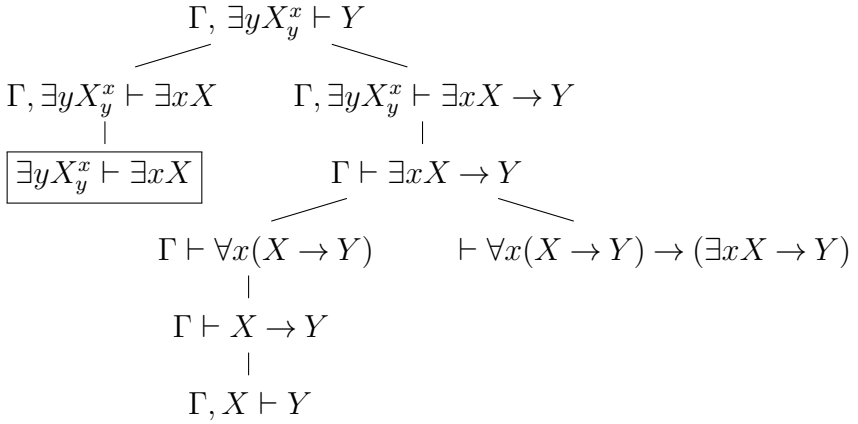
Формула р2 непосредственно следует из аксиомы 12:



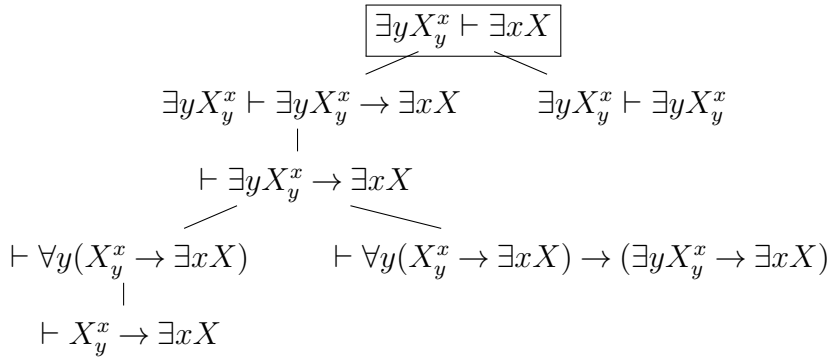
Аналогично выводится и формула р3:



Наиболее длинный вывод получается для правила р4:



В этом выводе осталась нераскрытой одна ветка (в рамке). Соответствующий вывод строится на той же аксиоме:



Использование правил естественного вывода такое же, как и в случае исчисления высказываний.

Пример 4.2. Используя правила естественного вывода, докажем выводимость в теории $\mathcal{P}$ формулы $\exists x\, X \to \neg\forall x \neg X$, т.е. построим вывод секвенции $\vdash \exists x\, X \to \neg\forall x \neg X$. Имеем цепочку преобразований „снизу вверх“:

$$\vdash \exists x\, X \to \neg\forall x \neg X \;\Leftarrow\; \exists x\, X \vdash \neg\forall x \neg X \;\Leftarrow\; X \vdash \neg\forall x \neg X \;\Leftarrow\; \forall x \neg X \vdash \neg X \;\Leftarrow\; \forall x \neg X \vdash \forall x \neg X.$$

Последняя секвенция вытекает из правила p2.

4.3. Глобальные свойства теории P

О множестве формул, выводимых в исчислении предикатов. Лемма об истинности при фиксированной оценке. Следствия: о том, что любая выводимая формула общезначима; о непротиворечивости исчисления предикатов. Понятие формальной аксиоматической теории. Непротиворечивость, полнота формальной аксиоматической теории.

Остановимся на вопросах непротиворечивости, полноты и разрешимости теории $\boldsymbol{P}$. Ответ на эти вопросы тесно связан с ответом на вопрос: каково множество формул заданного языка, выводимых в исчислении предикатов? Разумеется, это множество зависит от выбора языка. Однако эта зависимость не связана с логической частью языка.

В принципе можно ограничиться вариантом исчисления предикатов, который базируется на односортном языке без функциональных символов. В этом варианте каждая элементарная формула является простой: каждый аргумент есть предметная переменная. Действительно, зафиксировав некоторый порядок в счетном множестве переменных, мы можем каждую элементарную формулу интерпретировать как простую, вводя взамен каждого терма новую переменную. В результате язык модифицируется так, что в нем не будет функциональных символов. Наличие сортов переменных ограничивает их использование при формировании элементарных формул. Снятие ограничений приводит к тому, что, например, вместе с $p(x, y)$ появляется формула $p(y, x)$, т.е. происходит расширение множества элементарных формул. Допуская такое расширение, приходим к односортному языку, в котором термы — это переменные одного сорта.

Далее мы покажем, что множество выводимых формул совпадает с множеством логических законов в данном языке. Начнем с важного свойства, присущего выводу из гипотез. Оказывается, что вывод из гипотез сохраняет истинность формул при фиксированной модели и фиксированной оценке.

Лемма 4.1. Пусть Γ — список формул X_1, X_2, …, X_m и $\Gamma \vdash Y$. Если для данной модели M и данной оценки θ формул X_1, X_2, …, X_m, Y в M истинны $X_1\theta$, $X_2\theta$, …, $X_m\theta$, то и $Y\theta$ в M истинна.

◀ Доказательство проводится индукцией по длине вывода. Утверждение очевидно, если вывод содержит одну формулу, которая в этом случае есть либо аксиома (являющаяся логическим законом), либо гипотеза X_j. Но тогда $Y\theta$ и $X_j\theta$ — одно и то же.

Предположим утверждение доказано для всех выводов длины менее k. Рассмотрим произвольный вывод $Z_1, Z_2, \ldots, Z_k$ из гипотез Γ длины k с конечной формулой $Y = Z_k$. Формула Y есть либо логический закон, либо гипотеза, либо получена по правилу заключения, либо по правилу всеобщности. В первых двух случаях рассуждения те же, что и при $k = 1$. В третьем случае в выводе есть формулы Z_j и $Z_l = Z_j \to Y$. Если в какой-либо модели M при оценке θ истинны формулы $X_1\theta$, $X_m\theta$, то в силу предположения индукции формулы $Z_j\theta$ и $(Z_j \to Y)\theta$ истинны в M. В этом случае правило заключения гарантирует истинность в M формулы $Y\theta$, т.е. утверждение теоремы доказано в случае, когда формула Y получена по правилу заключения.

Пусть Y получена по правилу обобщения, т.е. $Y = \forall x Z_j$ для некоторой формулы Z_j, $j < k$. Предположим, что в выводе формулы Z_j использованы все гипотезы $X_1, \ldots, X_m$ (неиспользуемые можно просто не рассматривать). В силу структурного требования каждая из этих формул не имеет свободных вхождений переменной x. Обозначив через $x_1 = x, x_2, \ldots, x_r$ все переменные, имеющие вхождения в вывод формулы Z_j, рассмотрим любую совокупность оценок θ_a, отличающихся только значением a, которое ставится в соответствие переменной x. Такие оценки можно записать в виде $\theta_a = \binom{x}{a}\theta\big)$, где θ — общая часть всех этих оценок, относящаяся к переменным $x_2, \ldots, x_r$. Применение любой из оценок θ_a к формулам $X_1, \ldots, X_m$ дает один и тот же результат $X_1\theta, \ldots, X_m\theta$, поскольку переменная x в эти формулы не входит. Пусть все формулы $X_1\theta, \ldots, X_m\theta$ в выбранной модели M истинны. Тогда для любого значения a истинны формулы $X_1\theta_a, \ldots, X_m\theta_a$. Согласно индуктивному предположению для любого значения a истинна формула $Z_j\theta_a$. Следовательно, для данной оценки θ в соответствии с интерпретацией квантора всеобщности истинна формула $(\forall x Z_j)\theta$ или, что то же самое $Y\theta$. Тем самым утверждение теоремы доказано для формулы Y в случае, когда она получена по правилу обобщения. ▶

Следствие 4.1. Если $\Gamma \vdash X$ и все формулы в Γ являются логическими законами, то и X есть логический закон. В частности, если $\vdash X$, то X — логический закон.

Следствие 4.2. Теория $\boldsymbol{P}$ непротиворечива.

◀ Как и в случае исчисления высказываний, последнее следствие показывает, что исчисление предикатов является непротиворечивым. Действительно, если $\vdash X$, то по следствию 4.1 формула X общезначима. Предположение $\vdash \neg X$ приводит к выводу, что и $\neg X$ общезначима. Но нет ни одной формулы, которая общезначима вместе со своим отрицанием. Это непосредственно вытекает из определения: если X общезначима, то, выбрав произвольную модель M, а в ней оценку θ, получим истинную формулу $X\theta$. Но тогда $(\neg X)\theta \equiv \neg X\theta$ ложна, а формула $\neg X$ не является тождественно истинной в M, т.е. $\neg X$ не является общезначимой. ▶

Далее мы докажем, что любая общезначимая формула выводима в исчислении предикатов. Однако воспроизвести доказательство из исчисления высказываний не удастся. Появились новые понятия интерпретации и модели, и нам необходимо понятие, напоминающее вывод из гипотез, но связанное с понятием интерпретации.

Уточним смысл термина аксиоматической формальной теории, понимая под этим пару $T = (\Omega, A)$ из некоторого логико-математического языка Ω, снбженного набором 15 схем аксиом и 2 правил вывода исчисления предикатов, и некоторого множества A замкнутых формул в языке Ω. Формулы из множества A будем называть ***нелогическими аксиомами*** теории. Выводом в теории $T = (\Omega, A)$ будем называть вывод в исчислении предикатов из гипотез Γ, где $\Gamma \subset A$ — конечный список нелогических аксиом. Таким образом, нелогические аксиомы используются в дополнение к стандартному набору логических аксиом (т.е. аксиом исчисления предикатов). Конечная формула X всякого вывода теории называется выводимой в ней, и этот факт мы будем обозначать $T \vdash X$. Выводимые формулы еще называют ***теоремами***

этой теории. Построенное нами исчисление предикатов можно трактовать как формальную аксиоматическую теорию с пустым множеством нелогических аксиом.

Любая интерпретация языка Ω будет также называться интерпретацией формальной аксиоматической теории T. ***Модель*** теории T — это такая интерпретация теории, при которой каждая нелогическая аксиома является истинной (аксиомы замкнуты и не требуют оценки). Для исчисления предикатов любая интерпретация является моделью, поскольку имеет только логические аксиомы, истинные в любой интерпретации.

Если формальная теория имеет модель, то она называется ***интерпретируемой***. Отметим, что если среди нелогических аксиом есть противоречия (т.е. формулы, ложные в любой интерпретации), то такая теория не интерпретируема. ***Теория непротиворечива***, если не существует такой замкнутой формулы X, что $T \vdash X$ и $T \vdash \neg X$.

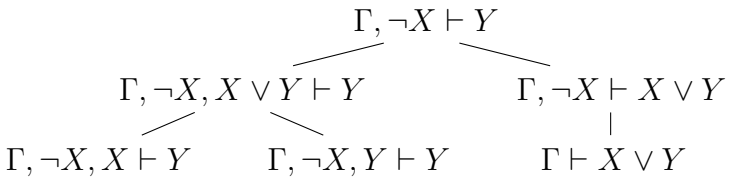
Формальную теорию называют ***полной в узком смысле***, если присоединение любой невыводимой замкнутой формулы к списку нелогических аксиом приводит к противоречивой теории. ***Полнота в широком смысле для исчисления предикатов*** — выводимость в этой теории любой общезначимой формулы.

Покажем, что исчисление предикатов не является полным в узком смысле. Полагаем, что язык этой теории имеет хотя бы один предикатный символ p положительной арности (иначе это исчисление высказываний). Рассмотрим формулу

$$X = \forall x_2 \forall x_3 \ldots \forall x_n (\exists x_1 p(x_1, x_2, \ldots, x_n) \to \forall x_1 p(x_1, x_2, \ldots, x_n)).$$

Нетрудно убедиться в том что эта формула истинна в одних интерпретациях и ложна в других. Присоединим ее в качестве нелогической аксиомы. Тогда получим непротиворечивую теорию T_0. Действительно, если $T_0 \vdash Y$ и $T_0 \vdash \neg Y$, то $X \vdash Y$ и $X \vdash \neg Y$ (X — единственная нелогическая аксиома), откуда в силу правила введения отрицания заключаем, что $\vdash \neg X$. Но это неверно, поскольку $\neg X$ не является общезначимой. В то же время и сама формула X по тем же соображениям не выводима в исчислении предикатов.

Однако исчисление предикатов полно в широком смысле. Доказательство этого факта, известного как ***теорема Геделя о полноте***, опирается на следующую теорему, также доказанную Геделем. Предварительно вспомним дополнительное правило $\dfrac{\Gamma, \neg X \vdash Y}{\Gamma \vdash X \vee Y}$ (см. пример 2.3). Это правило обратимо, т.е. $\dfrac{\Gamma \vdash X \vee Y}{\Gamma, \neg X \vdash Y}$. Действительно, воспользовавшись правилом сечения, получаем:



Теорема 4.2. Любая непротиворечивая формальная аксиоматическая теория имеет счетную модель.

◀ Для построения соответствующей интерпретации нужно выбрать произвольное счетное множество (неважно какое), конкретизировать функциональные символы и константы (опять-таки неважно как), и придать смысл предикатным символам. Последнее является ключевым: необходимо так конкретизировать предикатные символы, что все нелогические аксиомы станут истинными. Ясно, что сосредоточиться надо на символах, входящих в нелогические аксиомы. Разобраться можно, проведя синтаксический анализ нелогических аксиом. Сделать это можно так.

В логико-математический язык Ω формальной теории введем дополнительное счетное множество констант, получив расширенный язык Ψ. Очевидно, что рассматриваемая формальная теория T будет формальной теорией и с языком Ψ, причем непротиворечивой (обозначим ее T_ψ). В этом языке рассмотрим множество D всех замкнутых термов (чтобы такие существовали в нужном количестве, и нужно расширение языка). Это будет носителем нашей интерпретации.

Любопытна следующая интерпретация функциональных символов: функциональному символу f ставим в соответствие отображение, которое термы t_1, t_2, …, t_n преобразует в терм $f(t_1, t_2, \ldots, t_n)$.

Надо построить интерпретацию предикатных символов. Сформируем два множества замкнутых формул L и R, выдерживая следующее условие разделенности: для любого списка X_1, X_2, …, X_n из L и любого списка Y_1, Y_2, …, Y_m из R формула $X_1 \wedge X_2 \wedge \ldots \wedge X_n \to Y_1 \vee Y_2 \vee \ldots \vee Y_m$ не выводима в исчислении $\boldsymbol{P}$ с языком Ψ. Для этого выберем выводимую в исчислении предикатов замкнутую формулу S (например $S = C \vee \neg C$) и сформируем множество L_0 из всех нелогических аксиом теории с дополнительной формулой S и множество R_0, содержащее единственную формулу $\neg S$. В силу непротиворечивости теории множества L_0 и R_0 разделены. Далее строим последовательность пар разделенных множеств L_n, R_n, в которой каждая очередная пара есть расширение предыдущей: $L_n \supset L_{n-1}$, $R_n \supset R_{n-1}$. Затем объединяем две последовательности множеств: $L = \bigcup_{n=1}^{\infty} L_n$, $R = \bigcup_{n=1}^{\infty} R_n$. Очевидно, что если все пары L_n, R_n разделены, то и множества L, R разделены. Опишем процедуру расширения пары множеств L_n, R_n. При этом отметим, что невыводимость в исчислении $\boldsymbol{P}$ формулы $X_1 \wedge X_2 \wedge \ldots \wedge X_n \to Y_1 \vee Y_2 \vee \ldots \vee Y_m$ равносильна тому, что не существует вывода $X_1, X_2, \ldots, X_n \vdash Y_1 \vee Y_2 \vee \ldots \vee Y_m$. Дизъюнкцию всех формул списка Δ будем обозначать V_Δ.

Выбираем произвольную формулу $Z \in L_n \setminus L_{n-1}$. Пусть $Z = U \wedge W$. Для произвольной пары списков $\Gamma \in L_n$ и $\Delta \in R_n$ секвенция $\Gamma \wedge Z \vdash V_\Delta$ невыводима. Это значит, что для любых списков Γ и Δ нет выводимости $\Gamma, U, W \vdash V_\Delta$, и мы можем включить формулы U и W в множество L_{n+1}, не нарушая условия разделенности. Действительно, в противном случае существуют списки Γ_1, Δ_1, для которых, например, секвенция $\Gamma_1, U \vdash V_{\Delta_1}$ явлется выводимой. но тогда $\Gamma_1, U, W \vdash V_{\Delta_1}$ и $\Gamma_1, U \wedge W \vdash V_{\Delta_1}$, что противоречит разделенности множеств L_n и R_n.

Пусть $Z = U \vee W$. Невыводимость $\Gamma, U \vee W \vdash V_\Delta$ означает, что мы можем включить одну из формул U или W в множество L_{n+1}, не нарушая условия разделенности. Действительно, если для пары списков Γ_1, Δ_1 есть выводимость $\Gamma_1, U \vdash V_{\Delta_1}$ а для пары списков Γ_2, Δ_2 — выводимость $\Gamma_2, W \vdash V_{\Delta_2}$, то сразу получаем выводимости $\Gamma_1, \Gamma_2, U \vdash V_{\Delta_1} \vee V_{\Delta_2}$ и $\Gamma_1, \Gamma_2, W \vdash V_{\Delta_1} \vee V_{\Delta_2}$ (по правилам введения гипотез и введения дизъюнкции), откуда $\Gamma_1, \Gamma_2, U \vee W \vdash V_{\Delta_1} \vee V_{\Delta_2}$, что противоречит разделенности L_n и R_n.

Пусть $Z = U \to W$. Тогда мы, не нарушая разделенности, можем включить либо U в R_{n+1}, либо W в L_{n+1}. Если бы это было не так, то существовали бы выводимости $\Gamma_1 \vdash V_{\Delta_1} \vee U$ и $\Gamma_2, W \vdash V_{\Delta_2}$. Значит, есть выводимость $\Gamma_1, \neg V_{\Delta_1} \vdash U$, и мы заключаем, что существует выводимость $\Gamma_1, \Gamma_2, \neg V_{\Delta_1}, U \to W \vdash V_{\Delta_2}$, из которой заключаем о существовании выводимости $\Gamma_1, \Gamma_2, U \to W \vdash V_{\Delta_1} \vee V_{\Delta_2}$, а это противоречит разделенности L_n и R_n.

Пусть $Z = \neg U$. Тогда мы можем включить U в множество R_{n+1}, поскольку из существования выводимости $\Gamma \vdash U \vee V_\Delta$ следует существование выводимости $\Gamma, \neg U \vdash V_\Delta$, что противречит разделенности L_n и R_n.

Пусть $Z = \forall x U$. В множество L_{n+1} включаем все формулы U_c^x, где c пробегает множество всех констант языка. В противном случае для некоторой константы c существует выводимость $\Gamma, U_c^x \vdash V_\Delta$, откуда с учетом $\forall x U \vdash U_c^x$ в силу правила сечения получаем выводимость $\gamma, \forall x U \vdash V_\Delta$, противоречащую разделенности L_n и R_n.

Пусть $X = \exists x U$. Тогда есть такая константа c, для которой добавление U_c^x в L_n не нарушает условия разделенности с R_n. Впрочем, отметим, что если для любой константы c имеет место $\Gamma_c, U_c^x \vdash \Delta_c$, то в силу счетности множества констант сделать заключение о существовании выводимости $\Gamma, \exists x U \vdash \Delta$ мы не сможем, поскольку объединение всех участвующих гипотез может быть счетным. Выход из возникшей проблемы такой. Выберем константу c, не входящую ни в одну из формул множеств L_n и R_n. Тогда существование выводимости $\Gamma, U_c^x \vdash V_\Delta$ с некоторыми списками Γ и Δ равносильно существованию выводимости $\Gamma, U_y^x \vdash \Delta$ с произвольной переменной y, не входящей в формулы списка Γ (мы, не нарушая требований по всему выводу можем заменить c на y). Отсюда заключаем о существовании выводимости $\Gamma, \exists x U \vdash \Delta$, что противоречит разделенности L_n и R_n. Чтобы эта схема работала, надо гарантировать, что

на любом шаге будет существовать свободная константа c. Для этого надо скорректировать действия при анализе формулы вида $Z = \forall x U$. При обработке такой формулы надо включать в L_{n+1} не все формулы U_c^x, а хотя бы одну с нерассмотренной еще константой c, но обеспечить бесконечное количество обработок этой формулы: если в других случаях обработанная формула больше не рассматривается, то эту надо периодически просматривать, генерируя очередные формулы вида U_c^x. Сделать это можно, поскольку и количество формул в рассматриваемом языке, и количество констант счетно.

Рассмотрим теперь формулу $Z \in R_n$. Если $Z = U \wedge W$, то в множество R_{n+1} можно включить одну из формул U или W, не нарушая условия разделенности. Действительно, если существуют выводимости $\Gamma_1 \vdash V_{\Delta_1} \vee U$ и $\Gamma_2 \vdash V_{\Delta_2} \vee W$, то существуют выводимости $\Gamma_2, \neg V_{\Delta_1} \vdash U$ и $\Gamma_2, \neg V_{\Delta_2} \vdash W$, откуда с помощью введения гипотез и введения конъюнкции получаем $\Gamma_1, \Gamma_2, \neg V_{\Delta_1}, \neg V_{\Delta_2} \vdash U \wedge W$ и $\Gamma_1, \Gamma_2 \vdash V_{\Delta_1} \vee V_{\Delta_2} \vee (U \wedge W)$, что противоречит разделенности L_n и R_n.

Если $Z = U \vee W$, то в R_{n+1} включаем и U, и W. Например, если есть выводимость $\Gamma \vdash V_\Delta \vee U$, то сразу $\Gamma \vdash V_\Delta \vee U \vee W$, а порядок дизъюнкций справа не играет роли.

Пусть $Z = U \to W$. Тогда включаем U в L_{n+1}, а W в R_{n+1}. Действительно, если существует выводимость $\Gamma, U \vdash V_\Delta$, то $\Gamma, U \vdash V_\Delta \vee W$, откуда $\Gamma, \neg V_\Delta, U \vdash W$ и в силу теоремы о дедукции $\Gamma, \neg V_\Delta \vdash U \to W$. Следовательно, $\Gamma, \neg V_\Delta \vdash V_\Delta \vee (U \to W)$, что противоречит разделенности L_n и R_n.

Если $Z = \neg U$, формулу U включаем в L_{n+1}: из выводимости $\Gamma, U \vdash V_\Delta$ следует выводимость $\Gamma \vdash V_\Delta \vee \neg U$, которая противоречит разделенности L_n и R_n.

Пусть $Z = \forall x U$. Выбрав константу c, не входящую ни в одну из формул множеств L_n и R_n, заключаем, что нет выводимости $\Gamma \vdash V_\Delta \vee U_c^x$. Действительно, из существования такой выводимости с некоторыми списками Γ и Δ следует выводимость $\Gamma \vdash V_\Delta \vee U_y^x$, где переменная y не входит в формулы списков Γ и Δ, а из последней — выводимость $\Gamma \vdash V_\Delta \vee \forall x U$, противоречащая условию разделенности L_n и R_n. Для выбранной константы в список R_{n+1} включаем формулу U_c^x.

Если $Z = \exists x U$, то в R_{n+1} включаем формулы U_c^x для всех констант c, обеспечив их „постепенное поступление“, как в случае формулы $Z = \forall x U$ из множества L_n.

Построив последовательности разделенных множеств $\{L_n\}$ и $\{R_n\}$, переходим к их объединениям L и R. Эти объединения тоже разделены: если бы существовала выводимость $\Gamma \vdash V_\Delta$ с некоторыми списками Γ и Δ, то в силу конечности списков можно было бы утверждать, что $\Gamma \subset L_k$ и $\Delta \subset R_k$ для некоторого номера k и что множества L_k и R_k не являются разделенными.

Элементарным формулам множества L присвоим истинностное значение 1, элементарным формулам множества R — истинностное значение 0, остальным — неважно. Тогда можно утверждать, что все формулы множества L будут иметь истинностное значение 1, а все формулы множества R — истинностное значение 0. Доказательство этого проводится индукцией по построению формул. Действительно, для элементарных формул это вытекает из заданной интепретации. Пусть $Z \in L$ имеет, например, вид $Z = U \to W$. Тогда в соответствии с построением множеств L и R имеем либо $U \in R$, либо $W \in L$. В первом случае, согласно индуктивному предположению, формула U ложна, а значит, формула $U \to W$ истинна. Во втором случае формула W истинна, поэтому $U \to W$ истинна. Если $Z = U \to W \in R$, то по построению $U \in L$, $W \in R$. Значит, U истинна, а W ложна, откуда заключаем, что Z ложна. Аналогично рассматриваются остальные случаи.

Поскольку множество L содержит все нелогические аксиомы, заключаем, что они в построенной интерпретации истинны, а интерпретация является моделью. ▶

Замечание. Изложенное доказательство верно по сути, но, вообще-то не может быть признано строгим по некоторым соображениям. Дело в том, что в доказательстве идет манипуляция бесконечными множествами, причем на содержательном уровне. Такого рода рассуждения в современной математической логике не считаются достаточно надежными. Однако следует учесть, что все множества, рассматриваемые в доказательстве, являются счетными. Поэтому построение множеств L и R, обладающих свойством замкнутости (в определенном смысле)

можно провести так, что на каждом шаге мы будем иметь дело с конечными множествами. Требуется аккуратная работа с нумерациями счетных множеств, но эти детали затушевывают основную идею доказательства.

Доказанная теорема позволяет сделать вывод о полноте исчисления предикатов в широком смысле.

Теорема 4.3. В исчислении предикатов выводима любая тавтология.

◀ Пусть общезначимая формула A не выводима в теории $\boldsymbol{P}$. Мы можем считать, что эта формула замкнута, добавляя к ней в случае необходимости кванторные приставки. Замкнутая формула $\neg A$, не являясь общезначимой, тоже не выводима в теории $\boldsymbol{P}$. Рассмотрим формальную теорию T, которая получается из $\boldsymbol{P}$ добавлением нелогической гипотезы $\neg A$. Эта теория непротиворечива, так как иначе в ней выводилась бы любая формула, в том числе и A. Другими словами, в исчислении предикатов имела бы место выводимость $\neg A \vdash A$. Но это возможно только в случае, когда A — выводимая формула. Но если T — непротиворечивая теория, то она имеет модель, т.е. такую реализацию, в которой формула $\neg A$ истинна. В этой же реализации формула A ложна. Следовательно, она не является общезначимой. Полученное противоречие доказывает, что предположение о невыводимости A неверно. ▶

Исчисление предикатов не является разрешимой теорией. Это становится понятно, если учесть, что средств исчисления предикатов достаточно, чтобы представить содержание практически любой математической теории. Было бы странно предполагать, что есть некий формальный алгоритм, по которому можно любую теорему проверить, верна она или нет. Это, разумеется, расходится с нашим представлением о бесконечном разнообразии окружающего мира. Однако строгое доказательство этого факта выходит за рамки курса: требуется строгое математическое описание понятия „алгоритм“ и развитая теория об алгоритмах, чтобы можно было строго показать неразрешимость исчисленния предикатов.

Но здесь важно одно обстоятельство. Доказанная теорема устанавливает эквивалентность понятий „выводимая“ и „истинная“. Значит, метод, которым разворачивалась невыводимая формула может быть использован (с некоторыми модификациями) для анализа формулы на истинность. Поясним сказанное. Ставим задачу убедиться в общезначимости некоей формулы X или опровергнуть это. Последнее означает построение интерпретации, в которой эта формула ложна. Чтобы работать только с замкнутыми формулами, можем эту формулу замкнуть кванторами всеобщности. Что значит ложность X? Допустим $X = X_1 \vee X_2$. Тогда ложность X равносильна ложности и X_1, и X_2. Допустим $X = X_1 \to X_2$. Тогда ложность X равносильна истинности X_1 и ложности X_2. Эти рассуждения показывают, что исходная формула разворачивается в два множества L и R более простых формул, причем все формулы в L должны быть истинны, а все формулы в L_2 ложны. Разворачивание сопровождается уменьшением логической сложности формул, и в конце концов дело дойдет до элементарных формул. Однако при этом возникает несколько вариантов (например, истинность $U \vee W$ обеспечивается истинностью U или истинностью W; какой из этих вариантов возможен, на этапе анализа формулы не известно; надо иметь в виду все варианты) Если при этом множества L и R будут пересекаться, то мы приходим к невозможности обеспечить главное требование: истинность формул из L и ложность формул из R. Это означает, что интерпретаций, в которых исходная формула истинна, нет. Если же пересечений нет, то можно так устроить интерпретацию предикатных символов, что основное требование будет выполняться.

Пример 4.3. Рассмотрим формулу $\exists x(p \to q(x)) \to (p \to \forall x q(x))$. Попробуем построить контрпример, т.е. интерпретацию, в которой она ложна, или убедиться в ее истинности. Редукцию формулы удобно построить в виде дерева, в котором преобразуется формула вида $\Gamma \Rightarrow \Delta$, где Γ и Δ — конечные списки формул. Слева — формулы множества L, справа — формулы множества

R. Получится следующее:

$$\begin{array}{c}
\vdash\!\!> \exists x(p \to q(x)) \to (p \to \forall x q(x)) \\
| \\
\exists x(p \to q(x)) \vdash\!\!> p \to \forall x q(x) \\
| \\
\exists x(p \to q(x)), p \vdash\!\!> \forall x q(x) \\
| \\
p \to q(a_0), p \vdash\!\!> \forall x q(x) \\
| \\
p \to q(a_0), p \vdash\!\!> q(a_1) \\
\diagup \qquad\qquad \diagdown \\
p \vdash\!\!> p, q(a_1) \qquad\qquad q(a_0), p \vdash\!\!> q(a_1)
\end{array}$$

Мы видим, что в левой ветке формула p встречается и слева, и справа, и в этом случае обеспечить истинность формул левее $\vdash\!\!>$ и ложность правее нельзя. В то же время правая ветка обеспечивает нужную интерпретацию: предикатный символ p нулевой арности полагаем равным 1, а для q полагаем $q(a_0) = 1$, $q(a_1) = 0$. Таким образом, если носитель интерпретации содержит два элемента, а предикатные символы имеют заданный смысл, формула будет ложной. Впрочем, если носитель интерпретации имеет всего лишь один элемент (а почему бы и нет?), то нужное условие не удается обеспечить и в правой ветви. Что это значит? В любой момент в левом и правом списках формул есть формулы эквивалентные. Если две такие формулы истинны, то дизъюнкция всех правых формул истинна, т.е. истинна исходная формула. Если же обе формулы ложны, то конъюнкция левых формул ложна, откуда следует истинность исходной формулы. Вывод: анализируемая формула истинна при любой интерпретации.

Разобранный пример можно истолковать как алгоритм проверки формулы на истинность. Это верно, но этот алгоритм не всегда приводит к ответу за конечное число шагов.

Пример 4.4. Рассмотрим формулу $\neg\forall x\exists y p(x, y)$. Для нее дерево разбора имеет следующий вид:

$$\begin{array}{c}
\vdash\!\!> \neg\forall x\exists y p(x, y) \\
| \\
\forall x\exists y p(x, y) \vdash\!\!> \\
| \\
\exists y p(c_0, y), \forall x\exists y p(x, y) \vdash\!\!> \\
| \\
p(c_0, c_1), \forall x\exists y p(x, y) \vdash\!\!> \\
| \\
p(c_0, c_1), \exists y p(c_1, y), \forall x\exists y p(x, y) \vdash\!\!> \\
| \\
p(c_0, c_1), p(c_1, c_2), \forall x\exists y p(x, y) \vdash\!\!> \\
| \\
\ldots\ldots\ldots\ldots\ldots\ldots
\end{array}$$

При разборке квантора всеобщности мы выбираем первую не реализованную для этого квантора константу, а при разборке квантора существования — первую свободную константу. Для построения контрпримера есть два варианта. Можно в очередной момент вместо свободной константы использовать прежнюю (считать, что новых констант уже нет), например, считать, что $c_2 = c_1$. Тогда истинность формул $p(c_0, c_1)$ и $p(c_1, c_1)$ в интерпретации с двумя элементами c_1 и c_2 будет означать и истинность формулы $\forall x\exists y p(x, y)$. Мы получаем контрпример, т.е. интерпретацию, в которой анализируемая формула $\neg\forall x\exists y p(x, y)$ является ложной. Возможен и другой вариант — счетный набор констант. Мы получаем набор формул $p(c_k, c_{k+1})$, которые объявляем истинными. Считая, что других констант нет, делаем заключение об истинности формулы $\forall x\exists y p(x, y)$ и ложности формулы $\neg\forall x\exists y p(x, y)$. Мы снова получили контрпример, но уже со счетным носителем интерпретации.

Однако, если бы проверяемая формула была тавтологией, мы должны были бы рассмотреть оба этих варианта, за конечное число шагов установить истинность формулы нам бы не удалось.

Процесс „расшифровки“, предпринятый в разобранном примере очень похож на построение вывода в рамках правил естественного вывода, но с одним уточнением. В секвенциях исчисления предикатов справа могла быть только одна формула, в то время как в разобранном примере и слева, и справа находятся списки формул.

Приведенное построение, а также теория $\boldsymbol{N}$, в которой дело свелось к манипуляциям с секвенциями, приводит к идее построить совсем другое исчисление, основанное на понятии секвенции. Такое исчисление называют ***исчислением генценовского типа*** в противовес исчислениям $\boldsymbol{N}$ и $\boldsymbol{P}$, которые относят к ***исчислениям гильбертовского типа***.

5. ПРИМЕРЫ ФОРМАЛЬНЫХ ТЕОРИЙ

Рассмотрим построение формальных теорий, описывающих некоторые конкретные математические дисциплины.

5.1. Теория групп

Подбор составляющих любой формальной теории — творческий процесс, не дающий однозначного результата. В основе, конечно, лежат содержательные аксиомы соответствующей дисциплины. В теории групп встречаются объекты одного типа. Значит, можно использовать один сорт переменных. Для обозначения операции группы нужен один функциональный символ арности 2. Но есть еще две вторичные операции, точнее очевидная константа и унарная операция вычисления обратного элемента. Для записей условий нужен хотя бы один предикатный символ и наиболее уместным является символ равенства. Это бинарный символ, но чтобы бинарный предикатный символ мог расцениваться именно как знак равенства, нужны некоторые требования (аксиомы).

Учитывая вышесказанное опишем формальную теорию групп $\boldsymbol{G}$. Рассмотрим односортный логико-математический язык с одной константой, которую мы обозначим через e, двумя функциональными символами $m(x,y)$ и $i(x)$ и одним бинарным предикатным символом $p(x,y)$. Правила вывода — стандартные для исчисления предикатов. Запишем нелогические аксиомы:

1) $p(x,y) \to (p(x,z) \to p(y,z))$;
2) $p(x,y) \to p(m(x,z), m(y,z))$;
3) $p(x,y) \to p(m(z,x), m(z,y))$;
4) $p(m(m(x,y),z), m(x,m(y,z)))$;
5) $p(m(x,e),x)$;
6) $p(m(x,i(x)),e)$.

Напомним, что нелогические аксиомы любой формальной теории должны быть замкнутыми формулами. Поэтому здесь неявно предполагается, что аксиомы теории групп замкнуты кванторами весобщности по всем свободным переменным.

Запись нелогических аксиом прямо следует описанию логико-математического языка, но неудобна для восприятия человеком. Введем альтернативные обозначения: $x * y$ вместо $m(x,y)$, x^{-1} вместо $i(x)$, 1 вместо e и $x = y$ вместо $p(x,y)$. Тогда перечисленные аксиомы будут выглядеть следующим образом:

1° $(x = y) \to ((x = z) \to (y = z))$;
2° $(x = y) \to (x * z = y * z)$;
3° $(x = y) \to (z * x = z * y)$;
4° $(x * y) * z = x * (y * z)$;
5° $x * 1 = x$;
6° $x * x^{-1} = 1$.

Из рассмотренных аксиом (в любой записи) первая характеризует предикат, вторая и третья связывают предикат с основной операцией, а последние три фиксируют свойства операций группы. Вообще говоря, бинарный предикат определяет на носителе интерпретации отношение, и если мы хотим рассматривать предикат как равенство, указанное отношение должно быть отношением эквивалентности, т.е. должны быть истинными утверждения о рефлексивности, симметричности и транзитивности. В аксиоме 1 можно усмотреть транзитивность, но не хватает двух других свойств.

Не следует ожидать, что выбором некоторых формул в качестве аксиом можно гарантировать, что рассматриваемый предикат будет именно равенством, т.е. истинным при совпадении объектов, обозначаемых двумя термами. Для теории групп такие попытки даже вредны. Дело в том, что термин „равны“ следует понимать как „неразличимы“, т.е. это могут быть разные, вообще говоря, объекты, но подпадающие под одно понятие. В математике формирование понятий связано с отношением эквивалентности. Так и здесь: отношение $x = y$ на самом деле — это отношение эквивалентности.

Отметим, что в $\boldsymbol{G}$ сохраняются все правила естественного вывода, а доказательство теорем представляет собой частный вывод, в котором роль гипотез играют нелогические аксиомы. Для сокращения вывода сами нелогические аксиомы удобно заменить некоторыми правилами вывода, аналогичными правилам естественного вывода. Так, аксиомам 2 и 3 соответствуют правила

$$\frac{\Gamma \vdash X = Y}{\Gamma \vdash X * Z = Y * Z}, \qquad \frac{\Gamma \vdash X = Y}{\Gamma \vdash Z * X = Z * Y}, \tag{5.1}$$

а аксиомы 4–6 трансформируются в схемы выводимых формул:

$$\vdash (X * Y) * Z = X * (Y * Z), \qquad \vdash X * 1 = X, \qquad \vdash X * X^{-1} = 1. \tag{5.2}$$

Например, первое из правил (5.1) устанавливается следующим образом:

$$\begin{array}{l}
\Gamma \vdash X * Z = Y * Z \\
\quad\diagup\quad\diagdown \\
\Gamma \vdash X = Y \qquad \Gamma \vdash X = Y \to X * Z = Y * Z \\
\qquad\qquad\qquad\quad | \\
\qquad\qquad\qquad \vdash X = Y \to X * Z = Y * Z \\
\qquad\qquad\qquad\quad | \\
\qquad\qquad\qquad \vdash \forall z(X = Y \to X * z = Y * z) \\
\qquad\qquad\qquad\quad | \\
\qquad\qquad\qquad \vdash \forall y \forall z(X = y \to X * z = y * z) \\
\qquad\qquad\qquad\quad | \\
\qquad\qquad\qquad \vdash \forall x \forall y \forall z(x = y \to x * z = y * z)
\end{array}$$

Обращаем внимание на переход от переменной (например x) к произвольной формуле (в данном случае X) через снятие квантора с подстановкой. Аналогично устанавливаются секвенции (5.2). приведем вывод второй из них:

$$\vdash X * 1 = X \quad \Leftarrow \quad \forall x(x * 1 = x).$$

Аксиома 1 позволяет установить правила

$$\vdash X = X, \quad \frac{\Gamma \vdash X = Y}{\Gamma \vdash Y = X}, \quad \frac{\Gamma \vdash X = Y; \Gamma \vdash Y = Z}{\Gamma \vdash X = Z},$$

выражающие свойства, характеризующие отношение эквивалентности (рефлексивность, симметричность, транзитивность).

В принципе формулу $X = X$ (точнее $\forall x(x = x)$) следовало бы ввести как аксиому поддержки формального равенства. Однако в данном случае эта формула оказывается выводимой в теории $\boldsymbol{G}$. Действительно,

$$\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad \vdash X = X \\
\qquad\qquad\qquad\qquad\qquad\quad\diagup\qquad\diagdown \\
\qquad\qquad\qquad\qquad \vdash X * 1 = X \to X = X \qquad \vdash X * 1 = X \\
\qquad\qquad\qquad\quad\diagup\qquad\diagdown \\
\vdash X * 1 = X \to (X * 1 = X \to X = X) \qquad \vdash X * 1 = X \\
\qquad\qquad | \\
\vdash \forall z(X * 1 = X \to (X * 1 = z \to X = z)) \\
\qquad\qquad | \\
\vdash \forall y \forall z(X * 1 = y \to (X * 1 = z \to y = z)) \\
\qquad\qquad | \\
\vdash \forall x \forall y \forall z(x = y \to (x = z \to y = z))
\end{array}$$

Используя доказанную рефлексивность, можно доказать и симметричность:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash Y = X}{\Gamma \vdash X = Y \qquad X = Y \vdash Y = X}}{X = Y, X = X \vdash Y = X \qquad \vdash X = X}}{X = Y \vdash X = X \to Y = X}}{\vdash X = Y \to (X = X \to Y = X)}}{\vdash \forall x \forall y \forall z (x = y \to (x = z \to y = z)}$$

Транзитивность можно получить из аксиомы 1 с доказанным свойством симметричности:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash X = Z}{\Gamma \vdash X = Y \wedge Y = Z \qquad X = Y \wedge Y = Z \vdash X = Z}}{\Gamma \vdash X = Y \qquad \Gamma \vdash Y = Z \qquad X = Y, Y = Z \vdash X = Z}}{Y = X, Y = Z \vdash X = Z \qquad X = Y \vdash Y = X}}{Y = X \vdash Y = Z \to X = Z \qquad X = Y \vdash X = Y}}{\vdash Y = X \to (Y = Z \to X = Z)}}{\vdash \forall x \forall y \forall z (x = y \to (x = z \to y = z)}$$

Действие симметричности, рефлексивности и транзитивности можно записывать как цепочку равенств: $X = Y = Z = \ldots$

Отметим, что существование левой единицы и левого обратного в аксиомах не оговорено. Однако это можно получить из существования правой единицы и правого обратного для каждого элемента. Действительно, докажем, что правый обратный является левым:

$$x^{-1} * x = (x^{-1} * x) * 1 = (x^{-1} * x) * (x^{-1} * (x^{-1})^{-1}) = x^{-1} * \big(x * (x^{-1} * (x^{-1})^{-1})\big) =$$
$$= x^{-1} * \big((x * x^{-1}) * (x^{-1})^{-1}\big) = x^{-1} * \big(1 * (x^{-1})^{-1}\big) = (x^{-1} * 1) * (x^{-1})^{-1} = x^{-1} * (x^{-1})^{-1} = 1.$$

Теперь докажем, что правая единица является левой:

$$1 * x = (x * x^{-1}) * x = x * (x^{-1} * x) = x * 1 = x.$$

Остальные свойства могут быть получены использованием цепи равенств. Докажем формулу $(A * X = A) \to (X = 1)$, означающую, что правая единица единственная. Содержательное рассуждение таково. Если $A * X = A$, то $A^{-1} * (A * X) = A^{-1} * A = 1$. Но $A^{-1} * (A * X) = (A^{-1} * A) * X = 1 * X = X$. Поэтому $X = 1$.

Соответствующий вывод формулы $(A{*}X = A){\to}(X = 1)$ может быть таким (для сокращения знак умножения опущен):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash AX = A \to X = 1}{AX = A \vdash X = 1}}{AX = A \vdash X = A^{-1}A \qquad AX = A \vdash A^{-1}A = 1}}{AX = A \vdash X = A^{-1}AX \qquad AX = A \vdash A^{-1}AX = A^{-1}A \qquad \vdash A^{-1}A = 1}}{AX = A \vdash A^{-1}AX = X \qquad AX = A \vdash AX = A}}{\vdash A^{-1}AX = 1 * X \qquad \vdash 1 * X = X}}{\vdash A^{-1}A = 1}$$

5.2. Формальная арифметика

Рассмотрим формальную теорию ***AP***, описывающую элементарную теорию чисел (арифметику), т.е. теорию натуральных чисел с арифметическими операциями. Отметим, что при построении формальных систем всегда руководствуются некоторой реальной математической дисциплиной, которая в таком случае обеспечивает ***естественную интерпретацию*** строящейся теории. Построенная теория может иметь множество интерпретаций (как теория ***G***). Но может быть (а в некоторых случаях должно быть) так, что все интерпретации являются изоморфными, т.е. между этими интерпретациями есть взаимно однозначное соответствие, сохраняющее все свойства этих интерпретаций.

Так должно быть в случае формальной арифметики, поскольку наличие нескольких моделей натуральных чисел, ясно различающихся по своим свойствам, вряд ли можно считать приемлемым. Подчеркнем, что под натуральным мы будем понимать целое неотрицательное число (т.е. включая нуль).

Логико-математический язык должен быть односортным. Есть одна константа, которую мы обозначим через 0, две основные арифметические операции сложение и умножение реализуем двумя бинарными функциональными символами, которые будем обозначать в инфиксной форме знаками $+$ и $\cdot$ (этот знак по традиции опускается между двумя переменными). Описание двух операций кольца недостаточно. Характерным свойством натуральных чисел является переход к следующему натуральному числу, вытекающий из процесса пересчета предметов. Это отразим унарным функциональным символом, который будем обозначать в виде x^+. Наконец, нужен предикатный символ, в качестве которого выбираем формальное равенство.

В основе нашей формальной теории лежит аксиоматика натуральных чисел Пеано, включающая пять аксиом:

P1. 0 — натуральное число.

P2. Если n — натуральное число, то n^+ — натуральное число.

P3. Если m, n — натуральные числа и $m^+ = n^+$, то $m = n$.

P4. Если n — натуральное число, то $n^+ \neq 0$.

P5. Пусть A — некоторое свойство, которым могут обладать натуральные числа. Предположим, что: а) 0 обладает свойством A; б) из того, что n обладает свойством A, вытекает, что и n^+ обладает свойством A. Тогда любое натуральное число обладает свойством A.

Нетрудно убедиться в том, что все эти аксиомы необходимы. Первые две дают, по существу, индуктивное определение натуральных чисел. Отсутствие аксиомы P3 приводит к существованию древовидной модели, в которой некоторые ветви начинаются не с нуля. Аксиома P4 препятствует появлению циклической модели (например, комплексных корней n-й степени из 1). Наконец, аксиома P5 обеспечивает связность натурального ряда, т.е. свойство, что любое натуральное число получается с помощью аксиом P1 и P2. Она же лежит в основе принципа математической индукции. С помощью этой аксиомы вводятся основные арифметические операции — сложение и умножение:

P6. $m + 0 = m$;

P7. $m + n^+ = (m + n)^+$;

P8. $m \cdot 0 = 0$;

P9. $m \cdot n^+ = m \cdot n + m$.

При построении формальной теории первые две аксиомы Пеано не нужны, поскольку реализуются на уровне логико-математического языка. Например, первая аксиома фактически утверждает, что в языке арифметики есть константа. В то же время в состав аксиом необходимо включить аксиомы поддержки формального равенства. Кроме того, в состав аксиом мы включаем формулы определения арифметических операций. Все аксиомы, кроме аксиомы индукции, должны быть замкнутыми формулами (хотя для краткости мы кванторы опустим,

неявно полагая, что формулы замкнуты кванторами всеобщности по всем свободным переменным). Исключение составляет аксиома индукции. Эта аксиома связана с произвольным свойством, которое характеризуется некоторым предикатом A, и задается некоторой схемой.

В результате мы получаем одну схему аксиом и восемь аксиом:

A1. $\big(A_0^x \wedge \forall x(A \to A_{x^+}^x)\big) \to A$.
A2. $(x^+ = y^+) \to (x = y)$.
A3. $\neg(x^+ = 0)$.
A4. $(x = y) \to ((x = z) \to (y = z))$.
A5. $(x = y) \to (x^+ = y^+)$.
A6. $x + 0 = x$.
A7. $x + y^+ = (x + y)^+$.
A8. $x \cdot 0 = 0$.
A9. $xy^+ = xy + x$.

Как и в случае теории $\boldsymbol{G}$, эти аксиомы удобно преобразовать в правила вывода:

C1. $\dfrac{\Gamma \vdash A_0^x;\ \Gamma \vdash \forall x(A \to A_{x^+}^x)}{\Gamma \vdash A}$;
C2. $\dfrac{\Gamma \vdash X^+ = Y^+}{\Gamma \vdash X = Y}$;
C3. $\vdash \neg(X^+ = 0)$;
C4. $\vdash X = X$;
C5. $\dfrac{\Gamma \vdash X = Y}{\Gamma \vdash Y = X}$;
C6. $\dfrac{\Gamma \vdash X = Y;\ \Gamma \vdash Y = Z}{\Gamma \vdash X = Z}$;
C7. $\dfrac{\Gamma \vdash X = Y}{\Gamma \vdash X^+ = Y^+}$;
C8. $\vdash X + 0 = X$;
C9. $\vdash X + Y^+ = (X + Y)^+$;
C10. $\vdash X \cdot 0 = 0$;
C11. $\vdash X \cdot Y^+ = X \cdot Y + X$.

Все эти правила устанавливаются так же как и в теории групп. Например правило C1:

$$\begin{array}{ccc}
 & \Gamma \vdash A & \\
\swarrow & & \searrow \\
\Gamma \vdash A_0^x \wedge \forall x(A \to A_{x^+}^x) & & \Gamma \vdash A_0^x \wedge \forall x(A \to A_{x^+}^x) \to A \\
\swarrow \quad \searrow & & \big| \\
\Gamma \vdash A_0^x \qquad \Gamma \vdash \forall x(A \to A_{x^+}^x) & & \vdash A_0^x \wedge \forall x(A \to A_{x^+}^x) \to A \\
 & & \big| \\
 & & \vdash \forall x(A_0^x \wedge \forall x(A \to A_{x^+}^x) \to A)
\end{array}$$

Отметим, что в этом правиле выбор переменной x не является существенным: последний шаг может идти с переименованием. На практике в качестве переменной x следует выбирать любую, не входящую свободно в формулы списка Γ.

Правила C4–C6 отражают характерные черты формального равенства как предиката для отношения эквивалентности, они полностью определяются аксиомами A4 и A6. Вывод этих правил повторяет вывод соответствующих правил формальной теории групп.

Многие ключевые факты арифметики (например, коммутативность сложения) вытекают из сформулированных аксиом. В их доказательстве особую роль играет схема аксиом, отражающая принцип математической индукции. Рассмотрим некоторые формулы.

1. $(X = Y) \to (X + Z = Y + Z)$.

◀ На содержательном уровне рассуждение таково. При $z = 0$ равенство $X + z = Y + z$ сводится к очевидному равенству $X + 0 = Y + 0$, вытекающему из цепочки $X + 0 = X = Y = Y + 0$. Если верно равенство $X + z = Y + z$, то $X + z^+ = (X + z)^+ = (Y + z)^+ = Y + z^+$. На основании метода математической индукции делаем вывод, что равенство $X + z = Y + z$ верно для любого z. Формальный вывод рассматриваемой формулы строится на сведении к правилу C1 и может выглядеть так:

$$\vdash X = Y \to X + Z = Y + Z$$
$$\vdash X = Y \to \forall z(X + z = Y + z)$$
$$\vdash \forall z(X = Y \to X + z = Y + z)$$
$$\vdash \forall z(X = Y \to \forall z(X + z = Y + z \to X + z^+ = Y + z^+))$$
$$\vdash \forall z(X = Y \to \forall z(X + z = Y + z \to X + z^+ = Y + z^+))$$

Обе ветки заканчиваются секвенциями, которые можно обосновать цепочкой равенств. Ветка $\boxed{1}$ подробно раскрывается так:

$$X = Y \vdash X + 0 = Y + 0$$
$$X = Y \vdash X + 0 = X \qquad X = Y \vdash X = Y + 0$$
$$\vdash X + 0 = X \qquad X = Y \vdash X = Y \qquad X = Y \vdash Y = Y + 0$$
$$\vdash Y = Y + 0$$
$$\vdash Y + 0 = Y$$

Ветка $\boxed{2}$ ракрывается аналогично (список Γ содержит одну формулу $X + z = Y + z$):

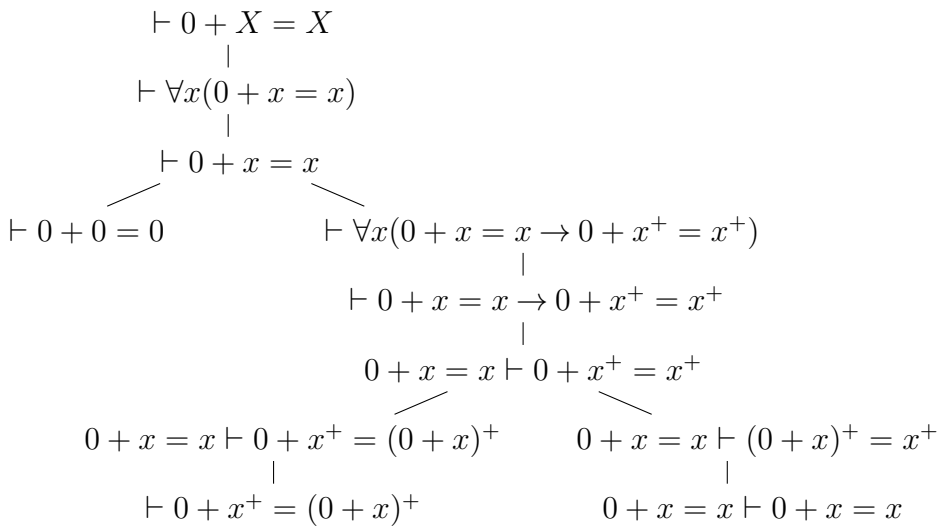
$$\Gamma \vdash X + z^+ = Y + z^+$$
$$\Gamma \vdash X + z^+ = (X + z)^+ \qquad \Gamma \vdash (X + z)^+ = Y + z^+$$
$$\vdash X + z^+ = (X + z)^+ \qquad \Gamma \vdash (X + z)^+ = (Y + z)^+ \qquad \Gamma \vdash (Y + z)^+ = Y + z^+$$
$$\Gamma \vdash X + z = Y + z \qquad \vdash (Y + z)^+ = Y + z^+$$
$$\vdash Y + z^+ = (Y + z)^+$$

Как видим, формальный вывод заметно длиннее содержательного. ▶

2. $0 + X = X$.

◀ Содержательное доказательство основано на замене Z переменной и применении индукции по этой переменной. Имеем $0 + 0 = 0$. Если $0 + x = x$, то $0 + x^+ = (0 + x)^+ = x^+$. Значит,

$0 + x = x$. Формальное доказательство следующее:



Отметим переход от формулы X к переменной x, совершенный в начале вывода. Это стандартная подготовка к применению идукции, состоящая в замене одной из подформул переменной, которая становится переменной индукции. ►

3. $X^+ + Y = X + Y^+$.

◄ Ограничимся содержательным рассуждением. Имеем $x^+ + 0 = x^+ = (x + 0)^+ = x + 0^+$. Пусть $x^+ + y = x + y^+$. Тогда $x^+ + y^+ = (x^+ + y)^+ = (x + y^+)^+ = x + y^{++}$. Таким образом, $x^+ + y = x + y^+$. ►

4. $X = Y \to Z + X = Z + Y$.

◄ Содержательное рассуждение можно строить на переменной z, поставленной взамен формулы Z. при $z = 0$ имеем $0 + X = X = Y = 0 + Y$ (использовано уже установленное свойство 2). Пусть доказано, что $z + X = z + Y$. Тогда, согласно свойству 3

$$z^+ + X = z + X^+ = (z + X)^+ = (z + Y)^+ = z + Y^+ = z^+ + Y.$$

В соответствии с методом математической индукции, если $X = Y$, то $z + X = z + Y$. ►

5. $(X + Y) + Z = X + (Y + Z)$.

◄ Содержательное доказательство основано на методе математической индукции по переменной z, включенной в формулу взамен Z. Согласно свойству 4 имеем: $(X + Y) + 0 = X + Y = = X + (Y + 0)$, поскольку $Y = Y + 0$. Предположим $(X + Y) + z = X + (Y + z)$. Тогда

$$(X + Y) + z^+ = ((X + Y) + z)^+ = ((X + (Y + z))^+ = X + (Y + z)^+ = X + (Y + z^+).$$

В соответствии с методом математической индукции заключаем, что $(X+Y)+z = X+(Y+z)$. ►

6. $X + Y = Y + X$.

◄ Заменяем формулу Y индутивной переменной y. При $y = 0$ имеем $X + 0 = X = 0 + X$. Если $X + y = y + X$, то $X + y^+ = (X + y)^+ = (y + X)^+ = y + X^+ = y^+ + X$. Значит, $X + y = y + X$ для любого значения переменной y. ►

7. $X + Y = 0 \to X = 0$.

◄ Заменяем формулу X индуктивной переменной x, При $x = 0$ имеем формулу $0+Y = 0\to 0 = 0$, которая получается из истинной формулы $0 = 0$ присоединением посылки. Пусть формула

$x + Y = 0 \to x = 0$ истинна. Тогда

$$\vdash (x^+ + Y = 0) \to (x^+ = 0)$$
$$\Big|$$
$$x^+ + Y = 0 \vdash x^+ = 0$$

$$x^+ + Y = 0 \vdash (x + Y)^+ = 0 \qquad (x + Y)^+ = 0 \vdash x^+ = 0$$

$$x^+ + Y = 0 \vdash (x + Y)^+ = x^+ + Y \qquad x^+ + Y = 0 \vdash x^+ + Y = 0 \qquad \Gamma \vdash (x + Y)^+ = 0 \qquad \Gamma \vdash \neg((x + Y)^+ = 0)$$
$$\Big| \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \Big|$$
$$\vdash (x + Y)^+ = x^+ + Y \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdash \neg((x + Y)^+ = 0)$$

Следовательно, формула $x + Y = 0 \to x = 0$ истинна при любом x. Отметим, что индуктивное предположение об истинности формулы $x + Y = 0 \to x = 0$ фактически не нужно: общезначимость формулы $(x^+ + y = 0) \to (x^+ = 0)$ устанавливается без использования аксиомы индукции. ►

8. $0 \cdot X = 0$.

◄ Заменяем формулу X индуктивной переменной x. При $x = 0$ равенство $0 \cdot 0 = 0$ вытекает из правила C10. Предположим $0 \cdot x = 0$. Тогда $0 \cdot x^+ = 0 \cdot x + 0 = 0 + 0 = 0$. Значит, формула $0 \cdot x = 0$ истинна для любого значения переменной x. ►

9. Введем обозначение $1 = 0^+$. Тогда $X \cdot 1 = X$ и $1 \cdot X = X$.

◄ Для первого равенства имеем $x \cdot 1 = x \cdot 0^+ = x \cdot 0 + x = 0 + x = x$. Второе равенство $1 \cdot x$ верно при $x = 0$ (по аксиоме). Пусть $1 \cdot x = x$. Тогда $1 \cdot x^+ = 1 \cdot x + 1 = 1 \cdot x + 0^+ = x + 0^+ = (x + 0)^+ = x^+$. Значит, $1 \cdot x = 1$. ►

10. $X^+ \cdot Y = X \cdot Y + Y$.

◄ Содержательное доказательство проводится индукцией по переменной y, включенной в формулу взамен подформулы Y. При $y = 0$ имеем $X^+ \cdot 0 = 0 = 0 + 0 = X \cdot 0 + 0$. Пусть $X^+ \cdot y = X \cdot y + y$. Тогда

$$X^+ \cdot y^+ = X^+ \cdot y + X^+ = (X \cdot y + y) + X^+ = X \cdot y + (y + X^+) =$$
$$+ X \cdot y + (y^+ + X) = X \cdot y + (X + y^+) = (X \cdot y + X) + y^+ = X \cdot y^+ + y^+.$$

Поэтому формула $X^+ \cdot y = X \cdot y + y$ верна при любом значении переменной y. ►

11. Умножение коммутативно: $X \cdot Y = Y \cdot X$.

◄ Содержательное доказательство проводится индукцией по переменной x, включенной в формулу взамен X. При $x = 0$ имеем $x \cdot Y = 0 \cdot Y = 0 = Y \cdot 0 = Y \cdot x$, т.е. при $x = 0$ формула верна. Пусть $x \cdot Y = Y \cdot x$. Тогда

$$x^+ \cdot Y = x \cdot Y + Y = Y \cdot x + Y = Y \cdot x^+.$$

Следовательно, формула $x \cdot Y = Y \cdot x$ верна при любом значении переменной x. ►

12. Умножение дистрибутивно относительно сложения: $(X + Y) \cdot Z = X \cdot Z + Y \cdot Z$.

◄ Доказательство также строится на принципе математической индукции по переменной z, включенной в формулу взамен подформулы Z. При $z = 0$ имеем

$$(X + Y) \cdot 0 = 0 = 0 + 0 = X \cdot 0 + Y \cdot 0.$$

Пусть $(X + Y) \cdot z = X \cdot z + Y \cdot z$. Тогда

$$(X+Y) \cdot z^+ = (X+Y) \cdot z + (X+Y) = (X \cdot z + Y \cdot z) + (X+Y) = (X \cdot z + X) + (Y \cdot z + Y) = X \cdot z^+ + Y \cdot z^+.$$

Следовательно, формула $(X+Y)\cdot z = X\cdot z + Y\cdot z$ истинна при любом значении переменной z. ▶

На множестве натуральных чисел существует отношение порядка. Можно рассмотреть формулу $p(x,y) \equiv \exists z(x+z=y)$ и переобозначить ее как $x \leqslant y$. Двуместный предикатный символ можно интерпретировать как бинарное отношение. Это отношение оказывается рефлексивным, транзитивным и антисимметричным, т.е. истинны формулы $x \leqslant x$, $(x \leqslant y) \wedge (y \leqslant z) \to (x \leqslant z)$ и $(x \leqslant y) \wedge (y \leqslant x) \to (x = y)$. Рефлексивность устанавливается просто:

$$(x \leqslant x) \equiv \exists z(x+z=x) \quad \Leftarrow \quad x+0=x.$$

Упражнение на транзитивность посложнее:

$$\vdash \exists u(x+u=y) \wedge \exists u(y+u=z) \to \exists u(x+u=z)$$
$$\exists u(x+u=y), \exists u(y+u=z) \vdash \exists u(x+u=z)$$
$$x+u_1=y, y+u_2=z \vdash \exists u(x+u=z)$$
$$x+u_1=y, y+u_2=z \vdash x+(u_1+u_2)=z$$
$$x+u_1=y, y+u_2=z \vdash (x+u_1)+u_2=z$$
$$x+u_1=y, y+u_2=z \vdash (x+u_1)+u_2=y+u_2 \qquad x+u_1=y, y+u_2=z \vdash y+u_2=z$$
$$x+u_1=y, y+u_2=z \vdash x+u_1=y$$

Но самое сложное — антисимметричность. В основе — утверждение, что если сумма двух чисел равна нулю, то каждое слагаемое равно нулю. Это утверждение устанавливается формулой 7, которую легко превратить в правила $\dfrac{\Gamma \vdash X+Y=0}{\Gamma \vdash X=0}$ и $\dfrac{\Gamma \vdash X+Y=0}{\Gamma \vdash Y=0}$.

Теперь выведем формулу $(X \leqslant Y) \wedge (Y \leqslant X) \to (X=Y)$. Эта задача сводится к доказательству выводимости $\exists u(X+u=Y), \exists u(Y+u=X) \vdash X=Y$. Вывод сводится к удалению кванторов существования (список Γ содержит формулы $X+u_1=Y$, и $Y+u_2=X$):

$$\exists u(X+u=Y), \exists u(Y+u=X) \vdash X=Y$$
$$X+u_1=Y,\ Y+u_2=X \vdash X=Y$$
$$\Gamma \vdash Y+u_2=Y \qquad \Gamma \vdash X=Y+u_2$$
$$\Gamma \vdash u_2=0 \qquad \Gamma \vdash Y+u_2=X$$
$$\Gamma \vdash u_1+u_2=0$$
$$\Gamma \vdash X+u_1+u_2=X$$
$$\Gamma \vdash X+u_1+u_2=Y+u_2 \qquad \Gamma \vdash Y+u_2=X$$
$$\Gamma \vdash X+u_1=Y$$

6. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ АЛГОРИТМОВ

6.1. Основные характеристики алгоритма

Интуитивное понятие алгоритма. Его отличительные черты: дискретность, детерминированность, элементарность шагов, направленность, массовость). Работа алгоритма как вычисление функции. Понятие вычислимой функции. Алгоритм как функционирование абстрактной машины.

Алгоритм — одно из фундаментальных понятий не только современной математики, но и всего естествознания. Развитие информационных технологий и группы наук, обеспечивающих эти технологии, привело к тому, что понятие „алгоритм“ стало одним из ключевых.

Фундаментальность этого понятия означает, что мы не можем рассчитывать на точное определение алгоритма: это означало бы, что есть более фундаментальные понятия. Но эта ситуация с понятием „алгоритм“ далеко не уникальна. Скажем, как определить, что такое энергия? Да, в теоретической механике, объектом исследований которой являются относительно простые физические объекты, можно точно сформулировать что такое энергия. В других физических дисциплинах это понятие приходится уточнять и расширять. В конечном счете оказывается, что энергия — нечто большее, нежели физический термин, это некоторая сторона нашего мироощущения.

То же самое можно сказать о понятии „алгоритм“. Мы можем выделить лишь некоторые отличающее это понятие черты. Какие же?

Можно сказать, что алгоритм — это точное предписание, задающее вычислительный процесс, который начинается с произвольного объекта некоторой совокупности и направлен на получение полностью определяемого исходным объектом результата.

Приведенное описание понятия можно уточнить, указав некоторые характерные черты алгоритма.

1. Дискретность: алгоритмический процесс расчленяется на отдельные шаги ограниченной сложности, каждый из которых состоит в непосредственной переработке полученных к этому моменту результатов.

2. Детерминированность: непосредственная обработка результатов на каждом шаге полностью определяется исходным объектом и результатами, полученными на предыдущих шагах*.

3. Направленность: алгоритмический процесс, примененный к исходному объекту, дает „решение задачи“ — заключительный объект. По-другому: на каждом шаге алгоритмического процесса известно, что является результатом.

4. Массовость: исходный объект может выбираться произвольно из некоторой бесконечной совокупности.

Описание алгоритма даже с учетом перечисления характерных черт очень расплывчатое. Впрочем это не мешает, когда речь идет о поиске решения задачи. Конкретное описание решения можно непосредственно проанализировать и сказать, является это описание алгоритмом или нет.

Приведем некоторые примеры.

*К этому иногда добавляют условие локальности, заключающееся в том, что на каждом шаге используются результаты только ограниченного количества предыдущих шагов. Однако в силу конечности всех рассматриваемых процессов условие конечности всегда можно считать выполненным. Это условие носит скорее методологический характер.

Пример 6.1. Самые первые примеры алгоритмов — алгоритмы выполнения арифметических операций над числами, записанными в десятичной (или какой-то иной) системе исчисления. Попутно отметим, что алгоритмы эти работают не с самими числами,а с их представлениями. Можно себе представить, как надо выполнять арифметические операции над числами, записанными римскими цифрами. Во всяком случае ясно, что это будут совсем другие алгоритмы, заметно отличающиеся от привычных нам.

Пример 6.2. Алгоритм Евклида вычисления наибольшего общего делителя — один из самых древних в истории математики. Чтобы найти наибольший общий делитель натуральных чисел p и q, делим p на q с остатком: $p = \alpha_1 q + \beta_1$. Затем второе число q делим первый остаток β_1 с остатком: $q = \alpha_2 beta_1 + \beta_2$. Затем делим β_1 на β_2 и т.д. Наступит момент, когда очередной остаток равен нулю: $\beta_{k-1} = \alpha_{k+1}\beta_k$. В этот момент процедура прекращается, а пользователю предъявляется решение: наибольший общий делитель — это число β_k.

Весь этот процесс можно представить себе как работу некоей машины-делителя. В начале мы вносим в нее исходные числа p и q и запускаем процесс. После остановки процесса извлекаем результат β_k. Машина будет работать при любых натуральных числах и всегда будет давать ответ.

Пример 6.3. Еще один известный алгоритм — алгоритм Гаусса решения линейных систем. Ограничимся крамеровскими системами, хотя это и не принципиально. Отметим, что этот алгоритм в некоторых ситуациях может приводить к делению на нуль и тем самым не давать решения, хотя решение существует (вопрос, для каких систем алгоритм Гаусса не работает?). Модификация алгоритма Гаусса с выбором главного элемента дает решение для любой крамеровской системы. Впрочем, последнее утверждение тоже следует считать условным: при работе с числами, имеющими фиксированное число знаков, алгоритм может аварийно останавливаться в двух случаях: переполнение и деление на нуль.

Вообще говоря, в качестве объектов, которые перерабатываются алгоритмом, могут выступать в принципе любые объекты. Однако отметим два обстоятельства.

Во-первых, все объекты, которые изучает математика могут быть представлены в числовой форме (в виде чисел и совокупностей чисел), причем все в конечном счете сводится к натуральным числам. Так, функция действительного переменного — это некоторое подмножество в $\mathbb{R}^2$ (то, что мы называем графиком функции), т.е. некоторое множество пар действительных чисел. В свою очередь, каждое действительное число можно представить в виде бесконечной десятичной дроби. Метод координат позволяет все геометрические объекты перевести в числовую форму.

Во-вторых, используемые на практике алгоритмы работают не с самими объектами, а с их представлениями в том или ином языке. Все наши процессы обработки сводятся к переработке групп чисел с фиксированным числом разрядов (целых чисел), записанных в той или иной системе исчисления. Поэтому алгоритм можно интерпретировать как функцию нескольких переменных, аргументы и значения которой являются натуральными числами. Отметим, что не всякую целочисленную функцию можно связать с алгоритмом, поскольку алгоритм — это конечное предписание, а значит, их счетное число, а целочисленных функций — континуум. Значит, надо выделять специальный класс целочисленных функций, значение которых можно получить за конечное число элементарных действий. В литературе такие функции называют ***эффективно вычислимыми***. Это понятие, как и алгоритм, относится к интуитивным, первичным. Фактически, эффективно вычислимая функция — это алгоритм, но как бы с другой точки зрения.

Один из подходов в теории алгоритмов — представление алгоритма как теории эффективно вычислимых функций. Другой подход — представление алгоритма как процесса функционирования некоторой абстрактной машины. Исходный объект, представленный группой натуральных чисел, интерпретируется как исходной состояние абстрактной машины. В каждый момент времени (которое является дискретным) абстрактная машина из известного состояния пере-

ходит в другое состояние, строго определенное исходным состоянием. Некоторые состояния являются конечными. Достигнув такого состояния, машина „выплевывает“ результат.

Оба этих подхода являются ограничением исходного интуитивного понятия „алгоритм“. Есть несколько целей такого ограничения. Во-первых, общее понятие алгоритма слишком обширно, чтобы о нем можно было сказать что-то содержательное. (Аналогично что можно сказать о функции действительного переменного вообще? Чтобы получить какую-то теорию, мы ограничиваем этот слишком широкий класс функций, вводя понятие непрерывности, дифференцируемости и т.п.) Во-вторых, теория алгоритмов начиналась с решения вполне конкретных задач, в первую очередь задач математической логики и оснований математики. Для таких целей общее понятие алгоритма не нужно. Наконец, в-третьих, для решения проблем разрешимости формальных теорий, когда требуется установить отсутствие алгоритмов, решающих задачу, необходимо математически точное определение алгоритма, а это без ограничения понятия невозможно.

Исторически сложились три подхода к формальному определению алгоритма:

1) машины Тьюринга — Поста, предложенные Тьюрингом и независимо от него Постом в 1936 г.;

2) частично рекурсивные функции, предложенные Клини в 1936–37 гг.;

3) нормальные алгорифмы Маркова (1951 г.).

Все эти подходы объединяют общим понятием „вычислительная модель“. Указанные три вычислительные модели обладают важным свойством представительности. Это свойство заключается в том, что любой алгоритм со счетным множеством исходов X и со счетным множеством результатов Y может быть реализован в рамках данной вычислительной модели. Утверждение такого сорта не может быть доказано математическими средствами, поскольку общее понятие алгоритма выходит за рамки математики. Его истинность может быть лишь подтверждена практикой. Оно известно как ***тезис Черча***, который впервые был сформулирован для рекурсивных функций в форме: „любая эффективно вычислимая функция является частично рекурсивной“. Однако отметим, что эквивалентность вычислительных моделей может быть доказана математически строго.

В связи с понятием „алгоритм“ вспомним другое понятие „исчисление“, под которым в рамках математической логики имелось в виду логическое исчисление (т.е. исчисление математической логики). В общем исчисление представляет собой некий алфавит и набор правил, каждое из которых позволяет заменять то или иное слово другим. Процесс применения правил — это вывод. Вывод начинается с применения одного из правил с пустой посылкой, которые обычно формулируют в виде аксиом. Это действительно похоже на применение алгоритма, но отличается от алгоритма следующим. Если алгоритм носит повелительный характер (из данного слова мы можем перейти к другому только одним способом — детерминированность алгоритма), то исчисление носит разрешительный характер (для преобразования данного слова мы можем применять любое из возможных правил). Можно сказать, что алгоритмический процесс — это линейный вывод, в то время как вывод в исчислении носит древовидный характер. Связь между алгоритмами и исчислениями можно уловить из дальнейшего изложения.

6.2. Конструктивные объекты

Объекты с которыми работает алгоритм. Конструктивные объекты. Алфавит. Слова. Конкатенация. Подслова и вхождения.

Как отмечено, на объекты, которые перерабатываются алгоритмами, накладываются серьезные ограничения. В связи с этим возникает понятие ***конструктивный объект***, под которым понимается такой объект, который создан из конечного числа простейших объектов и в котором эти простейшие составляющие могут быть легко локализованы. Обычно считают, что простейших объектов конечное множество. Таким образом, конструктивный объект —

это конечный объект, т.е. представляет собой конечное множество. Однако не всякое конечное множество конструктивно. Требуется, чтобы по объекту можно было понять и историю его создания из простейших.

Например, само по себе натуральное число, как мера количества, не является конструктивным объектом. Однако его легко превратить в конструктивный, если рассматривать десятичную запись натурального числа. При этом простейшими объектами являются десятичные цифры. На самом деле можно ограничиться лишь одним символом: I —1, II — 2, III — 3 и т.д.

Как видим понятие конструктивного объекта легко укладывается в терминологию „буква“ — „алфавит“ — „слово“. При этом алфавит —- произвольное конечное множество, элементы которого мы называем буквами, слово — произвольный упорядоченный набор букв (***кортеж***).

Разумеется, в качестве букв, составляющих алфавит, мы будем выбирать не отвлеченные объекты (например, автомобили в г. Москва), а привычные языковые символы (буквы латинского, греческого алфавитов, кириллицы и др.). Порядок в конечном множестве можно рассматривать как установление взаимно-однозначного (биективного) соответствия между элементами множества и некоторым отрезком натурального ряда. Однако в данном случае проще считать понятия „меньше“ — „больше“ синонимами понятий „левее“ —„правее“ при записи кортежа. Единственное здесь уточнение: мы не будем использовать для записи кортежа разделителей, поскольку объекты, составляющие кортеж, неделимы.

Напомним, что существует формальный объект, называемый пустым кортежем, который есть пустое множество букв алфавита. Такое множество можно считать упорядоченным априори. Пустой кортеж (пустое слово) будем обозначать символом Λ.

Пример 6.4. Алфавит, содержащий единственную букву, например I, позволяет представить слова Λ, I, II, III, IIII, ... Эти слова можно отождествить с натуральными числами (при этом пустое слово будет соответствовать числу 0, которое в математической логике и теории алгоритмов считают натуральным). Введение дополнительного символа $-$ позволяет записывать и другие целые числа: $-$I, $-$II. Есть, правда и другие слова, например II$-$III, которые можно рассматривать как запись разности двух натуральных чисел. Но есть и такое слово: I$-$II$-$III. Как его рассматривать? Ему также можно придать смысл, установив порядок выполнения операций. Но есть и другой вариант действий: объявить, что не все слова являются правильными. В принципе это естественно: те слова, которые у нас будут возникать, будут порождаться определенными правилами. Некоторые слова, которые вообще-то состоят из букв данного алфавита, не могут появиться, поскольку для них нет порождающих правил. Если считать, что мы можем добавить к слову символ $-$ только если это слово пустое, а символ I можем добавлять только справа, то получим только „правильные“ записи целых чисел.

Пример 6.5. Терминология „буква“ — „алфавит“ — „слово“ уже встречалась в курсе математической логики. Однако там под буквой понималась переменная, а алфавит автоматически считался счетным множеством. На самом деле требование счетности с точки зрения конструктивности не обременительно. Формальную теорию можно было строить и с конечным алфавитом. Например, рассмотрим алфавит с буквами V, C, F, P, I, $\#$. Эти буквы позволяют порождать слова вида $VIII\#$, образующие уже новый счетный алфавит. Здесь первая буква определяет тип символа (в данном случае V — переменная), последующие символы — порядковый номер (в данном случае три), а последняя буква — признак конца этого микрослова. Эти микрослова можно соединять в любом порядке, но при этом всегда сохраняется возможность выделять эти образования в большом слове. В логико-математическом языке это дополнительное построение не вносит ничего нового, но усложняет теоретические построения.

Слова в данном алфавите A можно соединять. Если X и Y — слова, то под XY (или $X \cdot Y$ будем понимать слово полученное присоединением к слову X справа слова Y. Например, если $A = \{a, b\}$, $X = abb$, $Y = baba$, то $XY = abbbaba$. Такая операция часто называется конкатенацией (соединением). Мы используем для нее мультипликативную форму записи операции

(т.е. трактуем как умножение). Отсюда понятны обозначения $Y^0 = \Lambda$ (поскольку нейтральным элементом по конкатенации является пустое слово), $Y^k = Y^{k-1}Y$, $k = 1, 2, \ldots$

Отметим и другую терминологию, связанную с понятиями „алфавит“, „слово“ и уже встречавшуюся в курсе математической логики. Количество символов в слове X называется ***длиной слова*** X. Так, для алфавита A множество A^r представляет собой совокупность всех слов длины r. Объединение $\bigcup\limits_{r\geqslant 0} A^r$, обозначаемое как A^*, есть множество всех слов в алфавите A.

Если слово Y представимо в виде $Y = ZXW$, где Z и W могут быть и пустыми, то слово X называется ***подсловом*** слова Y. В частности, каждое слово является подсловом самого себя, а пустое слово является подсловом любого другого. У слова Y может быть несколько представлений вида $Y = Z_1XW_1 = Z_2XW_2 = \ldots$ Эти представления можно упорядочить по возрастанию длин подслов Z_1, Z_2 и т.д. При этом говорят о первом ***вхождении***, втором и т.п. слова X В слово Y. Так, первое вхождение пустого подслова — в самом начале слова.

Также говорят „слово X входит в слово Y“, имея ввиду, что X есть подслово слова Y. Если в представлении $Y = ZXW$ подслово Z пустое, т.е. на самом деле $Y = XW$, то подслово X называется ***началом слова*** Y. Аналогично вводится понятие ***конца слова***. Подслово (начало слова, конец) называется собственным, если оно не совпадает со всем словом и не является пустым.

Замечание 6.1. Слова — не единственный конструктивный объект, который можно построить с помощью данного алфавита. Другим простейшим конструктивным объектом являются деревья. Пусть задан алфавит A. Рассмотрим ориентированное дерево, вершины которого помечены буквами алфавита A, а дуги, исходящие из одной вершины, упорядочены, например маркировкой натуральными числами. Такое дерево назовем A-деревом. Максимальная степень исхода k по вершинам дерева может служить мерой сложности этого дерева. (A, k)-деревом называют любое A-дерево, максимальная степень исхода которого не превышает k.

Слова над алфавитом A можно рассматривать как $(A, 1)$-деревья. В то же время любое (A, k)-дерево можно по определенным правилам преобразовать в слово. Так что с теоретической точки зрения новые конструктивные объекты не дают ничего нового. #

Итак, объектами, над которыми работают алгоритмы, являются слова в некотором алфавите A. Результат работы алгоритма можно представить как своего рода правило, которое каждому слову из некоторого множества $X \subset A^*$ ставит в соответствие другое слово. Значит, с каждым алгоритмом можно связать отображение $\varphi\colon X \to A^*$, называемое ***словарной функцией***. Такая функция называется ***частичной***, если $X \neq A^*$, и ***полностью определенной***, если $X = A^*$.

Вообще говоря, естественно пробовать применять алгоритм (как отображение) к любому слову в данном алфавите. Тогда возможны три сценария завершения работы алгоритма. В стандартном сценарии мы получаем новое слово как результат работы алгоритма. Во втором сценарии мы на очередном шаге можем получить слово, к которому элементарные операции, используемые в алгоритме, не применимы. В этой ситуации работа алгоритма прекращается безрезультатно (например, деление на нуль в ряде численных алгоритмов). Мы можем модифицировать алгоритм, считая, что в данной ситуации он прекращает работу, а его результатом является некое заранее оговоренное слово (например ERROR). Наконец, возможен третий сценарий, когда алгоритм „зацикливается“. Простейший вариант такой ситуации — повторение результатов, получаемых на каждом шаге. Но такое повторение не обязательно.

Исходя из изложенного можно сказать, что алгоритм — это многократное применение некоторой словарной функции, начинающееся с некоторого исходного слова. Этот процесс останавливается, если на очередном шаге выполнено условие останова или слово, полученное на предыдущем шаге, вышло за пределы области определения словарной функции. Мы видим, что исследование алгоритмов по существу есть исследование словарных функций и решение задач представимости одних словарных функций с помощью других.

7. НОРМАЛЬНЫЕ АЛГОРИФМЫ МАРКОВА

7.1. Определение

Алфавит и подстановки. Схемы. Нормальный алгорифм Маркова. Терминология. Примеры НА. Эквивалентные НА. Принцип нормализации.

Мы считаем, что задан некоторый конечный алфавит, и рассматриваем разные способы преобразования слов в этом алфавите. В качестве базовой операции рассмотрим так называемую ***контекстную замену***. При этой операции определенное подслово данного слова заменяется на другое слово. Иными словами, если исходное слово имеет вид $X = Y * W * Z$, то контекстная замена $W \to V$ приводит к слову $X' = Y * V * Z$.

Существует несколько вариантов контекстной замены, которые различаются по своему действию при нескольких вхождениях подслова и наличием специальных подстановочных знаков. Мы остановимся на простейшей операции, при которой заменяется только первое вхождение данного слова.

Нормальный алгорифм Маркова[*] определяется некоторым непустым конечным упорядоченным набором формул вида $W \to V$ в заданном алфавите A, где W и V — произвольные слова в A (одно из двух слов может быть пустым). Каждая такая формула относится к одной из двух категорий: первая категория — простые формулы, вторая — терминальные (заключительные). Терминальные формулы выделяют, ставя обычно точку после стрелки, например $abba \to\cdot\ ab$. Здесь предполагается, что символы $\to$ и $\cdot$ не входят в алфавит A. Указанный упорядоченный набор формул называют ***схемой нормального алгорифма*** в алфавите A.

Действие нормального алгорифма состоит в пошаговом преобразовании исходного слова с помощью контекстной замены. Формально это действие на одном шаге описывается следующим образом. Пусть X — текущее слово. Последовательно просматривается список формул в схеме алгоритма начиная с первой. Для очередной формулы вида $W \to * V$, где символ $*$ обозначает или пустое слово, или точку, ищется первое вхождение слова W в слове X. Если такое вхождение найдено, выполняется подстановка вместо этого вхождения слова V. При этом, если формула подстановки терминальная, работа алгоритма заканчивается. Если вхождений нет, переходим к следующей формуле схемы. Если все формулы схемы проанализированы, но вхождений не найдено, алгоритм заканчивает работу.

Введем некоторые обозначения. Алгоритмы будем обозначать готическими буквами: $\mathfrak{A}$, $\mathfrak{B}$ и т.д. Если слово X преобразуется в слово Y с помощью простой подстановки, будем писать $\mathfrak{A}$: $X \vdash Y$ и говорить, что алгоритм $\mathfrak{A}$ переводит X в Y. В аналогичной ситуации при терминальной подстановке пишем $\mathfrak{A}$: $X \vdash\cdot\ Y$ и говорим, что алгоритм $\mathfrak{A}$ терминально переводит X в Y. Если в слове X не может быть выполнена ни одна подстановка схемы, будем писать $\mathfrak{A}$: $X\dashv$ и говорить, что алгоритм $\mathfrak{A}$ не переводит слово X.

Предположим, что в результате работы алгоритма $\mathfrak{A}$ мы получили последовательность слов $X = X_0,\ X_1,\ \ldots,\ X_k = Y$, где $\mathfrak{A}$: $X_{j-1} \vdash X_j$, $j = \overline{0,\ k-1}$. Тогда будем обозначать $\mathfrak{A}$: $X \vDash Y$ и говорить, что алгоритм $\mathfrak{A}$ преобразует X в Y. Если на последнем шаге $\mathfrak{A}$: $X_{k-1} \vdash\cdot\ X_k$, т.е. последнее слово в последовательности получено применением терминальной подстановки, то будем писать $\mathfrak{A}$: $X \vDash\cdot\ Y$ и говорить, что алгоритм $\mathfrak{A}$ преобразует X в Y терминально. Короче: „переводит“ за один шаг, „преобразует“ за конечное число шагов, „терминально“ —

[*]Вообще говоря, в русском языке используют написания „алгоритм“ и „алгорифм“ как равноценные. Словосочетание „нормальный алгорифм Маркова“ используется как установившийся термин.

последняя подстановка (возможно, единственная) терминальная. По определению считаем, что для любого слова $\mathfrak{A}$: $X \vDash X$.

С каждым нормальным алгорифмом связывается словарная функция, которую мы будем обозначать так же, как и сам алгоритм. Для произвольного слова $X \in A^*$ применение нормального алгорифма приводит к одному из трех результатов:

1) на некотором k-м шаге мы получаем слово X_k, для которого $\mathfrak{A}$: $X_{k-1} \vdash\cdot\ X_k$;

2) на некотором k-м шаге мы получаем слово X_k, для которого $\mathfrak{A}$: $X_k\dashv$;

3) алгоритм продолжает работу неограниченное число шагов.

В первых двух случаях считаем слово X_k результатом работы алгоритма и записываем $X_k = \mathfrak{A}(X)$ (как значение словарной функции). При этом также говорим, что алгоритм $\mathfrak{A}$ применим к слову X и пишем $!\mathfrak{A}(X)$. В третьем случае говорим, что алгоритм $\mathfrak{A}$ неприменим к слову X и пишем $\neg!\mathfrak{A}(X)$.

Пример 7.1. Рассмотрим некоторый алфавит A и схему из одной подстановки $\to\cdot\ P$, где P — некоторое фиксированное слово. Соответствующий алгоритм $\mathfrak{A}_P$ применим к любому слову и за один шаг приписывает к любому слову слово P, т.е. $\mathfrak{A}_P(X) = P * X$.

Отметим, что подстановка с пустой левой частью применима всегда и первое вхождение пустой строки — перед первым символом слова. Если в схему алгоритма добавить какие-то другие подстановки, они никогда не будут использованы. Поэтому подстановки с пустой левой частью уместны в конце схемы.

Пример 7.2. Рассмотрим нормальный алгорифм со схемой из одной формулы $\to P$. Такой алгорифм неприменим ни к какому слову, поскольку на каждом шаге подстановка срабатывает и приводит к вставке в начало слова фрагмента P, т.е. мы получаем неограниченную последовательность X, PX, PPX, ...

Отметим, что отсутствие в схеме терминальных подстановок еще не означает, что алгоритм неприменим к любому слову. Может образоваться последовательность слов, завершающаяся непереводимым словом. Это тоже считается нормальным завершением работы алгоритма.

Как добавить фрагмент в конец слова? Это более сложная задача, чем добавление фрагмента в начало слова. Условимся о следующем. В формулах подстановки будем использовать специальные символы, не входящие в алфавит (для этих символов будем использовать греческий алфавит). На содержательном уровне эти символы играют ту же роль, что и символ „?“ в имени файла, т.е. в левой части такой символ обозначает произвольный символ алфавита (но не пустое слово), а в правой части — тот символ алфавита, на который указывает этот же символ в левой части. С формальной стороны использование такого символа, например ξ, означает, что для каждого символа алфавита в формуле подстановки ξ заменяется на этот символ, давая тем самым конкретную формулу подстановки. Иначе говоря, если в формуле есть специальный символ, то это не формула подстановки, а схема формул. Каждая схема формул дает конечный набор конкретных формул, который и подразумевается взамен схемы формул. Например, пусть дан алфавит $A = \{a, b, c\}$. Тогда запись $a\xi \to \xi a$ обозначает три формулы подстановки: $aa \to aa$, $ab \to ba$, $ac \to ca$.

Чтобы фиксировать порядок формул подстановки в схеме алгоритма при замене схем формул условимся, что алфавит считается упорядоченным. Тогда в каждой схеме формул выбираем самый первый (левый) специальный символ и, последовательно его заменяя в схеме символами алфавита от первого до последнего, получим набор формул, который и заменяет исходную схему формул. Если в новых формулах остаются специальные символы, процедуру повторяем.

Пример 7.3. По-видимому, можно строго показать, что ни один нормальный алгорифм в алфавите A не реализует вставку в конец любого слова фрагмента P. В чем проблема? На последнем шаге должна произойти замена какого-то подслова в конце слова фрагментом P. Но не факт, что это подслово не имеет других вхождений. Возможная стратегия такова. Нужна

какая-то комбинация символов, которая гарантированно не встретится при работе алгоритма. Пусть это будет слово S. Тогда можно составить такую схему:

$$(S\xi \to \xi S,\ S \to\cdot\ P,\ \ \to S).$$

Предполагая, что в исходном слове подслово S не встретится, заключаем, что на первом шаге применяется 3-я подстановка, которая в начало слова вставляет фрагмент S. Далее работает первая схема подстановок, которая любую букву справа от S переставляет в позицию перед S. Первая схема будет пропущена, когда S окажется в конце слова. Тогда сработает вторая подстановка, которая заменит S на P и остановит работу.

Еще одно решение — расширение алфавита A добавлением к нему еще одной буквы $\overline{a} \notin A$. Тогда схема

$$(\overline{a}\xi \to \xi\overline{a},\ \overline{a} \to\cdot\ P,\ \ \to \overline{a})$$

даст нормальный алгорифм, который к любом слову в алфавите A добавит в конце фрагмент P.

В связи с последним примером введем некоторые понятия. Если алфавит B как часть содержит алфавит A, то говорим, что B есть расширение A. Запись этого обычная: $A \subset B$. Формально считаем, что каждый алфавит есть расширение самого себя (т.е. подразумевается нестрогое включение). Мы говорим, что $\mathfrak{A}$ — алгоритм над алфавитом A, если $\mathfrak{A}$ — алгоритм в некотором расширении B алфавита A. Формально алгоритм в алфавите A является алгоритмом над A.

Как связаны алгоритм и алфавит? В схеме алгоритма есть некоторый набор символов. Если алфавит содержит все эти символы (но может содержать и еще какие-то символы), то алгоритм в этом алфавите. Если часть символов схемы не входит в алфавит (такие символы можно назвать служебными), то это алгоритм над алфавитом. Как видим жесткой связи между двумя понятиями нет. В принципе мы можем взять любую схему и рассматривать ее как алгоритм над любым алфавитом. Однако если эта парочка несогласована, не следует думать, что алгоритм будет работать так, как мы задумали.

Пусть A — некоторый алфавит. Назовем алгоритмы $\mathfrak{A}$, $\mathfrak{B}$ над алфавитом A ***эквивалентными относительно*** A, если для любого слова $X \in A^*$ выполнены условия:

- если $!\mathfrak{A}(X)$ и $\mathfrak{A}(X) \in A^*$, то $!\mathfrak{B}(X)$ и $\mathfrak{B}(X) = \mathfrak{A}(X)$;
- если $!\mathfrak{B}(X)$ и $\mathfrak{B}(X) \in A^*$, то $!\mathfrak{A}(X)$ и $\mathfrak{A}(X) = \mathfrak{B}(X)$.

Приведенное определение равносильно тому, что два алфавита $\mathfrak{A}$ и $\mathfrak{B}$ либо оба неприменимы к слову $X \in A^*$, либо оба применимы, причем два результата либо равны, либо выходят за пределы алфавита A.

Алгоритмы $\mathfrak{A}$, $\mathfrak{B}$ над алфавитом A назовем ***вполне эквивалентными относительно*** A, если для любого слова $X \in A^*$ либо оба неприменимы к X, либо оба дают одинаковый результат, т.е. либо $\neg!\mathfrak{A}(X) \wedge \neg!\mathfrak{B}(X)$, либо $!\mathfrak{A}(X) \wedge !\mathfrak{B}(X) \wedge (\mathfrak{A}(X) = \mathfrak{B}(X))$.

В дальнейшем для любых алгоритмов $\mathfrak{A}$ и $\mathfrak{B}$ и любого слова X будем использовать запись $\mathfrak{A}(X) \simeq \mathfrak{B}(X)$, если либо оба алгоритма неприменимы к слову X, либо оба они применимы и дают одинаковый результат. Выражение $\mathfrak{A}(X) \simeq \mathfrak{B}(X)$ будем называть ***условным равенством***. С помощью этого понятия можно сказать, что алгоритмы $\mathfrak{A}$ и $\mathfrak{B}$ вполне эквивалентны относительно алфавита A, если $\mathfrak{A}(X) \simeq \mathfrak{B}(X)$ для любого $X \in A^*$. В этом случае часто пишут $\mathfrak{A} \simeq \mathfrak{B}$.

Принцип нормализации. Всякий алгоритм в алфавите A вполне эквивалентен относительно A некоторому нормальному алгорифму над алфавитом A.

Этот принцип не является строгим математическим утверждением, поскольку здесь есть не определенное с математической точки зрения понятие: алгоритм в алфавите A. Здесь имеется ввиду, некая механическая процедура, которая для любого слова в алфавите A дает результат — слово в том же алфавите или оказывается неприменимым к этому слову. Но и такое пояснение не есть строгое математическое определение.

7.2. Теорема о переводе и теорема приведения

Постановка задачи о переводе. Лемма о взаимно однозначном соответствии. Теорема о переводе. Теорема о приведении (к двухбуквенному алфавиту).

Введенное понятие эквивалентности алгоритмов по существу означает, что эти алгоритмы дают одинаковые результаты. Предполагается, что эти алгоритмы действуют в одном алфавите. Однако ясно, что один и тот же алгоритм (в содержательном смысле) можно реализовывать в разных алфавитах (подобно тому, что одни и те же знания можно описать разными языками). Как осуществлять перенос алгоритма в другой алфавит? Отметим, что, каков бы ни был по существу перевод алгоритма из одного алфавита в другой, этот перевод — тоже алгоритм.

Пусть задан алфавит A и его некоторое расширение $C = A \cup \{t_1, t_2, \ldots, t_n\}$. Выберем еще две буквы p и q, не входящие в C, и введем еще одно расширение алфавита A — алфавит $B = A \cup \{p, q\}$. Поставим задачу о переносе алгоритмов в алфавите C на алфавит B. Такой перенос может строиться как перенос соответствующих словарных функций с одного алфавита на другой. А для этого необходимо установить соответствие между словами в двух алфавитах.

Если буква $x \in C$ есть буква алфавита A, то ее переводом в алфавит B будет она сама, т.е. $x^\tau = x$, если $x \in A$. Если $x = t_i$, то полагаем $x^\tau = pq^ip$ (степень — в смысле операции конкатенации). Наш перевод должен сохранять операцию конкатенации. Значит, $(XY)^\tau = X^\tau Y^\tau$, в частности, если $X = x_1x_2\ldots x_k$, то $X^\tau = x_1^\tau x_2^\tau \ldots x_k^\tau$. Последняя формула — это определение отображения $\tau\colon C^* \to A^*$. Довольно очевидно, что это отображение сохраняет операцию, т.е. является гомоморфизмом одного моноида в другой. Также очевидны следующие свойства этого отображения.

Теорема 7.1. Пусть A — произвольный алфавит, B и C — его расширения, введенные выше. Тогда отображение τ обладает следующими свойствами:

1) отображение τ — мономорфизм, т.е. из равенства $X^\tau = Y^\tau$ следует равенство $X = Y$;
2) равенство $X^\tau = X$ верно тогда и только тогда, когда $X \in A^*$.

◀ Второе свойство очевидно. Действительно, если $X \in A^*$, то его переводом, согласно определению является оно само, поскольку перевод осуществляется побуквенно. Если $X \notin A^*$, то в X есть одна из букв $t_1, \ldots, t_n$. Значит в слове X^τ появляются буквы p и q, т.е. $X^\tau \notin A^*$.

Рассмотрим первое свойство. Пусть $X \neq Y$. Обозначим $X = x_1x_2\ldots x_m$, $Y = y_1y_2\ldots y_l$. Найдется такой индекс i, что $x_1 = y_1, \ \ldots, x_{i-1} = y_{i-1}$, но $x_i \neq y_i$. Тогда X^τ имеет вид $x_1^\tau \ldots x_{i-1}^\tau x_i^\tau \ldots$, а Y^τ — вид $y_1^\tau \ldots y_{i-1}^\tau y_i^\tau \ldots$. В данном случае $x_1^\tau = y_1^\tau, \ldots, x_{i-1}^\tau = y_{i-1}^\tau$. Однако $x_i^\tau \neq y_i^\tau$. Поэтому $X^\tau \neq Y^\tau$. ▶

Мономорфизм τ, согласно доказанной теореме, устанавливает взаимно однозначное соответствие между множеством C^* всех слов в алфавите C и некоторым множеством слов в алфавите B. Это соответствие позволяет реализовать любую словарную функцию в алфавите C как словарную функцию в алфавите B.

Для нормального алгорифма $\mathfrak{A}$ в алфавите C со схемой $(P_1 \to * Q_1,\ P_2 \to * Q_2,\ P_k \to * Q_k)$ рассмотрим нормальный алгорифм $\mathfrak{A}^\tau$ в алфавите B со схемой $(P_1^\tau \to * Q_1^\tau,\ P_2^\tau \to * Q_2^\tau,\ P_k^\tau \to * Q_k^\tau)$. Этот алгоритм будем называть переводом нормального алгорифма $\mathfrak{A}$ из алфавита C в алфавит B.

Теорема 7.2 (о переводе алгоритма). Пусть A — произвольный алфавит, B и C — построенные выше его расширения, $\mathfrak{A}$ — нормальный алгорифм в алфавите C. Тогда для любого слова $X \in C^*$ выполнено равенство $\mathfrak{A}^\tau(X^\tau) \simeq \mathfrak{A}(X)^\tau$. При этом $\mathfrak{A}^\tau(X) \simeq \mathfrak{A}(X)^\tau$, если $X \in A^*$, и $\mathfrak{A}^\tau(X) \simeq \mathfrak{A}(X)$, если $X \in A^*$ и $\mathfrak{A}(X) \in A^*$.

◀ Следует проверить пошаговую работу алгоритма: если утверждение теоремы верно для любого алгоритма, то оно верно и для алгоритма, в котором все подстановки имеют статус терминальных. Такой алгоритм выполняет в точности то же, что и заданный, но останавливается после первого шага. Для этого достаточно убедиться, что произвольная подстановка

$P \to Q$ применима к слову $X \in C^*$ тогда и только тогда, когда подстановка $P^\tau \to Q^\tau$ применима к слову X^τ, причем результат второй есть перевод результата первой.

Если $X = V_1PV_2$, то по свойствам операции перевода $X^\tau = V_1^\tau P^\tau V_2^\tau$, и мы видим, что подстановка $P^\tau \to Q^\tau$ применима к слову X^τ, причем

$$\mathfrak{B}^\tau(X^\tau) = V_1^\tau Q^\tau V_2^\tau = (V_1QV_2)^\tau = \mathfrak{B}(X)^\tau$$

(здесь $\mathfrak{B}$ — алгоритм со схемой из единственной подстановки $P \to\cdot\, Q$).

Покажем, что каждому вхождению фрагмента P^τ в слово X^τ соответствует вхождение фрагмента P в слово X. Достаточно рассмотреть случай, когда P — это одна буква. Пусть $Y = X^\tau = y_1y_2\ldots y_m$ и вхождение фрагмента P^τ начинается с буквы y_j. Если $P \in A$, то $P^\tau = P$, это единственная буква $y_j \in A$. Эта буква не входит в слова $t_i^\gamma = pq^ip$. Значит, в слове X букве y_j соответствует та же буква. Если $P = t_i$, то $y_j = p$ и не является буквой алфавита A. При этом $y_{j+1} = q$, а буквенная пара pq в слове Y обозначает начало перевода x_l^τ некоторой буквы x_l слова X. Аналогично слово P^τ заканчивается парой qp, что соответствует концу перевода в слове X^τ. Наконец, фрагмент Y^τ содержит всего лишь две буквы p, а значит в слове X^τ может соответствовать переводу только одной буквы. Тем самым показано, что если существует вхождение слова P^τ в слово X^τ, то буква P входит в слово X. Отсюда легко сделать вывод, что для произвольного слова P из существования вхождения P^τ в X^τ следует существование вхождения P в X. ►

Теорема 7.3 (о приведении к двухбуквенному алфавиту). Пусть A — произвольный алфавит, p и q — две разные буквы, не входящие в A. Тогда любой нормальный алгорифм над A, не содержащий в схеме букв p и q, эквивалентен относительно A некоторому нормальному алгорифму в алфавите $A \cup \{p,\ q\}$.

◄ Пусть $\mathfrak{A}$ — нормальный алгорифм над алфавитом A и $t_1,\ t_2,\ \ldots,\ t_n$ — множество всех букв, использованных в схеме $\mathfrak{A}$, но не входящих в алфавит $B = A \cup \{p,\ q\}$. Тогда $\mathfrak{A}$ — нормальный алгорифм в алфавите $C = A \cup \{t_1,\ t_2,\ \ldots,\ t_n\}$. Построим перевод τ из алфавита C в алфавит B. Согласно теореме 7.2, имеем $\mathfrak{A}^\tau(X^\tau) \simeq \mathfrak{A}(X)^\tau$, причем, если $X \in A$, $\mathfrak{A}(X) \in A$, то $\mathfrak{A}^\tau(X) \simeq \mathfrak{A}(X)$. Это означает, что алгоритмы $\mathfrak{A}$ и $\mathfrak{A}^\tau$ эквивалентны. При этом $\mathfrak{A}^\tau$ — алгоритм в алфавите B. ►

7.3. Операции над нормальными алгорифмами

Расширение НА. Замыкание. Композиция. Соединение. Разветвление. Повторение.

Над алфавитами можно выполнять различные операции. Комбинируя определенным образом алгоритмы, можно получать новые алгоритмы. В некоторых ситуациях такое комбинирование не затрагивает сути комбинируемых алгоритмов. В таких ситуациях возникает операция над алгоритмами. Поскольку для нас важна не формальная запись алгоритма, а выполняемое им действие, операции над алгоритмами могут выполняться с точностью до эквивалентности, т.е. эквивалентные алгоритмы мы не различаем.

Расширение алфавита. Как было отмечено, нормальный алгорифм слабо связан с алфавитом, над которым он действует. Одна и та же схема порождает алгоритмы над разными алфавитами. Различие между такими алгоритмами — лишь в области применимости.

Рассмотрим два варианта расширения нормального алгорифма. Пусть $\mathfrak{A}$ — нормальный алгорифм над алфавитом A, заданный схемой $T = \{P_1 \to* Q_1,\ \ldots,\ P_n \to* Q_n\}$. Алгоритм $\mathfrak{A}^e$, который над алфавитом $B \supset A$ задан в точности той же схемой T, называется ***естественным расширением*** алгоритма $\mathfrak{A}$. Работа этого алгоритма совпадает с работой исходного алгоритма, если исходное слово берется в алфавите A. Однако фактическая область применимости при

этом расширяется, причем действие алфавита будет зависеть от того, входят ли новые буквы (т.е. из $B \setminus A$) в схему или нет.

При втором варианте расширения алфавита мы также сохраняем эквивалентность алгоритма относительно A, но сохраняем также и область применимости. Пусть $\mathfrak{A}$ — нормальный алгорифм над алфавитом A, заданный схемой $T = \{P_1 \to * Q_1,\ \ldots,\ P_n \to * Q_n\}$. Алгоритм $\mathfrak{A}^f$ над $B \supset A$ со схемой

$$\begin{cases} \xi \to \xi, \quad \xi \in B \setminus A, \\ P_1 \to * Q_1, \\ P_2 \to * Q_2, \\ \quad \ldots\ldots\ldots\ldots \\ P_n \to * Q_n \end{cases}$$

называется ***формальным расширением*** алгоритма $\mathfrak{A}$. Если в схему не входят новые буквы (из множества $B \setminus A$), в частности, если $\mathfrak{A}$ — алгоритм в алфавите A, то добавление новых подстановок не отразится на работе алгоритма, начинающейся со слова в алфавите A, т.е. новый алгоритм вполне эквивалентен исходному. Однако алгоритм $\mathfrak{A}^f$ неприменим к словам, содержащим хотя бы одну новую букву. Другими словами, формальное расширение алгоритма сохраняет область применимости алгоритма.

Пример 7.4. Рассмотрим алгоритм $\mathfrak{A}$ над алфавитом A со схемой

$$\begin{cases} *\xi \to \xi *, \\ * \to \cdot\ P, \\ \Lambda \to *, \end{cases}$$

где символ $*$ не входит в A. Этот алгоритм добавляет в конец слова из алфавита A слово P (которое можем считать словом в алфавите A). Рассмотрим алфавит $B = A \cup \{*\}$ и формальное расширение $\mathfrak{A}^f$ алгоритма $\mathfrak{A}$, расширив его схему:

$$\begin{cases} * \to *, \\ *\xi \to \xi *, \\ * \to \cdot\ P, \\ \Lambda \to *. \end{cases}$$

Нетрудно увидеть, что этот алгоритм неприменим ни к одному слову в алфавите A: на первом шаге в начало слова вставляется символ $*$, а далее работает только первая подстановка, приводящая к зацикливанию.

Мы видим, что наличие в схеме одной из новых букв может привести к алгоритму, не эквивалентному исходному.

Замыкание. В общем случае есть два сценария прекращения работы нормального алгорифма: по терминальной подстановке и по отсутствию подходящей подстановки. Можно модифицировать алгоритм так, что второй сценарий никогда не будет случаться. Такая модификация называется ***замыканием нормального алгорифма***. Выполняется замыкание добавлением в конец схемы подстановки $\Lambda \to \cdot\ \Lambda$. Наличие подобной подстановки в конце схемы означает, что при любом варианте подстановка произойдет, т.е. прекращение работы алгоритма по отсутствию подходящей подстановки никогда не происходит.

Пример 7.5. Рассмотрим алгоритм $\mathfrak{A}$ над алфавитом A со схемой

$$\begin{cases} *\xi \to \xi *, \\ * \to \cdot\ P, \\ \Lambda \to *, \end{cases}$$

где символ $*$ не входит в A. Его замыканием, согласно определению, является алгоритм со схемой

$$\begin{cases} *\xi \to \xi *, \\ * \to\cdot\ P, \\ \Lambda \to *, \\ \Lambda \to\cdot\ \Lambda. \end{cases}$$

Отметим, что добавление в конец схемы еще одной подстановки по сути ничего не меняет: ни при каких условиях добавленная подстановка не будет применяться из-за предшествующей подстановки. Последнее надо трактовать так: исходный алгоритм уже является замкнутым.

Как показывает пример, наличие подстановки с пустой левой частью — достаточное условие замкнутости. С другой стороны, если в схеме алгоритма отсутствует подстановка с пустой левой частью, то, по крайней мере, есть одно слово в рассматриваемом алфавите, к которому неприменима ни одна подстановка схемы: пустое слово. Значит, наличие подстановки с пустой левой частью — и необходимое условие замкнутости.

Теорема 7.4. Пусть A — произвольный алфавит. Замыкание $\overline{\mathfrak{A}}$ алгоритма $\mathfrak{A}$ вполне эквивалентно исходному алгоритму $\mathfrak{A}$.

◀ Утверждение почти тривиально. Пусть X — исходное слово. Если в алгоритме gA есть подстановка τ, применимая к X, то эта же подстановка есть и в алгоритме $\overline{\mathfrak{A}}$, причем множество подстановок в схеме $\mathfrak{A}$, предшествующих τ, совпадает с множеством подстановок схемы $\overline{\mathfrak{A}}$, предшествующих τ. Значит, если к слову X в алгоритме $\mathfrak{A}$ будет применена подстановка τ, то и в алгоритме $\overline{\mathfrak{A}}$ будет применена она же. В этом случае результаты работы двух алгоритмов будут одинаковы, т.е. они дают одно и то же преобразование слова X и имеют одно и то же условие прекращения работы (это условие — терминальность τ).

Если в алгоритме $\mathfrak{A}$ нет ни одной подстановки, применимой к X, то этот алгоритм прекращает работу и итогом этой работы будет слово X. Для алгоритма $\overline{\mathfrak{A}}$ это означает, что к слову X может быть применена только последняя подстановка $\Lambda \to\cdot\ \Lambda$. В результате его применения слово X не изменится, а алгоритм завершит работу. Мы видим, что и в этом случае алгоритмы работают одинаково.

Наши рассуждения показывают, что пошаговая последовательность работы двух алгоритмов совпадает. Следовательно, они вполне эквивалентны. ▶

Композиция нормальных алгорифмов. Алгоритм $\mathfrak{A}$ (в нашем понимании нормальный алгорифм) перерабатывает исходное слово X в некоторое слово Y, т.е. задает некоторое отображение (словарную функцию) $\mathfrak{A}$: $D \to A^*$, $D \subset A^*$, т.е. частичную словарную функцию. Для таких функций есть привычная и естественная операция — композиция. Отображение $\mathfrak{B} \circ \mathfrak{A}$ — это функция $\mathfrak{B}(\mathfrak{A}(X))$, областью определения которой является множество $\mathfrak{A}^{-1}(\mathrm{dom}(\mathfrak{B}))$, где $\mathrm{dom}(\mathfrak{B})$ — область определения функции $\mathfrak{B}$, а $\mathfrak{A}^{-1}(Y)$ — стандартное обозначение полного прообраза множества Y.

Два алгоритма $\mathfrak{A}$ и $\mathfrak{B}$ задают словарную функцию $\mathfrak{B} \circ \mathfrak{A}$, которая описывает последовательное применение двух алгоритмов (рис. 7.1). Ясно, что эта функция эффективно вычислимая (налицо конечная последовательность правил, описывающих получение значения этой функции). Однако возникает вопрос: можно ли эту функцию задать нормальным алгорифмом? Исходя из принципа нормализации мы должны дать положительный ответ. Но принцип — всего лишь надежда, возможно уверенность, но никак не строгое математическое утверждение. Мы постоянно должны подтверждать этот принцип, проверяя его в конкретных ситуациях, как и в ситуации с композицией.

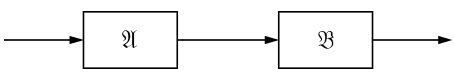


Рис. 7.1

Теорема 7.5. Пусть $\mathfrak{A}$ и $\mathfrak{B}$ — нормальные алгорифмы в алфавите A. Существует нормальный алгорифм $\mathfrak{C}$ над алфавитом A, такой, что для любого слова $X \in A^*$ выполняется соотношение $\mathfrak{C}(X) \simeq \mathfrak{B}(\mathfrak{A}(X))$.

◀ Здесь „существует“ — вполне в духе математической логики и оснований математики: речь идет о построении конкретной схемы, обеспечивающей нужное свойство.

Собственно, надо соединить две схемы в одну, но при этом обеспечить, чтобы до определенного момента подходили подстановки только первой схемы $\mathfrak{A}$, а затем только второй схемы $\mathfrak{B}$. Первое сделать нетрудно: достаточно расположить подстановки первой схемы перед подстановками второй. Чтобы не было проскакивания (когда ни одна подстановка алгоритма $\mathfrak{A}$ не подходит), алгоритм $\mathfrak{A}$ надо замкнуть. Чтобы после завершения работы $\mathfrak{A}$ можно было продолжить, надо схему $\mathfrak{A}$ перестроить, изменив терминальные подстановки. При этом каждая такая подстановка должна инициировать процесс, при котором слово изменяется так, что ни одна подстановка алгоритма $\mathfrak{A}$ не будет применена. После этого заработает алгоритм $\mathfrak{B}$, остановку которого надо переделать в заключительный процесс. Вот эти туманные соображения можно реализовать следующим образом.

Рассмотрим алфавит $\tilde{A}$, который содержит столько же букв, сколько и A, причем $A \cap \tilde{A} = \varnothing$. Тогда у каждой буквы $x \in A$ появляется двойник $\tilde{x}$. Выберем еще две разные буквы p и q, не входящие ни в A, ни в $\tilde{A}$. Новые буквы используем для построения схемы с нужными свойствами. Алгоритм $\mathfrak{A}$ замкнем (если он не замкнут) и в схеме замыкания алгоритма $\mathfrak{A}$ (применяемого первым) заменим каждую подстановку вида $P \to\cdot\, Q$ подстановкой $P \to pQ$ (слова P и Q могут быть пустыми). Полученную схему обозначим $\mathfrak{A}_p$.

Алгоритм $\mathfrak{B}$ замкнем, затем в его схеме заменим все буквы алфавита A на их двойники из алфавита $\tilde{A}$. Кроме того, заменим формулы вида $\Lambda \to P$ на $p \to pP$, а все терминальные формулы вида $P \to\cdot\, Q$ заменяем на $P \to qQ$. Полученную схему обозначим $\mathfrak{B}_{pq}$. Теперь строим следующую схему:

$$\begin{cases} \xi p \to p\xi & \text{(букву } p \text{ протаскиваем в начало)},\\ p\xi \to p\tilde{\xi} & \text{(первую букву меняем на двойника)},\\ \tilde{\xi}\eta \to \tilde{\xi}\tilde{\eta} & \text{(остальные буквы меняем двойниками)},\\ \tilde{\xi} q \to q\tilde{\xi} & \text{(букву } q \text{ протаскиваем в начало)},\\ q\tilde{\xi} \to q\xi & \text{(двойника первой буквы меняем на саму букву)},\\ \xi\tilde{\eta} \to \xi\eta & \text{(остальные двойники меняем на буквы)},\\ pq \to\cdot\, \Lambda & \text{(заканчиваем)},\\ \mathfrak{B}_{pq}, & \\ \mathfrak{A}_p. & \end{cases}$$

Здесь ξ, η — произвольная буква алфавита A, а $\tilde{\xi}$, $\tilde{\eta}$ — их двойники из алфавита $\tilde{A}$.

В сконструированной схеме первые три групповые подстановки перетаскивают спецсимвол p в начало слова и заменяют все буквы двойниками, подготавливая работу второго алгоритма. Запускаются эти подстановки, когда в слове появляется символ p, т.е. когда завершит работу группа $\mathfrak{A}_p$. Следующие три групповые подстановки делают обратную операцию, возвращая взамен двойников изначальные буквы. Седьмая строка завершает работу алгоритма.

Из построения видно, что в результате работы сконструированной схемы мы получим последовательную работу алгоритмов $\mathfrak{A}$ и $\mathfrak{B}$. ▶

Рассмотрим случай композиции алгоритмов в разных алфавитах.

Теорема 7.6 (общая теорема композиции). Для любого нормального алгорифма $\mathfrak{A}$ в алфавите A и нормального алгорифма $\mathfrak{B}$ в алфавите B существует нормальный алгорифм $\mathfrak{C}$ над алфавитом $A \cup B$, для которого $\mathfrak{C}(X) \simeq \mathfrak{B}(\mathfrak{A}(X))$, $X \in A^*$.

◄ Построим формальные расширения $\mathfrak{A}_1$, $\mathfrak{B}_1$ алгоритмов $\mathfrak{A}$ и $\mathfrak{B}$ на алфавит $C = A \cup B$. Согласно теореме 7.5, существует нормальный алгорифм $\mathfrak{C}$, для которого $\mathfrak{B}_1(\mathfrak{A}_1(X)) \simeq \mathfrak{C}(X)$, $X \in C^*$. В силу эквивалентности $\mathfrak{A}$ и $\mathfrak{A}_1$, $\mathfrak{B}$ и $\mathfrak{B}_1$ получаем $\mathfrak{B}(\mathfrak{A}(X)) \simeq \mathfrak{C}(X)$, $X \in C^*$. ►

Замечание 7.1. Теорему 7.6 можно переформулировать, заменив понятие „алгоритм в алфавите“ понятием „алгоритм над алфавитом“. Действительно, если $\mathfrak{A}$ — алгоритм над алфавитом A, $\mathfrak{B}$ — алгоритм над алфавитом B, то $\mathfrak{A}$ — алгоритм в алфавите $\tilde{A} \supset A$, который получен добавлением к A спецсимволов алгоритма $\mathfrak{A}$. Аналогично $\mathfrak{B}$ — алгоритм в алфавите $\tilde{B}$. Значит, существует нормальный алгорифм $\mathfrak{B} \circ \mathfrak{A}$ над алфавитом $\tilde{C} = \tilde{A} \cup \tilde{B}$. Сузив алфавит $\tilde{C}$ до $C = A \cup B$, получим требуемое утверждение.

Композицию нормальных алгорифмов $\mathfrak{A}$ и $\mathfrak{B}$ будем обозначать как композицию функций: $\mathfrak{B} \circ \mathfrak{A}$ (справа налево).

Соединение нормальных алгорифмов. Соединением алгоритмов $\mathfrak{A}$ и $\mathfrak{B}$ в алфавите A называется алгоритм $\mathfrak{C} = \mathfrak{A}\&\mathfrak{B}$, который каждому слову $X \in A^*$ ставит в соответствие слово $\mathfrak{A}(X)\mathfrak{B}(X)$ (рис. 7.2). Как и в случае композиции в данном случае необходимо показать, что соединение нормальных алгорифмов может быть реализовано как нормальный алгорифм.

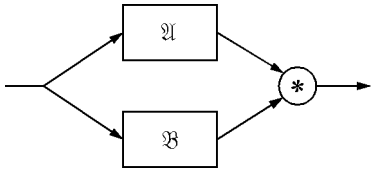


Рис. 7.2

Теорема 7.7. Для любых нормальных алгорифмов $\mathfrak{A}$ и $\mathfrak{B}$ в алфавите A существует нормальный алгорифм $\mathfrak{C}$ над алфавитом A, для которого $\mathfrak{C}(X) \simeq \mathfrak{A}(X)\mathfrak{B}(X)$, $X \in A^*$.

◄ Процесс перехода от X к $\mathfrak{A}(X)\mathfrak{B}(X)$ можно представить как последовательность преобразований:

$$X \to X\,\overline{X} \to \overline{\mathfrak{A}}(\overline{X})X = \overline{\mathfrak{A}(X)}X \to \overline{\mathfrak{A}(X)}\mathfrak{B}(X) \to \mathfrak{A}(X)\mathfrak{B}(X),$$

где черта сверху обозначает слово-двойник или алгоритм в алфавите-двойнике. Общее преобразование представляет собой композицию нескольких алгоритмов. Построим эти алгоритмы.

Первое преобразование осуществляется последовательным применением двух алгоритмов $\mathfrak{C}_1$ со схемой $\big(p\xi \to \xi\overline{\xi}p,\ p \to\cdot\ \Lambda,\ \Lambda \to p\big)$ и $\mathfrak{C}_2$ со схемой $\overline{\xi}\eta \to \eta\overline{\xi}$.

Второе преобразование осуществляется переводом $\overline{\mathfrak{A}}$ алгоритма $\mathfrak{A}$ на алфавит $\overline{A}$. Третье преобразование — с помощью алгоритма $\mathfrak{B}$. Наконец, необходимо вместо букв-двойников подставить оригиналы. Это осуществляется алгоритмом со схемой $\overline{\xi} \to \xi$. ►

Доказанную теорему расширяем на более общий случай алгоритмов в разных алфавитах.

Теорема 7.8. Для любого нормального алгорифма $\mathfrak{A}$ в алфавите A и нормального алгорифма $\mathfrak{B}$ в алфавите B существует нормальный алгорифм $\mathfrak{C}$ в алфавите $C = A \cup B$, для которого $\mathfrak{C}(X) \simeq \mathfrak{A}(X)\mathfrak{B}(X)$, $X \in C^*$.

◄ Как и в случае композиции, строим формальные расширения алгоритмов $\mathfrak{A}$ и $\mathfrak{B}$ на алфавит C, а затем применяем теорему 7.7. ►

Следствие 7.1. Для любого нормального алгорифма $\mathfrak{A}$ в алфавите A, нормального алгорифма $\mathfrak{B}$ в алфавите B и буквы p существует нормальный алгорифм $\mathfrak{C}$ в алфавите $C = A \cup B \cup \{p\}$, для которого $\mathfrak{C}(X) \simeq \mathfrak{A}(X)p\mathfrak{B}(X)$, $X \in (A \cup B)^*$.

◄ Алгоритм $\mathfrak{C}$ есть соединение трех алгоритмов: $\mathfrak{A}$, алгоритма $\mathfrak{P}$, который любое слово в алфавите $A \cup B$ заменяет на букву p, и алфавита $\mathfrak{B}$. Отметим, что объединение алфавитов ассоциативно: это просто поточечная конкатенация словарных функций, а теоремы лишь утверждают, что любая такая конкатенация может быть реализована как нормальный алгорифм. ►

Разветвление нормальных алгорифмов. На практике бывает, что при реализации того или иного метода в данный момент надо выбрать одно из возможных действий в зависимости от полученного результата. В языках программирования это реализуется условными конструкциями, например if … else …end. С точки зрения алгоритмов здесь участвуют три алгоритма $\mathfrak{L}$, $\mathfrak{A}$, $\mathfrak{B}$: первый $\mathfrak{L}$ формирует условие, при выполнении которого реализуется второй алгоритм $\mathfrak{A}$, а при нарушении — третий $\mathfrak{B}$ (рис. 7.3). Конкретизировать это общее описание можно с помощью словарных функции следующим образом:

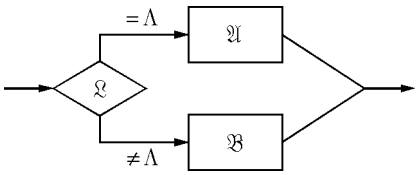


Рис. 7.3

$$\mathfrak{C}(X) = \begin{cases} \mathfrak{A}(X), & \mathfrak{L}(X) = \Lambda; \\ \mathfrak{B}(X), & \mathfrak{L}(X) \neq \Lambda. \end{cases}$$

Соответствующий алгоритм $\mathfrak{C}$ однозначно описывается следующими условиями:

1) $X \in A^* \wedge !\mathfrak{L}(X) \wedge (\mathfrak{L}(X) = \Lambda) \to \mathfrak{C}(X) \simeq \mathfrak{A}(X)$;
2) $X \in A^* \wedge !\mathfrak{L}(X) \wedge (\mathfrak{L}(X) \neq \Lambda) \to \mathfrak{C}(X) \simeq \mathfrak{B}(X)$;
3) $X \in A^* \to (\neg !\mathfrak{L}(X) \to \neg !\mathfrak{C}(X))$.

Такой алгоритм будем называть ***разветвлением на алгоритмы $\mathfrak{A}$ и $\mathfrak{B}$ под управлением $\mathfrak{L}$*** и обозначать $S(\mathfrak{L}, \mathfrak{A}, \mathfrak{B})$ (от английского select).

Ясно, что если все словарные функции $\mathfrak{L}$, $\mathfrak{A}$, $\mathfrak{B}$, эффективно вычислимы, то построенная функция $\mathfrak{C}$ эффективно вычислима. Как реализовать эту конструкцию в виде нормального алгорифма?

Теорема 7.9. Пусть $\mathfrak{L}$, $\mathfrak{A}$, $\mathfrak{B}$ — нормальные алгорифмы в алфавите A. Тогда существует нормальный алгорифм $\mathfrak{C}$ над алфавитом A, осуществляющий разветвление на $\mathfrak{A}$ и $\mathfrak{B}$ под управлением $\mathfrak{L}$.

◀ Один из вариантов построения требуемого алгоритма такой. На основе алгоритма $\mathfrak{L}$ с помощью некоторой буквы $p \notin A$ строим алгоритм $\mathfrak{L}_1$, обладающий свойствами: $!\mathfrak{L}(x) \Leftrightarrow !\mathfrak{L}_1(X)$; $\mathfrak{L}_1(X) = pX$, если $\mathfrak{L}(X) = \Lambda$; $\mathfrak{L}_1(X) = X$, если $\mathfrak{L}(X) \neq \Lambda$. Затем устраиваем композицию $\mathfrak{L}_1$, $\mathfrak{A}$ и $\mathfrak{B}$, но при этом скорректируем алгоритмы $\mathfrak{A}$ и $\mathfrak{B}$ так, чтобы отрабатывал только один из них. Для этого строим алгоритм $\mathfrak{B}_1$ следующим образом: берем замыкание алгоритма $\mathfrak{B}$, в начало его схемы добавляем подстановку $p \to\cdot \Lambda$, а в самой схеме каждую терминальную подстановку вида $P_i \to\cdot Q_i$ заменяем подстановкой $P_i \to\cdot qQ_i$, где $q \notin A$. Строим алгоритм $\mathfrak{A}_1$, добавляя в начало схемы алгоритма $\mathfrak{A}$ подстановку $q \to\cdot \Lambda$. После этого строим композицию $\mathfrak{C} = \mathfrak{A}_1 \circ \mathfrak{B}_1 \circ \mathfrak{L}_1$.

Работа построенного алгоритма протекает следующим образом. Для слова X алгоритм $\mathfrak{L}_1$ либо неприменим, либо дает pX, либо не изменяет слово X. Если $\mathfrak{L}_1$ (а значит, и $\mathfrak{L}$) неприменим к X, то и построенный алгоритм неприменим к X. Если алгоритм $\mathfrak{L}_1$ дает слово pX, то следующий алгоритм $\mathfrak{B}$ просто удаляет p и останавливается. Далее алгоритм $\mathfrak{A}$ работает со словом X. Если же алгоритм $\mathfrak{L}_1$ дает слово X, то далее алгоритм $\mathfrak{B}_1$ работает как $\mathfrak{B}$, давая в качестве результата слово $Y = \mathfrak{B}(X)$, в которое вставлена буква q. Для этого слова действие алгоритма $\mathfrak{A}_1$ состоит в удалении буквы q и остановке. Мы на выходе получаем $\mathfrak{B}(X)$.

Видно, что алгоритм $\mathfrak{C}$ удовлетворяет необходимым условиям. Остается построить алгоритм $\mathfrak{L}_1$. Сперва выбираем композицию $\mathfrak{L}_a \circ \mathfrak{L}$, где $\mathfrak{L}_a$ — нормальный алгорифм над A со схемой $(\xi\eta \to \xi,\ \xi \to\cdot \Lambda,\ \Lambda \to\cdot p)$, где $p \notin A$. В результате получаем алгорритм, который выдает два значения: Λ, если $\mathfrak{L}(X) \neq \Lambda$, и p, если $\mathfrak{L}(X) = \Lambda$. Теперь берем соединение построенного алгоритма с алгоритмом $\mathfrak{L}_b$ со схемой из одной подстановки $\Lambda \to\cdot \Lambda$ (он любое слово оставляет неизменным). ▶

Как и в случае объединения алгоритмов, доказанная теорема обобщается на случай неодинаковых алфавитов.

Теорема 7.10 (общая о разветвлении). Пусть $\mathfrak{L}$, $\mathfrak{A}$, $\mathfrak{B}$ — нормальные алгорифмы в алфавитах L, A и B соответственно. Существует нормальный алгорифм $\mathfrak{C}$ над алфавитом $C = A \cup B \cup L$, осуществляющий разветвление на $\mathfrak{A}$ и $\mathfrak{B}$ под управлением $\mathfrak{L}$.

◀ Достаточно устроить формальное расширение алфавитов на алфавит C и применить теорему 7.9. ▶

Повторение алгоритма. Разветвление алгоритма — первый пример работы под управлением алгоритма: алгоритм $\mathfrak{L}$ играет управляющую роль, которая в этом случае состоит в выборе одного из двух алгоритмов. Еще один пример работы под управлением — повторение алгоритма. Суть процедуры в том, что после применения некоторого алгоритма A результат тестируется с помощью еще одного алгоритма $\mathfrak{L}$. По результатам теста принимается решение, повторить алгоритм $\mathfrak{A}$ или остановить работу. Схематично эта процедура показана на рис. 7.4.

Будем говорить о том, что алгоритм $\mathfrak{C}$ реализует повторение алгоритма $\mathfrak{A}$ под управлением алгоритма $\mathfrak{L}$, если алгоритм $\mathfrak{C}$ применим к слову X тогда и только тогда, когда существуют слова X_0, X_1, …, $X_n = Y$, удовлетворяющие условиям $X_i = \mathfrak{A}(X_{i-1})$, $i = \overline{1, n}$, $\mathfrak{L}(X_i) \neq \Lambda$ при $i = \overline{1, n-1}$ и $\mathfrak{L}(X_n) = \Lambda$. При этом слово Y есть результат работы алгоритма $\mathfrak{C}(X)$.

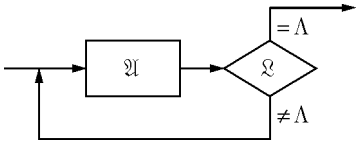


Рис. 7.4

Теорема 7.11. Для любых алгоритмов $\mathfrak{A}$ и $\mathfrak{L}$ в алфавите A существует нормальный алгорифм $\mathfrak{C}$ над алфавитом A, реализующий повторение $\mathfrak{A}$ под управлением $\mathfrak{L}$.

◀ Идея доказательства такова. По алгоритму $\mathfrak{L}$ строим алгоритм $\mathfrak{L}_1$, для которого $!\mathfrak{L}(X) \Leftrightarrow$ $\Leftrightarrow !\mathfrak{L}_1(X)$ и который добавляет к слову X символ p, если $\mathfrak{L}(X) = \Lambda$. Построение этого алгоритма описано в доказательстве теоремы 7.9. Рассмотрим алгоритм $\mathfrak{A}_1 = \mathfrak{L}_1 \circ \mathfrak{A}$. Он выполняет работу алгоритма $\mathfrak{A}$, а потом, если нужно, добавляет символ p слева, что сигнализирует о прекращении работы. Далее делаем следующее. Замыкаем $\mathfrak{A}_1$ и заменяем терминальные подстановки вида $P \to\cdot\ Q$ на подстановки $P \to qQ$ (буква q сигнализирует о завершении цикла). Получим алгоритм $\mathfrak{A}_q$. После этого строим схему

$$\begin{cases} \xi q \to q\xi, \\ pq \to\cdot\ q, \\ q \to \Lambda, \\ \mathfrak{A}_q. \end{cases}$$

Этот алгоритм решает поставленную задачу. ▶

Как и в других предшествующих ситуациях, обобщаем последнюю теорему.

Теорема 7.12 (общая о повторении). Для любого алгоритма $\mathfrak{A}$ в алфавите A и алгоритма $\mathfrak{L}$ в алфавите L существует алгоритм $\mathfrak{C}$ над алфавитом $C = A \cup L$, реализующий повторение $\mathfrak{A}$ под управлением $\mathfrak{L}$.

◀ Как и выше, достаточно построить формальное расширение алгоритмов на объединенный алфавит и затем применить теорему 7.11. ▶

Отметим, что реализованная схема повторения $\mathfrak{A}$ под управлением $\mathfrak{L}$ приводит к тому, что алгоритм $\mathfrak{A}$ выполняется по крайней мере один раз. Однако известно, что циклы можно

организовывать так, что тело цикла вообще может не выполняться ни разу. Предыдущий вариант алгоритма с повторением будем называть ***алгоритмом с постусловием*** и обозначать $RA(\mathfrak{L}, \mathfrak{A})$, а вариант алгоритма, когда цикл может ни разу не выполняться — ***алгоритмом с предусловием*** и обозначать $RB(\mathfrak{L}, \mathfrak{A})$. Схема такого алгоритма показана на рис. 7.5.

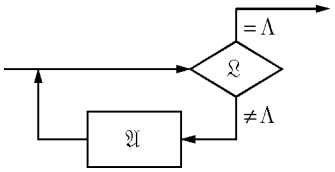


Рис. 7.5

Теорема 7.13. Для любого алгоритма $\mathfrak{A}$ в алфавите A и алгоритма $\mathfrak{L}$ в алфавите L существует алгоритм $\mathfrak{C}$ над алфавитом $C = A \cup L$, реализующий повторение $\mathfrak{A}$ под управлением $\mathfrak{L}$ следующим образом. Алгоритм $\mathfrak{C}$ применим к слову X тогда и только тогда, когда существуют слова $X_0 = X$, X_1, …, X_n, связанные соотношениями $X_i = \mathfrak{A}(X_{i-1})$, $i = \overline{1, n}$; $\mathfrak{L}(X_i) \neq \Lambda$, $i = \overline{0, n-1}$; $\mathfrak{L}(X_n) = \Lambda$.

◀ Построим алгоритм $\mathfrak{L}_1$, как выше, который к слову X добавляет символ p, если $\mathfrak{L}(X) = \Lambda$. Затем видоизменим алгоритм $\mathfrak{A}$, заменив разветвлением $(\mathfrak{A} \curlyvee \mathfrak{L}_1|\mathfrak{L})$ (т.е. разветвление на $\mathfrak{A}$ и $\mathfrak{L}_1$ под управлением $\mathfrak{L}$). Если $\mathfrak{L}(X) \neq \Lambda$, такой алгоритм работает как обычное повторение $\mathfrak{A}$, поскольку разветвление на $\mathfrak{A}$ и $\mathfrak{L}_1$ проходит после отрицательной проверки на $\mathfrak{L}$. Если же $\mathfrak{L}(X) = \Lambda$, то первое же разветвление даст $\mathfrak{L}_1$ с последующим завершением. что и требовалось. ▶

7.4. Теорема об универсальном нормальном алгорифме

Каждый алгоритм — заранее фиксированная последовательность действий. Его массовость обеспечивается произвольным характером исходного слова в данном алфавите. В таком контексте алгоритм можно уподобить программе для компьютера. Однако сам компьютер — тоже алгоритм: это автомат, который работает и с текстом программы, и с данными. Было бы печально, если бы требовалось при переходе к новой задаче менять компьютер.

Но если компьютер — тоже алгоритм, то его можно записать как нормальный алгорифм, для которого данными являются запись заданного нормального алгорифма (программа) и исходное слово, к которому нужно применить заданный алгоритм (данные). Такой нормальный алгорифм называют ***универсальным нормальным алгорифмом***.

Перейдем к точным формулировкам. Условимся, что задан и зафиксирован алфавит A, и мы рассматриваем нормальные алгорифмы в этом алфавите.

Сперва запись нормального алгорифма, а точнее, его схемы. Вводим новую букву $\gamma \notin A$ и рассматриваем слова в алфавите $A_\gamma = A \cup \gamma$ вида γP, где $P \in A_\gamma^*$ и заканчивается буквой γ. Каждое такое слово назовем ***системой слов*** в алфавите A (точнее γ-системой). Систему слов можно представить в виде

$$R = \gamma X_1 \gamma X_2 \ldots \gamma X_n \gamma,$$

где $X_i \in A^*$, $i = \overline{1, n}$, — ***элементы системы слов***. Отметим, что буква γ, рассматриваемая как слово, представляет собой ***пустую систему слов***.

Понятие системы слов позволяет представить в виде слова в некотором алфавите схему нормального алгорифма. Каждая подстановка есть упорядоченная пара слов в алфавите A и дополнительный атрибут (простая, терминальная). Вводим еще две буквы α и β, не входящие в A. Подстановку вида $P \to Q$ будем записывать в виде слова $P\alpha Q$, а заключительную подстановку $P \to\cdot\, Q$ — в виде слова $P\beta Q$. Систему таких слов назовем ***изображением нормального алгорифма***. Изображение нормального алгорифма $\mathfrak{A}$ будем обозначать $\mathfrak{A}^{\text{и}}$.

Пример 7.6. Для нормального алгорифма со схемой $(00 \to 00,\ 101 \to 0,\ 10 \to\cdot\ 1,\ 01 \to 01)$ изображение выглядит следующим образом:

$$\gamma 00\alpha 00\gamma 101\alpha 0\gamma 10\beta 1\gamma 01\alpha 01\gamma.$$

Введем четвертый спецсимвол δ и будем рассматривать слова вида $\mathfrak{A}^{\text{и}}\delta Q$, где $\mathfrak{A}^{\text{и}}$ — изображение нормального алгорифма $\mathfrak{A}$, а Q — слово в алфавите A. Нормальный алгорифм $\mathfrak{R}$ назовем ***универсальным нормальным алгорифмом***, если для любого нормального алгорифма $\mathfrak{A}$ в алфавите A и любого слова $Q \in A^*$ выполняется соотношение

$$\mathfrak{R}(\mathfrak{A}^{\text{и}}\delta Q) \simeq \mathfrak{A}(Q).$$

В дальнейшем в слове $\mathfrak{A}^{\text{и}}\delta Q$ левую часть $\mathfrak{A}^{\text{и}}$ (изображение нормального алгорифма) иногда будем называть программой, а правую часть Q — данными. Расширенный алфавит $A\alpha\beta\gamma\delta$ будем обозначать $\tilde{A}$.

Поскольку всевозможные пары „программа — данные“ представлены в виде слов в расширенном алфавите, согласно принципу нормализации универсальный алгорифм должен существовать.

Теорема 7.14. Универсальный нормальный алгорифм существует.

◀ Построение универсального алгорифма — сложная задача. Поэтому мы не станем строить конкретный алгорифм, а докажем его существование, опираясь на сочетания более простых алгорифмов.

В самом общем виде искомый алгоритм можно представить как повторение алгоритма $\mathfrak{P}_1$ под управлением алгоритма $\mathfrak{L}_2$ с последующим выполнением алгоритма $\mathfrak{P}_3$ (рис. 7.6). Алгоритм $\mathfrak{P}_1$ выполняет один шаг работы нормального алгорифма, т.е. либо находит подстановку и выполняет ее, либо ничего не делает, если подходящей подстановки нет. Алгоритм $\mathfrak{L}_2$ проверяет условие останова, т.е. была ли выполнена подстановка, и если была, то какая, терминальная или нет. Алгоритм $\mathfrak{P}_3$ убирает лишнее из конечной строки: спецсимволы и код программы.

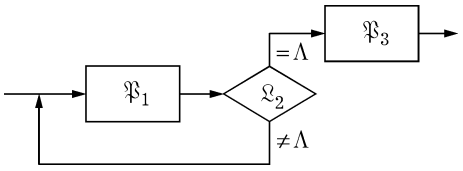


Рис. 7.6

Конкретизируем каждый из трех описанных алгоритмов. В ходе работы будем использовать спецсимволы p, q, u, t для фиксирования результатов работы: символ p будет указывать текущую подстановку в программе, находясь перед левой частью подстановки, символ q будет указывать текущий символ этой подстановки, а символы u и t — выделять фрагмент данных. Если в результате работы алгоритма все подстановки просмотрены и подходящей нет, символы pq будут находиться перед символом δ (за последней подстановкой программы). Если подходящая подстановка найдена и реализована в данных, то символ p будет располагаться перед этой подстановкой, а символ q — перед разделителем α или β в этой подстановке. Алгоритм $\mathfrak{L}_2$ должен проверять эти условия. Схема такого алгоритма может быть такой:

$$\begin{cases} \eta r \to r, & \eta \in Apq\alpha\beta\gamma, \\ r\eta \to r, & \eta \in Apq\alpha\beta\gamma\delta, \\ r \to\cdot\ \Lambda, \\ pq\delta \to r, \\ q\alpha \to\cdot\ q\alpha, \\ q\beta \to r. \end{cases}$$

Алгоритм $\mathfrak{P}_3$ просто удаляет спецсимволы p, q, u, t и все слева от δ (включая δ):

$$\begin{cases} \eta \to \Lambda, & \eta \in pqut, \\ \eta\delta \to \delta, & \eta \in \tilde{A}p, \\ \delta \to\cdot\, \Lambda. \end{cases}$$

Остановимся на алгоритме $\mathfrak{P}_1$. Его схема показана на рис. 7.7. Он начинается с инициализирующего алгоритма $\mathfrak{P}_{11}$, за которым следует повторение алгоритма $\mathfrak{P}_{13}$ под управлением $\mathfrak{L}_{12}$. Алгоритм $\mathfrak{P}_{14}$ завершает работу $\mathfrak{P}_1$. Алгоритм $\mathfrak{P}_{11}$ устанавливает спецсимволы p, q, u, t в начальное положение: символы pq перед первой подстановкой, символы ut в начале данных (вслед за разделителем δ). Алгоритм $\mathfrak{L}_{12}$ проверяет конфигурацию спецсимволов, выявляя два условия: если подходящая подстановка найдена или если все подстановки просмотрены, формируется пустое значение. Все подстановки просмотрены, если символы pq расположены непосредственно перед разделителем δ. Текущая подстановка, отмеченная символом p подходящая, если символ q предшествует разделителю α или β этой подстановки. Алгоритм $\mathfrak{P}_{14}$ реализует подходящую подстановку или не выполняет никаких действий, если подходящая подстановка не найдена.

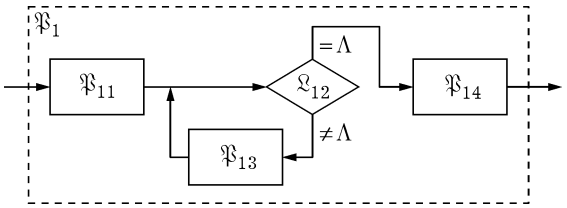


Рис. 7.7

Алгоритм $\mathfrak{P}_{11}$ можно реализовать как композицию $\mathfrak{P}_{113}\circ\mathfrak{P}_{112}\circ\mathfrak{P}_{111}$ трех простых алгоритмов: алгоритм $\mathfrak{P}_{111}$ удаляет спецсимволы (его схема $\eta \to \Lambda$, $\eta \in pqut$), алгоритм $\mathfrak{P}_{112}$ устанавливает в начальное положение символы pq (его схема $\gamma \to\cdot\, \gamma pq$), а алгоритм $\mathfrak{P}_{113}$ выполняет то же самое для символов ut (его схема $\delta \to\cdot\, \delta ut$).

Алгоритм $\mathfrak{L}_{12}$ проверят два условия: положение символа q в конце левой части подстановки, т.е. перед символом α или β, означает, что эта подстановка применима к данным; положение символов pq перед δ означает, что ни одна подстановка не применима к данным. И то, и другое означает выход из текущего шага работы алгоритма. Алгоритм $\mathfrak{L}_{12}$ можно задать схемой

$$\begin{cases} r\eta \to r, & \eta \in \tilde{A}pq, \\ \eta r \to r, & \eta \in \tilde{A}pq, \\ r \to\cdot\, \Lambda, \\ pq\delta \to r, \\ q\alpha \to r, \\ q\beta \to r. \end{cases}$$

Ключевой составной частью в $\mathfrak{P}_1$ является алгоритм $\mathfrak{P}_{13}$, который должен фактически проверить, является ли текущая подстановка подходящей. Такая проверка сводится к соответствующей расстановке спецсимволов. Если комбинация pq располагается перед δ, то все подстановки просмотрены и подходящей нет. Если символ q предшествует разделителю подстановки α или β, то подходящая подстановка выбрана, при этом символы u и t будут ограничивать вхождение левой части найденной подстановки в строку данных. В остальных случаях символы pq располагаются в начале следующей, еще не просмотренной подстановки. В соответствии со сказанным алгоритм $\mathfrak{P}_{13}$ можно реализовать как разветвление, которое представляет собой выбор одного из нескольких вариантов изменения положения спецсимволов.

Для объяснения сути указанного разветвления придадим следующий условный смысл спецсимволам: p — номер текущей подстановки; q — номер символа в левой части текущей подстановки; u — номер первого символа выделенного фрагмента данных; t — номер последнего символа выделенного фрагмента данных, номера u и t изменяются от 1 до m — длины строки данных. Структура алгоритма $\mathfrak{P}_{13}$ с содержательной интерпретацией структурных элементов показана на рис. 7.8. На рисунке условие $= \Lambda$ равносильно „да“, x_q и y_t обозначают символ подстановки с номером q и символ данных с номером t.

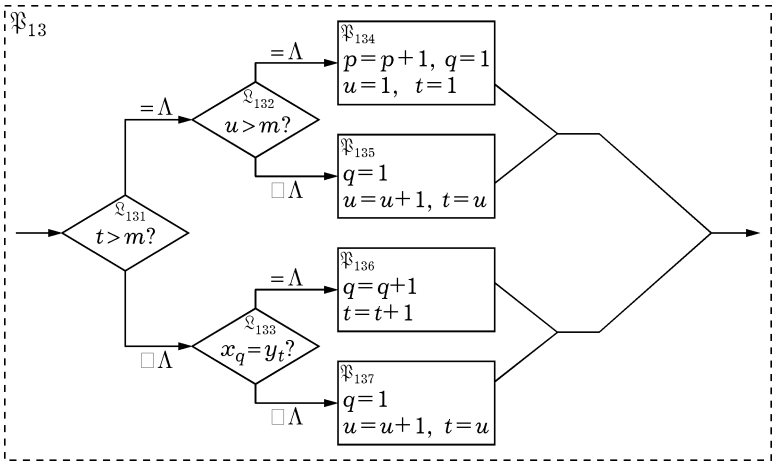


Рис. 7.8

Алгоритмы $\mathfrak{L}_{131}$ и $\mathfrak{L}_{132}$ однотипны и различаются используемым спецсимволом:

$$\mathfrak{L}_{131}\colon \begin{cases} r\eta \to r, & \eta \in \tilde{A}pq, \\ \eta r \to r, & \eta \in \tilde{A}pq, \\ r \to\cdot\ \Lambda, \\ t\xi \to\cdot\ .t\xi, \\ t \to r; \end{cases} \qquad \mathfrak{L}_{132}\colon \begin{cases} r\eta \to r, & \eta \in \tilde{A}pq, \\ \eta r \to r, & \eta \in \tilde{A}pq, \\ r \to\cdot\ \Lambda, \\ u\xi \to\cdot\ u\xi, \\ u \to r; \end{cases}$$

Алгоритм $\mathfrak{L}_{133}$ можно реализовать следующим образом:

$$\begin{cases} r\eta \to r, & \eta \in \tilde{A}pqut, \\ \eta r \to r, & \eta \in \tilde{A}pqut, \\ r \to\cdot\ \Lambda, \\ s\xi\eta \to \eta s\xi, & \xi \in A,\ \ \eta \in \tilde{A}u, \\ s\xi t\xi \to r, & \xi \in A, \\ s\xi t \to\cdot\ s\xi t, & \xi \in A, \\ q \to qs. \end{cases}$$

Алгоритм $\mathfrak{P}_{134}$ можно реализовать как композицию двух алгоритмов: $\mathfrak{P}_{134} = \mathfrak{P}_{1342} \circ \mathfrak{P}_{1341}$. Алгоритм $\mathfrak{P}_{1341}$ удаляет символ q, передвигает символ p на следующую подстановку и заканчивает работу вставкой символа q сразу после p. Алгоритм $\mathfrak{P}_{1342}$ удаляет символы u и t, а затем вставляет после δ:

$$\mathfrak{P}_{1341}\colon \begin{cases} q \to \Lambda, \\ p\eta \to \eta p, & \eta \in A\alpha\beta, \\ p\gamma \to\cdot\ \gamma pq; \end{cases} \qquad \mathfrak{P}_{1342}\colon \begin{cases} u \to \Lambda, \\ t \to \Lambda, \\ \delta \to\cdot\ \delta ut. \end{cases}$$

Алгоритм $\mathfrak{P}_{135}$ также удобно разделить в композицию: $\mathfrak{P}_{135} = \mathfrak{P}_{1352} \circ \mathfrak{P}_{1351}$, где $\mathfrak{P}_{1351}$ отрабатывает ситуацию с символами p и q, а $\mathfrak{P}_{1352}$ — с символами u и t:

$$\mathfrak{P}_{1351}\colon \begin{cases} q \to \Lambda, \\ p \to\cdot\, pq; \end{cases} \qquad \mathfrak{P}_{1352}\colon \begin{cases} t \to \Lambda, \\ u\xi \to\cdot\, \xi ut, \quad \xi \in A. \end{cases}$$

Алгоритм $\mathfrak{P}_{136}$ также разделяем в композицию алгоритма $\mathfrak{P}_{1361}$, передвигающего символ q, и алгоритма $\mathfrak{P}_{1362}$, передвигающего символ t. Каждый реализуется одной групповой подстановкой: первый $q\xi \to\cdot\, \xi q$, второй $t\xi \to\cdot\, \xi t$.

Осталось описать алгоритм $\mathfrak{P}_{14}$. Этот алгоритм можно представить в виде разветвления $S(\mathfrak{L}_{141}, \mathfrak{P}_{142}, \mathfrak{P}_{143})$, в котором выполняется $\mathfrak{P}_{142}$, если $\mathfrak{L}_{141}(X) = \Lambda$, и $\mathfrak{P}_{143}$ в противном случае. Алгоритм $\mathfrak{L}_{141}$ проверяет, следует ли в данных выполнить замену. Его схема:

$$\begin{cases} r\eta \to r, \\ \eta r \to r, \\ r \to\cdot\, \Lambda, \\ pq\delta \to r. \end{cases}$$

Алгоритм $\mathfrak{P}_{142}$ ничего не делает: $\Lambda \to\cdot\, \Lambda$, а алгоритм $\mathfrak{P}_{143}$ выполняет подстановку, заменяя фрагмент данных между u и t правой частью подстановки. Его можно представить как композицию алгоритма $\mathfrak{P}_{1431}$, который стирает фрагмент между символами u и t, и алгоритма $\mathfrak{P}_{1432}$, который копирует правую часть подстановки в место между u и t. Алгоритм $\mathfrak{P}_{1431}$ можно задать схемой

$$\begin{cases} u\xi \to u, \quad \xi \in A, \\ ut \to\cdot\, ut. \end{cases}$$

Алгоритм $\mathfrak{P}_{1432}$ можно представить схемой

$$\begin{cases} s\xi\eta \to \eta s\xi, \quad \xi \in A, \ \ \eta \in \tilde{A}u, \\ s\xi t \to \xi t, \quad \xi \in A, \\ q\gamma \to\cdot\, \gamma, \\ q\xi \to \xi q s\xi, \quad \xi \in A, \\ q\alpha \to q\alpha q, \\ q\beta \to q\beta q. \end{cases}$$

Отметим, что этот алгоритм начинает работу в ситуации, когда символ q находится перед разделителем подстановки α или β. А алгоритме формируется второй символ q, помещаемый в правую часть подстановки. После завершения работы первое вхождение символа q остается перед разделителем, сигнализируя, что алгоритм $\mathfrak{P}_1$ выполнил подстановку, и обозначая тип этой подстановки. ►

Замечание 7.2. Из доказательства теоремы 7.14 ясно, что универсальный алгоритм устроен достаточно сложно, и привести конкретную схему этого алгоритма непросто. Однако такие построения были выполнены разными исследователями.

Приведем пример нормального алгорифма, построенного В.Г. Жаровым*. Этот алгорифм однократно применяет схему нормального алгорифма к исходному слову. Чтобы построить полновесный универсальный алгорифм, достаточно устроить его повторение. В нем использованы

*Вообще-то этот алгоритм построен для двухбуквенного алфавита. Однако он является рабочим и для алфавита с большим числом букв. Кроме того, он приписывает слева к слову спецсимвол π.

символы подстановки ξ и η, причем $\xi \in A$, а η пробегает алфавит, явно указанный. Кроме того, введено обозначение $\tilde{A} = A\alpha\beta\gamma$):

1. $\overline{\xi}\eta \to \eta\overline{\xi}, \quad \eta \in \tilde{A}u\delta;$
2. $\overline{\xi}t\xi \to \xi t;$
3. $\overline{\xi}t \to s;$
4. $\overline{\xi}r \to \xi r;$
5. $\eta s \to s\eta, \quad \eta \in \tilde{A}\delta;$
6. $us \to s\overline{u};$
7. $pqs \to u;$
8. $ps \to pq;$
9. $\overline{p}s\gamma \to \gamma pq;$
10. $qs \to$
11. $s \to pq$
12. $\overline{u}\xi \to s\xi ut$
13. $\overline{u} \to$
14. $v \to s$
15. $\overline{p}\eta \to \eta\overline{p}, \quad \eta \in A\alpha\beta;$
16. $\overline{p}q\eta \to \overline{\eta}\,\overline{q}, \quad \eta \in \alpha\beta;$
17. $pq\gamma \to \gamma pq;$
18. $q\xi \to \xi q\overline{\xi};$
19. $p \to \overline{p};$
20. $u\eta \to u, \quad \eta \in \tilde{A};$
21. $ut \to r;$
22. $u\delta \to \pi;$
23. $\overline{q} \to q;$
24. $r \to s;$
25. $\overline{\beta} \to sv;$
26. $\delta \to s\delta ut.$

Свойства приведенного алгоритма:

1) $\mathfrak{A}^{\text{и}}\delta Q \vDash pq\mathfrak{A}^{\text{и}}s\delta utQ$. Это начало работы алгоритма, состоящее в применении подстановки 26.

2) $pq\gamma X \vDash \gamma pqX$, где $X \in (\tilde{A}u\delta t)^*$. Этот этап реализуется в перемещении s в начало слова (подстановка 5), затем его замене на pq (подстановка 11), и смещении вставленной пары букв на одну позицию вправо (подстановка 17).

3) если $Q \not\subset S$, то $LpqQM\gamma N\delta utR \vDash LQM\gamma pqN\delta S$, где L, $N \in \tilde{A}$, $M \in A\alpha\beta$, Q, $S \in A$. Это основная часть работы алгоритма, соответствующая побуквенному сравнению, изложенному выше. Очередная буква-двойник смещается вправо до символа t с помощью подстановки 1, затем применяется подстановка 2 (сравнение успешное) или 3 (сравнение неуспешное)

4) если $Q \subset S$, то $LpqQ\alpha R\gamma N\delta utS \vDash LqQ\alpha R\gamma N\delta S^Q_R$ и $LpqQ\beta R\gamma N\delta utS \vDash \pi S^Q_R$, где L, $N \in \tilde{A}$, Q, R, $S \in A$.

Работу алгоритма Жарова проиллюстрируем на примере алгоритма со схемой, состоящей из единственной подстановки $ab \to c$, и исходного слова aabc. В представленной последовательности для наглядности выполнены замены α на $>$, β на $\geqslant$, γ на $\_$ и δ на $+$, номер примененной

подстановки указан слева:

_ab⩾c_+aabc
(26) _ab⩾c_s+utaabc
(5) _ab⩾cs_+utaabc
(5) _ab⩾sc_+utaabc
(5) _abs⩾c_+utaabc
(5) _asb⩾c_+utaabc
(5) _sab⩾c_+utaabc
(5) s_ab⩾c_+utaabc
(11) pq_ab⩾c_+utaabc
(17) _pqab⩾c_+utaabc
(18) _paqAb⩾c_+utaabc
(1) _paqbA⩾c_+utaabc
(1) _paqb⩾Ac_+utaabc
(1) _paqb⩾cA_+utaabc
(1) _paqb⩾c_A+utaabc
(1) _paqb⩾c_+Autaabc
(1) _paqb⩾c_+uAtaabc
(2) _paqb⩾c_+uatabc
(18) _pabqB⩾c_+uatabc
(1) _pabq⩾Bc_+uatabc
(1) _pabq⩾cB_+uatabc
(1) _pabq⩾c_B+uatabc
(1) _pabq⩾c_+Buatabc
(1) _pabq⩾c_+uBatabc
(1) _pabq⩾c_+uaBtabc
(3) _pabq⩾c_+uasabc
(5) _pabq⩾c_+usaabc
(6) _pabq⩾c_+sUaabc
(5) _pabq⩾c_s+Uaabc
(5) _pabq⩾cs_+Uaabc
(5) _pabq⩾sc_+Uaabc
(5) _pabqs⩾c_+Uaabc
(10) _pab⩾c_+Uaabc
(12) _pab⩾c_+sautabc
(5) _pab⩾c_s+autabc
(5) _pab⩾cs_+autabc
(5) _pab⩾sc_+autabc
(5) _pabs⩾c_+autabc
(5) _pasb⩾c_+autabc
(5) _psab⩾c_+autabc
(8) _pqab⩾c_+autabc
(18) _paqAb⩾c_+autabc
(1) _paqbA⩾c_+autabc
(1) _paqb⩾Ac_+autabc
(1) _paqb⩾cA_+autabc
(1) _paqb⩾c_A+autabc
(1) _paqb⩾c_+Aautabc
(1) _paqb⩾c_+aAutabc
(1) _paqb⩾c_+auAtabc
(2) _paqb⩾c_+auatbc
(18) _pabqB⩾c_+auatbc
(1) _pabq⩾Bc_+auatbc
(1) _pabq⩾cB_+auatbc
(1) _pabq⩾c_B+auatbc
(1) _pabq⩾c_+Bauatbc
(1) _pabq⩾c_+aBuatbc
(1) _pabq⩾c_+auBatbc
(1) _pabq⩾c_+auaBtbc
(2) _pabq⩾c_+auabtc
(19) _Pabq⩾c_+auabtc
(15) _aPbq⩾c_+auabtc
(15) _abPq⩾c_+auabtc
(16) _ab⩾Qc_+auabtc
(20) _ab⩾Qc_+aubtc
(20) _ab⩾Qc_+autc
(21) _ab⩾Qc_+arc
(23) _ab⩾qc_+arc
(18) _ab⩾cqC_+arc
(1) _ab⩾cq_C+arc
(1) _ab⩾cq_+Carc
(1) _ab⩾cq_+aCrc
(4) _ab⩾cq_+acrc
(24) _ab⩾cq_+acsc
(5) _ab⩾cq_+ascc
(5) _ab⩾cq_+sacc
(5) _ab⩾cq_s+acc
(5) _ab⩾cqs_+acc
(10) _ab⩾c_+acc
(25) _absvc_+acc
(5) _asbvc_+acc
(5) _sabvc_+acc
(5) s_abvc_+acc
(11) pq_abvc_+acc
(14) pq_absc_+acc
(5) pq_asbc_+acc
(5) pq_sabc_+acc
(5) pqs_abc_+acc
(7) u_abc_+acc
(20) uabc_+acc
(20) ubc_+acc
(20) uc_+acc
(20) u_+acc
(20) u+acc
(22) wacc
wacc

8. МАШИНЫ ТЬЮРИНГА

8.1. Основные понятия

Следующий вариант строгого определения алгоритма — представление результата как результата работы некоего автоматического устройства — машины. В принципе машина — дискретно работающее устройство, которое в каждый момент времени находится в одном из внутренних состояний. Внутреннее состояние меняется за один шаг в зависимости от текущего состояния и входных данных. За один шаг машина формирует какие-то выходные данные.

Разумеется, результат действий абстрактной машины можно описать без „костылей“ — представления их как результат деятельности некоторого механического устройства. Есть объект, который описывает доступную в данный момент входную информацию, есть множество внутренних состояний. В этом контексте речь идет о некотором отображении, которое паре множеств ставит в соответствие новое множество — множество выходных данных.

Машина Тьюринга — простейший вариант описанной абстрактной машины. Есть целый ряд ее модификаций. Опишем одну из них. Зададимся некоторым непустым алфавитом $A = \{a_0, a_1, \ldots, a_n\}$, который назовем ***внешним алфавитом***. В этом алфавите буква a_0 играет особую роль. Введем еще ***внутренний алфавит*** $Q = \{q_0, q_1, q_2, \ldots, q_m\}$, определяющий ***множество внутренних состояний*** машины. Считаем, что внутренний алфавит имеет по крайней мере два элемента q_0 и q_1, играющих специальную роль. В объединенный алфавит дополнительно включим четыре спецсимвола $\to$, R, L, C. Составляем слова специального вида $q_i a_j \to q_l a_k M$, где $1 \leqslant i \leqslant m$, $0 \leqslant l \leqslant m$, $0 \leqslant j, k \leqslant n$, а M — один из трех символов R, L, C. Множество таких слов, имеющих разные левые части (два символа перед $\to$), назовем ***программой*** (заметим, что символ $\to$ на самом деле не нужен и используется лишь для наглядности).

В дальнейшем символ a_0 будем называть ***пустым символом***, символ q_0 — ***заключительным состоянием***, символ q_1 — ***начальным состоянием***.

Действие машины состоит в следующем. Есть лента, разделенная на конечное число ячеек. Лента потенциально бесконечная, т.е. в любой момент к ней можно добавлять ячейки и слева (в начало) и справа (в конец)*. Каждая ячейка заполнена одним символом внешнего алфавита. Добавляемые ячейки заполняются пустым символом a_0. У машины есть ***читающая головка***, которая обозревает ленту. В каждый момент времени головка обозревает одну ячейку с некоторым символом a_j и находится в одном из внутренних состояний q_i. За один шаг работы машина изменяет свое внутреннее состояние, меняет обозреваемый головкой символ на ленте и может на одну позицию влево или вправо сместить положение читающей головки. Это происходит в соответствии с командой $q_i a_j \to q_l a_k M$, которая вызывает замену текущего символа a_j символом a_k, смену внутреннего состояния с q_i на q_l. При этом, если $M = R$, то читающая головка смещается вправо, если $M = L$, то влево, если же $M = C$, то остается на месте.

Шаги работы машины повторяются до тех пор, пока не произойдет одно из двух событий: 1) для текущего состояния и обозреваемого символа нет команды; 2) текущая команда привела к заключительному внутреннему состоянию q_0. В этом случае машина останавливается, а слово, оказавшееся на ленте, считается результатом работы машины.

В действительности представления о ленте и читающей головке, хотя и являются общеупотребительными, нужны лишь для наглядности. С математической точки зрения эти понятия

*Можно сказать и по-другому: лента бесконечная в обе стороны, но количество ячеек, заполненных непустыми символами, конечно.

ничтожные (как выражаются юристы). Вместо трех понятий „текущее состояние“, „обозреваемый символ“, „лента“ введем одно понятие: ***конфигурация*** (или машинное слово). Это понятие обозначает слово вида Xq_la_jY, где $X, Y \in A^*$, т.е. слово во внешнем алфавите A, в которое добавлена одна буква из внутреннего алфавита. Буква внутреннего алфавита отмечает текущее состояние машины Тьюринга, следующий за ней символ будет обозреваемым. Так, в конфигурации $a_3a_2q_1a_2a_1$ зафиксировано текущее состояние q_1 (начальное) и обозреваемый символ a_2. Машина Тьюринга реализует преобразование одной конфигурации в другую.

Дадим формальное описание действий, совершаемых машиной Тьюринга за один такт. Предполагаем, что текущая конфигурация имеет вид Xq_ia_jY.

• если в программе машины есть команда $q_ia_j \to q_ka_lR$ и $Y \neq \varnothing$, то новая конфигурация Xa_lq_kY;

• если в программе машины есть команда $q_ia_j \to q_ka_lR$ и $Y = \varnothing$, то новая конфигурация $Xa_lq_ka_0$;

• если в программе машины есть команда $q_ia_j \to q_ka_lL$ и $X = X_1a_m \neq \varnothing$, то новая конфигурация $X_1q_ka_ma_lY$;

• если в программе машины есть команда $q_ia_j \to q_ka_lL$ и $X = \varnothing$, то новая конфигурация $q_ka_0a_lY$;

• если в программе машины есть команда $q_ia_j \to q_ka_lC$, то новая конфигурация Xq_ka_lY;

Если конфигурация M' получена из конфигурации M за один такт работы машины Тьюринга, то будем говорить, что машина Тьюринга переводит M в M'. Машина Тьюринга преобразует конфигурацию M в конфигурацию $\tilde{M}$, если существует такая последовательность конфигураций $M_0 = M$, M_1, …, $M_n = \tilde{M}$, что конфигурация M_i, $i = \overline{0, n-1}$, машиной Тьюринга переводится в конфигурацию M_{i+1}. ***Вычислением*** машины Тьюринга назовем такое ее преобразование M в $\tilde{M}$, что конфигурация $\tilde{M}$ является заключительной (терминальной), т.е. либо нет команды, соответствующей ее левой части, либо эта конфигурация соответствует заключительному состоянию.

Существуют разные модификации машины Тьюринга, в основном различающиеся в трех аспектах.

Во-первых, операции сдвига читающей головки могут оформляться как самостоятельные, т.е. при сдвиге головки обозреваемый символ не изменяется (это так называемый вариант Поста). В этом случае команды машины имеют вид $q_ia_j \to q_lb$, где $b \in A \cup \{L, R\}$. Если дана, например, команда $q_ia_j \to q_lR$, то машина изменяет свое внутреннее состояние с q_i на q_l и сдвигает вправо читающую головку, а обозреваемый символ a_j не изменяется.

Вариант Поста машины Тьюринга можно считать частным случаем рассматриваемого варианта (варианта Клини), поскольку всегда команды вида $q_ia_j \to q_lL$ и $q_ia_j \to q_lR$ можно трансформировать в команды $q_ia_j \to q_la_jL$ и $q_ia_j \to q_la_jR$, а команды вида $q_ia_j \to q_la_k$ — в команды $q_ia_j \to q_la_kC$. В то же время можно произвольную программу трансформировать так, что операции изменения символа на ленте и смещения читающей головки будут выполняться раздельно. Для этого достаточно для каждого состояния q_i ввести два новых состояния q_i^p — сдвиг влево и q_i^s — сдвиг вправо. Команды для этих состояний должны иметь вид $q_i^p\xi \to q_i\xi L$ и $q_i^s\xi \to q_i\xi R$ (здесь ξ — любой символ внешнего алфавита). Затем команду с левым сдвигом читающей головки вида $q_ia_j \to q_ma_kL$ заменяем командой $q_ia_j \to q_m^pa_k$ и аналогично изменяем команды с правым сдвигом читающей головки.

Во-вторых, различия проявляются в условиях завершения. В описанном варианте есть два сценария останова машины Тьюринга: либо по отсутствию команды для текущего положения машины (т.е. комбинации внутреннего состояния и обозреваемого символа), либо по наступающему заключительному внутреннему состоянию. Эта ситуация аналогична нормальным алгорифмам. В действительности нет необходимости объявлять какое-либо состояние заключительным: заключительным является всякое состояние, которое не встречается в левой части команд в программе машины Тьюринга. Правда, в описанном варианте могут быть условно

заключительные состояния, которые приводят к останову лишь для некоторых обозреваемых символов.

Программу машины Тьюринга легко модифицировать так, что останов по отсутствию команды не будет происходить. Достаточно из программы удалить все команды для заключительного состояния q_0 (они все равно никогда не выполняются) и для каждого положения $q_i a_j$, для которого нет команды, добавить команду $q_i a_j \to q_0 a_j$. В результате программа будет содержать ровно $m(n+1)$ (количество незаключительных состояний на количество букв внешнего алфавита) команд, описывающих все незаключительные положения. Такую процедуру будем называть ***замыканием машины Тьюринга***.

В-третьих, машины Тьюринга различаются входными и выходными потоками. Так, рабочая лента может быть полуограниченной, т.е. неограниченное перемещение влево оказывается недоступно. Кроме того, могут использоваться несколько лент. Подробнее об этом будет сказано далее.

Далее в теоретических рассуждениях будем в основном предполагать, что машина Тьюринга замкнута, в то время как в конкретных примерах будем использовать незамкнутые программы, удаляя тривиальные команды типа $q_i a_j \to q_0 a_j m$, которые ничего не делают на ленте, а лишь приводят к останову машины Тьюринга.

С машиной Тьюринга можно связать словарную функцию следующим образом. Исходное слово X в алфавите A превращается в конфигурацию M_X приписыванием символа q_1 начального состояния к слову слева (т.е. фиксируется начальное состояние q_1 и читающая головка устанавливается на первый символ слова). Машина Тьюринга начинает работу с этой конфигурации. Если она останавливается на некоторой конфигурации M_T (по отсутствию команды или по заключительному состоянию — неважно), то слово, полученное из M_T удалением символа состояния, а также всех пустых символов в начале и конце слова считаем значением словарной функции на слове X. Если машина Тьюринга работает бесконечно (зацикливается), то считаем, что словарная функция не определена на слове X.

Пример 8.1. Рассмотрим алфавит $A = \{a,\ b,\ c\}$ и машину Тьюринга с программой

$$\begin{cases} q_1 b \to q_1 c R, \\ q_1 c \to q_1 b R, \\ q_1 a \to q_0 a. \end{cases}$$

Машина имеет два состояния q_0, q_1 (заключительное и начальное). Как сказано выше, начальная конфигурация машины Тьюринга $q_1 X$, где X — исходное слово. В процессе работы машины делает взаимные замены символов b и c, пока не встретит символ a (он играет роль пустого символа). Так, слово bab машина преобразует в слово cab, слово $bcbc$ — в слово $cbcb$ (останов произойдет, когда будет достигнут конец слова и для движения вправо будет добавлен пустой символ; этот символ в окончательном результате не учитывается).

Выделяют также в некотором смысле регулярные сценарии работы машины Тьюринга (***правильные вычисления***), при которых не происходит добавления пустых символов к слову справа и/или слева.

С помощью машины Тьюринга можно вычислять числовые функции $f(x_1, x_2, \ldots, x_n)$, $x_i \in \mathbb{N}$. Для этого выбираем машину Тьюринга с внешним алфавитом $A = \{0,\ 1\}$, в котором 0 играет роль пустого символа. Аргументы функции записываем в виде 0-системы, т.е. в виде $X = 01^{x_1}01^{x_2}0 \ldots 01^{x_n}0$. Машина Тьюринга с алфавитом $\{0,\ 1\}$ вычисляет функцию f, если соответствующая ей словарная функция преобразует слово X в слово 01^y0, где $y = f(x_1, x_2, \ldots, x_n)$, и не определена на слове X, если функция f не определена на сочетании аргументов x_1, x_2, …, x_n. Если функция $f(x_1, \ldots, x_n)$ может быть вычислена с помощью некоторой машины Тьюринга, ее называют ***вычислимой по Тьюрингу***.

Пример 8.2. Машина Тьюринга с алфавитом 01 и программой $q_1 0 \to q_0 1$, $q_1 1 \to q_1 1 L$ вычисляет функцию $f(x) = x + 1$ (инкремент). Отметим, что алгоритм использует смещение

читающей головки за левую границу слова. Для правильного вычисления инкремента, при котором не используется смещение за левую границу (или еще более жестко — и за правую границу), а заключительное положение читающей головки — первый символ слова, можно использовать машину Тьюринга с программой

$$\begin{cases} q_1 0 \to q_2 0 R, \\ q_1 1 \to q_1 1 \quad \text{(Неверное слово!)}, \\ q_2 1 \to q_2 1 R, \\ q_2 0 \to q_3 1, \\ q_3 1 \to q_3 1 L, \\ q_3 0 \to q_0 0. \end{cases}$$

Эта программа смещает читающую головку на конец слова, где завершающий нуль меняется на единицу, а затем головка смещается к началу слова, определяемому символом 0.

8.2. Сочетания машин Тьюринга

Как и нормальные алгорифмы, машины Тьюринга можно сочетать определенным образом. Базовой операцией является композиция машин Тьюринга.

Пусть T_1 и T_2 — две машины Тьюринга с внутренними алфавитами $Q_1 = \{q_0^1,\ q_1^1,\ \ldots,\ a_{m_1}^1\}$ и $Q_2 = \{q_0^2,\ q_1^2,\ \ldots,\ a_{m_2}^2\}$, имеющие одинаковый внешний алфавит $A = \{a_0,\ a_1,\ \ldots,\ a_n\}$. Пусть этим машинам соответствуют словарные функции f_1 и f_2. Существует машина Тьюринга с внешним алфавитом A, словарная функция f которой удовлетворяет условию $f(X) \simeq f_2(f_1(X))$. Такую машину можно построить, используя стандартную процедуру объединения машин T_1 и T_2. Выберем какое-либо расширение $\{q_{m_1+1}^1,\ \ldots,\ q_{m_1+m_2}^1\}$ внутреннего алфавита Q_1 машины T_1. В программе P_1 машины T_1 заменим все вхождения символа q_0^1 символом $q_{m_1+1}^1$, а в программе P_2 машины T_2 сделаем следующие замены: $q_0^2 \to q_0^1$, $q_j^2 \to q_{m_1+j}^1$, $j = \overline{1,\ m_2}$. После этого измененные программы объединим. Получим программу машины T, которую и назовем ***композицией машин Тьюринга*** T_1 и T_2.

Замечание 8.1. Поскольку ключевой целью машины Тьюринга является вычисление некоторой словарной функции, следовало бы ввести отношение эквивалентности между машинами, считая эквивалентными те, которые вычисляют одну и ту же словарную функцию. При этом возможны два уровня эквивалентности, как и в нормальных алгорифмах. Мы этого делать не будем, отметив однако, что композиция двух машин может осуществляться разными машинами.

Довольно трудно устроить соединение машин Тьюринга, так как механизм их работы заметно отличается от механизма работы нормальных алгорифмов. В первую очередь трудно реализовать вставку символа в слово (нормальный алгорифм это реализует легко). Надо сдвигать все символы вправо, а потом на освободившееся место записывать нужный символ.

В то же время разветвление машин сделать гораздо проще: никаких специальных управляющих алгоритмов не нужно, для выбора разных действий достаточно использовать разные внутренние состояния. Пусть даны три (замкнутые) машины Тьюринга T_1, T_2, T_3 с общим внешним алфавитом A и внутренними алфавитами $Q_s = \{q_0^s,\ q_1^s,\ \ldots,\ q_{m_s}^s\}$, $s = 1, 2, 3$ (полагаем, что три алфавита не имеют общих символов). Выделим в машине T_1 два внутренних состояния q_i^1 и q_j^1, $i,\ j \neq 0$. Объединим программы трех машин, предварительно выполнив в них следующие преобразования. В программе машины T_1 удаляем все команды, у которых в левой части указано состояние q_i^1 или q_j^1. В машине T_2 меняем внутренние состояния q_0^2 на q_0^1, q_1^2 на q_i^1. Аналогично в машине T_3 меняем внутренние состояния q_0^3 на q_0^1, q_1^3 на q_j^1. После этого объединяем три программы. Результирующая машина будет работать так. Стартовое состояние q_1^1 начинает работу машины T_1. Если будет достигнуто состояние q_0^1, машина оста-

новится, так что ни T_2, ни T_3 не будут использованы. Если будет достигнуто состояние q_i^1, начнет работу машина T_2, для которой это состояние стало стартовым. Завершение работы машины T_2 приведет к состоянию q_0^1, являющемуся заключительным. Если будет достигнуто состояние q_j^1, то далее будет работать машина T_3, которая остановит работу в состоянии q_0^1.

По этой же схеме легко устроить разветвление на большее число вариантов.

Замечание 8.2. И в композиции, и в разветвлении предполагается, что рассматриваемые алгоритмы заканчивают работу в стандартном положении, когда читающая головка обозревает первый символ слова. Если это условие не выполняется, то при передаче управления от одной машины к другой нужно вставлять специальную машину, которая не меняет данных, а лишь передвигает читающую головку в нужное положение.

Важное обстоятельство состоит в том, что идентифицировать в процессе работы стартовый символ не так-то просто. Если слово не содержит пустых символов, то это можно сделать движением влево до пустого символа, правда, это приведет к смещению за левую границу слова. Еще один вариант: использовать алфавит двойников. Тогда, заменив в самом начале первый символ слова его двойником, мы легко сможем идентифицировать его в дальнейшем.

Как устроить повторение машины Тьюринга по типу повторения нормальных алгорифмов? Например, так. Рассмотрим в качестве алгоритма T_2 специальный алгоритм, который не изменяет слова на ленте, а лишь формирует состояние q_0, которое возникает при обозревании первого символа слова. Изменим состояние этой машины с q_0 на q_1. Тогда при достижении состояния q_1 запускается машина T_2, которая организует повторный запуск машины T_1.

8.3. Эквивалентность машин Тьюринга и нормальных алгорифмов

Форма записи уже подсказывает, что программу машины Тьюринга легко трасформировать в схему нормального алгорифма. Действительно, всего пять сценариев одного такта машины Тьюринга: по два со смещением читающей головки и один без смещения. Выберем в качестве алфавита для нормального алгорифма внешний алфавит машины Тьюринга, а символы внутреннего состояния будем рассматривать как специальные символы (считаем, что машина Тьюринга замкнута). Заменим команду вида $q_ia_j \to q_la_kL$ двумя подстановками $\xi q_ia_j \to q_l\xi a_k$, $q_ia_j \to q_la_0a_k$, команду вида $q_ia_j \to q_la_kR$ — подстановками $q_ia_j\xi \to a_kq_l\xi$, $q_ia_j \to a_kq_la_0$, а команду вида $q_ia_j \to q_la_kC$ — подстановкой $q_ia_j \to q_la_k$. В конец записываем подстановку $\Lambda \to q_1$, осуществляющую запуск нормального алгорифма. Перед командой запуска поставим терминальную подстановку $q_0 \to\cdot\, q_0$. Полученный нормальный алгорифм выполнит те же преобразования, что и машина Тьюринга, поскольку в текущем слове только один специальный символ, причем этот символ не является в слове последним. Стартовая подстановка в конце схемы формирует начальную конфигурацию. Следовательно, в результате работы сконструированного нормального алгорифма мы получим конечную конфигурацию машины Тьюринга. Чтобы получить требуемый результат, необходимо удалить пустые символы в начале и конце слова, а также символ q_0 конечного состояния. Для этого достаточно взять композицию построенного нормального алгорифма с алгоритмом, например, с такой схемой:

$$\begin{cases} \xi q_0 \to q_0\xi, \\ q_0a_0 \to q_0, \\ q_0 \to r, \\ r\xi \to \xi r, \\ a_0r \to r, \\ r \to\cdot\, \Lambda. \end{cases}$$

Из приведенных рассуждений получаем следующий вывод.

Теорема 8.1. Любая словарная функция, вычислимая по Тьюрингу, вычислима и по Маркову. #

Доказанная теорема по существу означает, что возможности нормальных алгорифмов по крайней мере не хуже возможностей машин Тьюринга. Дополнительная мощь нормальных алгорифмов связана с нелокальным характером преобразования слова. Чтобы эту энергию ввести в управляемое русло, мы используем расширение алфавита — так называемые спецсимволы. Эти символы сродни символам, обозначающим внутренние состояния машины Тьюринга. И, в общем-то, ясно, как нормальный алгоритм с такими спецсимволами трансформировать в программу машины Тьюринга. Однако, во-первых, не все нормальные алгорифмы используют спецсимволы, а во-вторых, нормальный алгорифм может использовать одновременно несколько спецсимволов. В этом случае трансформировать нормальный алгоритм в машину Тьюринга сложнее. Вопрос, всегда ли такая трансформация возможна?

Теорема 8.2. Любая словарная функция, вычислимая по Маркову, вычислима и по Тьюрингу.

◀ Сумасшедшая идея — преобразовать в машину Тьюринга универсальный нормальный алгорифм. Но, во-первых, в универсальном алгорифме одновременно работают несколько спецсимволов, которые непосредственно интерпретировать как состояния машины не удастся, а во-вторых тогда надо составить еще машину, которая приписывает к слову изображение нормального алгорифма. Не факт, что это проще.

Можно сделать так. В качестве алфавита конструируемой машины Тьюринга рассматриваем алфавит нормального алгорифма с одним дополнительным символом, который будет играть роль пустого символа. Для каждой подстановки $B_i \to * C_i$ составляем машину Тьюринга T_i, которая ее реализует. Далее строим каскадное сочетание этих машин согласно схеме на рис. 8.1.

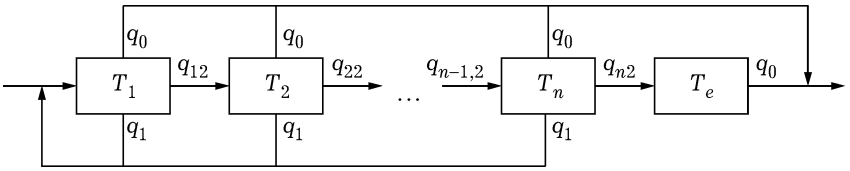


Рис. 8.1

Все машины T_i, $i = \overline{1,\, n}$, используют два общих состояния: q_0 — завершение работы всего комплекса (его формируют машины, реализующие терминальные подстановки, при успешной реализации такой подстановки и заключительная машина, если подходящей подстановки не найдено)); q_1 — повторение работы всего комплекса (его реализуют машины нетерминальных подстановок, причем каждая машина, завершая работу в этом состоянии, обеспечивает положение головки в начале слова). Остальные состояния машин T_i должны быть разными, и мы их будем обозначать $q_{i,1}$, $q_{i,2}$ и т.д. Состояние $q_{i,2}$ должно быть заключительным в случае нереализованной подстановки, состояние $q_{i,1}$ начальным, причем $q_{i,1} = q_{i-1,2}$, $i = \overline{2,\, n}$, а $q_{1,1} = q_1$, т.е. начальное состояние каждой машины, кроме первой, должно быть заключительным состоянием предыдущей, причем тем, которое указывает на неуспешную попытку реализации подстановки. При такой компоновке объединение всех машин даст машину Тьюринга, которая реализует схему нормального алгорифма.

Машину Тьюринга T_i, реализующую подстановку $B_i \to C_i$, можно реализовать как разветвление, построенное из двух составляющих машин:

• машина T_{i1} ищет первое вхождение подстроки B_i; если вхождение найдено, работа завершается в состоянии $q_{i,z}$ с положением читающей головки перед найденным вхождением; если вхождение не найдено, работа завершается в состоянии q_{i2} с положением читающей головки перед словом;

• машина T_{i2}, начиная работу в состоянии q_{iz}, заменяет найденное вхождение слова B_i словом C_i и заканчивает работу в состоянии q_1 (или q_0 в случае терминальной подстановки) и с положением читающей головки перед словом.

Пусть $B_i = b_1 b_2 \dots b_r$, $C_i = c_1 c_2 \dots c_s$. Машину T_{i1}, учитывая конкретный характер подстановки, можно реализовать следующим образом:

$$\begin{cases} q_{i1,1}a_0 \to q_{i1,r+4}a_0 L, & \text{(достигли конца слова, } B_i \text{ не нашли)} \\ q_{i1,1}b_1 \to q_{i1,3}uR, & \text{(начало посимвольного сравнения)} \\ q_{i1,1}\xi \to q_{i1,1}\xi R, \quad \xi \in A \setminus \{a_0,\ b_1\}, \\ q_{i1,3}b_2 \to q_{i1,4}b_2 R, \\ q_{i1,3}\xi \to q_{i1,r+3}\xi, \quad \xi \in A \setminus \{b_2\}, \\ \dots\dots\dots\dots\dots\dots \\ q_{i1,r+1}b_r \to q_{i1,r+2}b_r R, & \text{(фрагмент найден)} \\ q_{i1,r+1}\xi \to q_{i1,r+3}\xi, \quad \xi \in A \setminus \{b_r\}, \\ q_{i1,r+2}\xi \to q_{i1,r+2}\xi L, \\ q_{i1,r+2}u \to q_{i,z}b_1, & \text{(успешное завершение)} \\ q_{i1,r+3}\xi \to q_{i1,r+3}\xi L, \quad \xi \in A \setminus \{a_0\}, \\ q_{i1,r+3}u \to q_{i1,1}b_1 R, \\ q_{i1,r+4}a_0 \to q_{i,2}a_0 R, & \text{(неуспешное завершение, достигли начала слова)} \\ q_{i1,r+4}\xi \to q_{i1,r+4}\xi L, \quad \xi \in A \setminus \{a_0\} & \text{(неуспешное завершение)} \end{cases}$$

Этот алгоритм завершает работу в состоянии $q_{i,z}$, если сравнение удачное, причем положение читающей головки перед найденным фрагментом. Если вхождение неудачное, то завершаем работу в состоянии $q_{i,2}$, причем положение читающей головки в начале данного слова.

Машина T_{i2} начинает работу в состоянии $q_{i2,1} = q_{i,z}$. Она имеет три варианта конструкции в зависимости от соотношения длин r и s слов B_i и C_i. Если $r = s$, то машина T_{i2} реализуется без каких-либо проблем:

$$\begin{cases} q_{i2,1}b_1 \to q_{i2,2}c_1 R, \\ q_{i2,2}b_2 \to q_{i2,3}c_2 R, \\ \dots\dots\dots\dots\dots\dots \\ q_{i2,r}b_r \to q_{i2,r+1}c_r R, \\ q_{i2,r+1}\xi \to q_{i2,r+1}\xi L, \quad \xi \in A \setminus \{a_0\}, \\ q_{i2,r+1}a_0 \to q_{i2,0}a_0 R. \end{cases}$$

Эта машина завершает работу в состоянии $q_{i2,0}$, которое совпадает или с q_0 (терминальная подстановка), или с q_1 (нетерминальная), причем читающая головка позиционируется в начале слова.

Если $r > s$, то машину T_{i2} можно реализовать как композицию трех: первая машина T_{i21} заменяет B_i на C_i, заполняя лишние $r - s$ позиций пустым символом; вторая машина T_{i22} перемещает пустые символы в конец слова; третья T_{i23} позиционирует читающую головку в начале слова. Машина T_{i21} незначительно отличается от машины T_{i2} при $r = s$:

$$\begin{cases} q_{i21,1}b_1 \to q_{i21,2}c_1 R, \\ q_{i21,2}b_2 \to q_{i21,3}c_2 R, \\ \dots\dots\dots\dots\dots\dots \\ q_{i21,s}b_s \to q_{i21,s+1}c_s R, \\ q_{i21,s+1}b_{s+1} \to q_{i21,s+2}a_0 R, \\ q_{i21,s+2}b_{s+1} \to q_{i21,s+3}a_0 R, \\ \dots\dots\dots\dots\dots\dots \\ q_{i21,r}b_r \to q_{i21,0}a_0 R, \end{cases}$$

Машина завершает работу в состоянии $q_{i21,0}$ и с положением читающей головки вслед за полем из пустых символов. Машину T_{i22} можно реализовать как композицию $r-s$ машин T_{i22j}, каждая из которых перегоняет последний пустой символ поля в конец слова. Такую машину можно реализовать следующим образом:

$$\begin{cases} q_{i22j,1}\xi \to q_{i22j,\xi}a_0L, \quad \xi \in A \setminus \{a_0\}, \\ q_{i22j,\xi}a_0 \to q_{i22j,2}\xi R, \quad \xi \in A \setminus \{a_0\}, \\ q_{i22j,2}a_0 \to q_{i22j,1}a_0R, \\ q_{i22j,1}a_0 \to q_{i22j,3}a_0L, \\ q_{i22j,3}a_0 \to q_{i22j,4}a_0L, \\ q_{i22j,4}\xi \to q_{i22j,4}\xi L, \quad \xi \in A \setminus \{a_0\}, \\ q_{i22j,4}a_0 \to q_{i22j,0}a_0R. \end{cases}$$

Эта машина завершает работу в состоянии $q_{i22j,0}$ с положением головки за последним пустым символом или в начале слова.

Машину T_{i23} можно реализовать следующим образом:

$$\begin{cases} q_{i23,1}a_0 \to q_{i23,1}a_0R, \\ q_{i23,1}\xi \to q_{i23,1}\xi L, \quad \xi \in A \setminus \{a_0\}. \end{cases}$$

Если $r < s$ то для вставки символов c_{r+1}, c_{r+2} надо раздвигать слово, освобождая место для вставляемого символа. Машину T_{i2} реализуем, как и в случае $r > s$, в виде композиции машин T_{i21}, T_{i22} и T_{i23}, первая из которых меняет символы b_1, …, b_r, а вторая вставляет оставшиеся символы c_{r+1}, …, c_s, раздвигая слово, а третья перегоняет читающую головку в начало слова. Машина T_{i21} в данном случае реализуется как незначительная модификация машины T_{i2} для случая $r = s$:

$$\begin{cases} q_{i21,1}b_1 \to q_{i21,2}c_1R, \\ q_{i21,2}b_2 \to q_{i21,3}c_2R, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ q_{i21,r}b_r \to q_{i21,0}c_rR. \end{cases}$$

Машину T_{i22} реализуем как композицию $s - r$ машин T_{i22j}, каждая из которых вставляет один символ, завершая работу с головкой вслед за вставленным символом. Машину T_{i22j} можно реализовать следующим образом:

$$\begin{cases} q_{i22j,1}\xi \to q_{i22j,\xi}a_0R, \quad \xi \in A \setminus \{a_0\}, \\ q_{i22j,\xi}\eta \to q_{i22j,\eta}\xi R, \quad \xi, \eta \in A \setminus \{a_0\}, \\ q_{i22j,\xi}a_0 \to q_{i22j,3}\xi L, \quad \xi \in A \setminus \{a_0\}, \\ q_{i22j,3}\xi \to q_{i22j,3}\xi L, \quad \xi \in A \setminus \{a_0\}, \\ q_{i22j,3}a_0 \to q_{i22j,0}c_{r+j}R. \end{cases}$$

Машина T_{i23} реализуется так же, как и в случае $r > s$.

Итак, все машины T_i в каскадном сочетании (см. рис. 8.1) описаны. Осталось описать машину T_e. Эта машина представляет собой „заглушку“: она просто переводит состояние q_{n2} в состояние q_0.

В совокупности рассмотренный комплекс машин T_i реализует нормальный алгорифм с заданной схемой. ►

8.4. Обобщения машин Тьюринга

Доказанная эквивалентность машин Тьюринга нормальным алгорифмам подчеркивает ту мысль, что машины Тьюринга позволяют реализовать любой мыслимый алгоритм или, по-другому, вычислить любую эффективно вычислимую словарную функцию. Однако на практике, как и в языках программирования, для решения конкретных задач удобнее использовать не один-единственный универсальный язык программирования, а целое множество таких языков, из которого можно выбрать наиболее подходящий. Так и в теории алгоритмов: предложено много различных вариаций абстрактных машин, да и иных концепций, которые предназначены для формализации понятия алгоритма.

Уже отмечалось, что простейшая машина Тьюринга имеет несколько вариаций. Однако эти вариации схожи по конструкции и виду элементарных операций. Есть, однако, конструкции абстрактных машин, которые получены усложнением (или, так сказать, усилением) первоначальной машины Тьюринга. Рассмотрим некоторые варианты.

Однолентоточная машина с входом и выходом — это набор $\{I,\ O,\ W,\ Q,\ P\}$ из пяти множеств, в котором I — входной алфавит, O — выходной алфавит, W — рабочий алфавит, Q — множество состояний, P — программа. Команды программы имеют вид $q_i x_j w_k \to q_r w_s y_t M$, где:

$q_i \in Q$ — исходное состояние на данном такте;
$x_j \in I$ — входной символ (поступает с входной ленты);
$w_k \in W$ — обозреваемый символ рабочей ленты;
$q_r \in Q$ — результирующее состояние на данном такте;
$w_s \in W$ — символ, заменяющий обозреваемый символ рабочей ленты;
$y_t \in O$ — выходной символ (выводится на выходную ленту);
$M \in \{L,\ R,\ C\}$ — указатель движения головки на рабочей ленте.

На каждом такте с входной ленты поступает очередной символ x_j. Машина в соответствии с этим символом, внутренним состоянием q_i и обозреваемым символом w_k на рабочей ленте выполняет одновременно три действия: меняет внутреннее состояние на q_r, меняет обозреваемый символ на рабочей ленте на символ w_s и выводит на выходную ленту символ y_t. В начальном состоянии машина находится во внутреннем состоянии q_1, рабочая лента пуста. Завершение работы машины можно устроить по-разному. Например, как и в случае машины без входа и выхода, останов происходит при достижении внутреннего состояния q_0 (завершающего). Тогда исходное слово — последовательность использованных входных символов, результирующее слово — последовательность выходных символов. Как входные символы, так и выходные могут быть пустыми и в итоге не учитываются (нужны, чтобы сделать паузу на входе и выходе). Входная и выходная ленты бесконечны справа. Другой вариант, когда внутреннее завершающее состояние отсутствует, а машина останавливается, когда на выходной ленте появляется один из некоторого набора специальных символов. В этом случае нужен некий символ #, фиксирующий конец входного слова. Когда входное слово завершено, на вход подается символ # и с этого момента появление завершающего символа на выходе останавливает машину.

Многоголовочная машина Тьюринга имеет одну бесконечную двустороннюю ленту и некоторое количество k головок, обозревающих эту ленту. Положения головок независимы, команды такой машины имеют вид

$$q_i a_{j_1} a_{j_2} \dots a_{j_k} \to q_r a_{s_1} a_{s_2} \dots a_{s_k} M_1 M_2 \dots M_k. \tag{8.1}$$

При внутреннем состоянии q_i и обозреваемых символах a_{j_l}, $l = \overline{1,\ k}$, она переходит в состояние q_r, каждый l-й обозреваемый символ заменяется на a_{s_l}, а l-я головка смещается согласно символу $M_l \in \{L,\ R,\ C\}$. Независимое расположение головок на ленте может привести к ситуации, когда несколько головок обозревают один символ. В этом случае действует правило приоритета: за изменение символа отвечает головка с наименьшим номером (можем считать, что каждая

головка сделает свое дело, но первой пишет головка с наибольшим номером). Останов такой машины может происходить либо по отсутствию команды, либо по заключительному состоянию (что, в общем-то, одно и то же).

Из этих монстров наиболее распространены ***многоленточные машины Тьюринга***, которые используют в задачах оценки сложности алгоритмов. Так называют абстрактную машину, имеющую несколько лент. На каждой ленте имеется читающая головка, все головки перемещаются независимо. Команды машины Тьюринга имеют тот же вид (8.1), но теперь головки находятся на разных лентах. Конфигурацией такой машины является набор слов $X_l q_i Y_l$, $l = \overline{1, k}$, где X_l, Y_l, — слова во внешнем алфавите, одно из которых не пусто. Начальная конфигурация состоит из слов вида $q_1 Y_l$ (т.е. $X_l = \Lambda$). Машина завершает работу, как и одноленточная машина, по отсутствию команды или по заключительному состоянию.

Многоленточные машины удобны для реализации словарных функций нескольких переменных. Каждый аргумент записывается на своей ленте. Одна лента выделена под результат. Некоторые ленты могут использоваться для промежуточных вычислений.

Обобщения машины Тьюринга потенциально имеют больше возможностей, чем стандартная машина Тьюринга. Так одноленточная машина может быть реализована на многоленточной, например, так: на одной ленте происходят вычисления, а остальные не изменяются. Чтобы обеспечить соглашение о расположении исходного слова и результата на разных лентах, достаточно реализовать предварительное копирование исходного слова на результирующую ленту. Однако в действительности расширение возможностей не приводит к расширению множества вычислимых словарных функций. Можно придумать разные варианты эмуляции многоленточной машины Тьюринга на одноленточной. Один из них такой. Исходные слова на лентах записываем как γ-систему, добавляя в слова специальный символ μ, указывающий положение головки на этом слове. Одноленточная машина пробегает слово, выбирая обозреваемые символы, которые идут вслед за μ, и формируя соответствующее внутреннее состояние. Затем обратным ходом машина реализует требуемые изменения. Разумеется, такая эмуляция требует большой внутренней памяти (множества внутренних состояний) и большой длины программы. Ясно, что в определенных ситуациях при решении задач на существование алгоритмов многоленточные машины использовать проще.

9. РЕКУРСИВНЫЕ ФУНКЦИИ

Рекурсивные функции — некоторый класс функций одного или нескольких натуральных аргументов, которые можно получить из некоторого исходного набора функций с помощью определенных операций. Понятие „рекурсивная функция“ — удобное математическое описание класса вычислимых (в том или ином смысле) функций, поскольку дается в привычных в математике терминах операций. В то же время процедура построения рекурсивной функции с помощью последовательности операций сродни построению формулы в том или ином формальном исчислении. Это родство выражают словом „конструктивный“. Конструктивный объект отличается как раз тем, что известно, как он получен в результате конечной процедуры применения простейших операций. Поэтому с точки зрения формализации понятия „алгоритм“ рекурсивные функции ничуть не хуже абстрактных машин или нормальных алгорифмов.

Хотя нормальные алгорифмы и машины Тьюринга строились в произвольном алфавите, с принципиальной точки зрения выбор алфавита не является существенным, а любой алгоритм можно трансформировать в алгоритм с двухбуквенным алфавитом (это строго проверено в рамках нормальных алгорифмов). Это и понятно: множество слов в данном алфавите счетно, так что все слова можно перенумеровать, а любую словарную функцию рассматривать как функцию натурального аргумента, которая связывает не сами слова, а их номера. Это было понятно давно, и с самого начала теория алгоритмов строилась на базе двухбуквенного алфавита. В частности, в рамках рекурсивных функций рассматриваются не произвольные словарные функции, а словарные функции в двухбуквенном алфавите, которые легко интерпретируются как функции натурального аргумента.

Различают ***рекурсивные функции***, определенные для любых комбинаций значений аргументов, (т.е. область определения есть $\mathbb{N}^n$), и ***частично рекурсивные функции***, область определения которых составляет лишь часть множества $\mathbb{N}^n$.

Начнем с множества исходных (простейших) рекурсивных функций.

9.1. Примитивно рекурсивные функции

Выделим простейшие функции натурального аргумента:

1) инкремент $f(x) = x^+ = x + 1$;

2) константа нуль $0(x) = 0$;

3) проективная функция $I_m^n(x_1, x_2, \ldots, x_n) = x_m$, $1 \leqslant m \leqslant n$.

Теперь определим простейшие операции над функциями:

1° суперпозиция $f^m(g_1^n(x_1, x_2, \ldots, x_n), \ldots, g_m^n(x_1, x_2, \ldots, x_n))$ (здесь верхний индекс в обозначении функции указывает на ее арность);

2° примитивная рекурсия, которая из функций f^n и g^{n+2} строит новую функцию $h^{n+1} = R(f^n, g^{n+2})$ в соответствии с равенствами:

$$h^{n+1}(x_1, x_2, \ldots, x_n, 0) = f^n(x_1, x_2, \ldots, x_n),$$
$$h^{n+1}(x_1, x_2, \ldots, x_n, y+1) = g^{n+2}(x_1, x_2, \ldots, x_n, y, h^{n+1}(x_1, x_2, \ldots, x_n, y)).$$

Мы можем дать индуктивное определение примитивно рекурсивной функции.

1. Простейшие функции примитивно рекурсивные.

2. Если функции f^m, g_1^n, g_2^n, …, g_m^n примитивно рекурсивные, то их суперпозиция — примитивно рекурсивная функция.

3. Если функции f^n и g^{n+2} примитивно рекурсивные, то функция $R(f^n, g^{n+2})$ примитивно рекурсивная.

Отметим, что примитивно рекурсивных функций счетное множество. Действительно, простейших функций счетное множество (за счет проективных функций разной арности). С помощью одной операции из простейших мы получаем опять счетное множество. Следовательно, функций, получаемых двумя операциями счетное множество. В результате имеем счетное семейство счетных множеств, которое, как известно, счетно.

В то же время всех функций натурального аргумента континуум. Множество $\mathbb{N}^k$ при любом k является счетным. Множество всех отображений $\mathbb{N}^k$ в множество $\{0,\ 1\}$ есть булеан счетного множества, т.е. континуум. А множество всех функций k аргументов можно представить как множество подмножеств в $\mathbb{N}^{k+1}$ (графиков функций). Значит, не более чем континуум.

Из этих соображений вытекает, что подавляющее большинство функций натурального аргумента не является примитивно рекурсивным. Однако конкретный пример функции, не являющейся примитивно рекурсивной, привести не просто. Здесь та же ситуация, что и с функциями действительного переменного: весь ассортимент функций, которыми мы реально пользуемся — очень малая часть всего многообразия функций.

Замечание 9.1. Проективные функции позволяют из данной примитивно рекурсивной функции получать новые функции простой перестановкой или дублированием аргументов. Действительно, если $f(x_1, x_2)$ примитивно рекурсивна, то примитивно рекурсивными будут $f(x_2, x_1)$, $f(x, x)$, $f(x, 0)$, поскольку

$$\begin{aligned} f(x_2, x_1) &= f(I_2^2(x_1, x_2), I_1^2(x_1, x_2)),\\ f(x, x) &= f(I_1^1(x), I_1^1(x)),\\ f(x, 0) &= f(I_1^1(x), 0(x)). \end{aligned}$$

По этой же причине если функция $f(x_1, x_2, \ldots, x_n)$ является примитивно рекурсивной, то и функция $g(x_1, x_2, \ldots, x_n, x_{n+1}) = f(x_1, x_2, \ldots, x_n)$, полученная добавлением фиктивного аргумента, тоже примитивно рекурсивна:

$$g(x_1, x_2, \ldots, x_n, x_{n+1}) = f(I_1^{n+1}(x_1, \ldots, x_{n+1}), I_2^{n+1}(x_1, \ldots, x_{n+1}), \ldots, I_n^{n+1}(x_1, \ldots, x_{n+1})).$$

Эти правила можно обобщить, введя следующее понятие. Говорим, что функция h^m получена из функций f^m, $g_1^{k_1}$, …, $g_m^{k_m}$ с помощью подстановки, если

$$h^m(x_1, x_2, \ldots, x_n) = f^m(g_1^{k_1}(y_1^1, \ldots, y_{k_1}^1), \ldots, g_m^{k_m}(y_1^m, \ldots, y_{k_m}^m)),$$

где y_s^r обозначает одну из переменных x_1, x_2, …, x_n. Как сказано, подстановка может быть сведена к суперпозиции с помощью соответствующих проективных функций.

Пример 9.1. Все постоянные функции примитивно рекурсивные. Действительно, функция $0(x) = 0$ простейшая, функция $f(x) = 1$ есть суперпозия $0(x)^+$, а значит, простейшая. Далее применяя суперпозицию уже построенной функции $f(x) = k$ с функцией-инкрементом, получаем $f(x) = k + 1$. Постоянные функции от нескольких переменных можно рассматривать как функции, полученные из постоянных функций одного переменного добавлением фиктивных аргументов. Отметим, что установленное свойство позволяет формально интерпретировать постоянные функции одного переменного как функции, полученные из „нульарных“ функций добавлением фиктивного аргумента.

Замечание 9.2. Формально с помощью примитивной рекурсии мы можем получить лишь функцию, имеющую не менее двух аргументов (поскольку функция f имеет хотя бы один аргумент и, значит, $n \geqslant 1$). Однако использование постоянных функций позволяет построить следующую рекурсию:

$$\begin{aligned} h^1(0) &= k,\\ h^1(y^+) &= g^2(y, h^1(y)). \end{aligned}$$

Здесь k — произвольное натуральное число. Действительно, рассмотрим постоянную функцию $f^1(\xi) = k$ и функцию $g^3(\xi, x_1, x_2) = g^2(x_1, x_2)$. С помощью f^1 и g^3 строим функцию двух переменных:

$$h^2(\xi, 0) = f^1(\xi),$$
$$h^2(\xi, y^+) = g^3(\xi, y, h^2(\xi, y)).$$

После этого получаем нужную функцию: $h^1(y) = h^2(y, y)$ (на самом деле значение h^2 не зависит от первого аргумента). Приведенный пример наталкивает на мысль интерпретировать конкретные числа как нульарные функции, что позволяет ввести рассмотренный пример в рамки определения примитивной рекурсии.

Пример 9.2. Функция $s(x, y) = x + y$ примитивно рекурсивная. Действительно, из определения суммы в рамках формальной арифметики $x + 0 = x$, $x + y^+ = (x + y)^+$. Из этих равенств заключаем, что $s(x, y)$ есть примитивная рекурсия:

$$s(x, 0) = I_1^2(x, 0), \quad s(x, y^+) = s(x, y)^+ = I_3^3(0, 0, s(x, y)^+).$$

Вторая формула представляет $s(x, y^+)$ как суперпозицию функций $g^3(x, y, z) = z^+ = I_3^3(x, y, z^+)$, 0, 0 и $s(x, y)$.

Пример 9.3. Функция $m(x, y) = xy$ также примитивно рекурсивная. Вспомним из формальной арифметики, что $m(x, 0) = 0$, $m(x, y^+) = m(x, y) + x = s(x, m(x, y))$. Последняя функция имеет два аргумента, а должно быть три. Вводим функцию $s^3(x, y, z) = s(x, z)$. Тогда $m(x, y^+) = s^3(x, y, m(x, y))$. Остается показать, что $s^3(x, y, z)$ является примитивно рекурсивной. Ее можно получить с помощью суперпозиции: $s^3(x, y, z) = s(I_1^3(x, y, z), I_3^3(x, y, z))$.

Теорема 9.1. Если $f^n(x_1, x_2, \ldots, x_n)$ примитивно рекурсивна, то функции

$$s^n(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{x_n} f^n(x_1, x_2, \ldots, x_{n-1}, i) \quad \text{и} \quad m^n(x_1, x_2, \ldots, x_n) = \prod_{i=0}^{x_n} f^n(x_1, x_2, \ldots, x_{n-1}, i)$$

примитивно рекурсивны.

◀ В данном случае

$$s^n(x_1, x_2, \ldots, x_{n-1}, 0) = f^n(x_1, x_2, \ldots, x_{n-1}, 0),$$
$$s^n(x_1, x_2, \ldots, x_{n-1}, k^+) = s^n(x_1, x_2, \ldots, x_{n-1}, k) + f^n(x_1, x_2, \ldots, x_{n-1}, k + 1).$$

Функция $f^n(x_1, x_2, \ldots, 0)$ примитивно рекурсивна как суперпозиция проекций и нулевой функции. Выражение $s^n(x_1, x_2, \ldots, x_{n-1}, k) + f^n(x_1, x_2, \ldots, x_{n-1}, k + 1)$ можно представить в виде $g^{n+1}(x_1, \ldots, x_{n-1}, k, s(x_1, x_2, \ldots, x_{n-1}, k))$, где

$$g^{n+1}(x_1, \ldots, x_{n-1}, k, m) = f^n(x_1, x_2, \ldots, x_{n-1}, k + 1) + m.$$

Последняя функция, очевидно, примитивно рекурсивная.

Доказательство в случае умножения аналогично. ▶

Замечание 9.3. Доказанную теорему можно немного обобщить, а именно: в условиях теоремы для любого k примитивно рекурсивными являются функции

$$s^n(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{x_n - k} f^n(x_1, x_2, \ldots, x_{n-1}, i) \quad \text{и} \quad m^n(x_1, x_2, \ldots, x_n) = \prod_{i=0}^{x_n - k} f^n(x_1, x_2, \ldots, x_{n-1}, i),$$

где предполагается, что сумма нулевая, а произведение равно единице, если нижний предел суммирования больше верхнего. Возьмем, например, сумму. Очевидно, что

$$s^n(x_1, x_2, \ldots, x_n) = \sum_{j=k}^{x_n} f^n(x_1, x_2, \ldots, i - k).$$

Положив

$$g(x_1, x_2, \ldots, x_{n-1}, y) = \begin{cases} f^n(x_1, x_2, \ldots, y-k), & y \geqslant k; \\ 0, & 0 \leqslant y < k, \end{cases}$$

получим

$$s^n(x_1, x_2, \ldots, x_n) = \sum_{j=0}^{x_n} g(x_1, x_2, \ldots, i).$$

Согласно теореме 9.1 эта функция примитивно рекурсивна, если функция g примитивно рекурсивна. Доказать примитивную рекурсивность g — несложное упражнение.

Теорема 9.2. Пусть g_1^n, g_2^n, h^n — примитивно рекурсивные функции. Тогда примитивно рекурсивной является функция

$$f^n(x_1, x_2, \ldots, x_n) = \begin{cases} g_1^n(x_1, x_2, \ldots, x_n), & h^n(x_1, x_2, \ldots, x_n) \neq 0; \\ g_2^n(x_1, x_2, \ldots, x_n), & h^n(x_1, x_2, \ldots, x_n) = 0. \end{cases}$$

◀ Доказательство построено на представлении f^n в виде композиции

$$f^n(x_1, x_2, \ldots, x_n) = g_1^n(x_1, x_2, \ldots, x_n) \cdot \operatorname{sg}(h^n(x_1, x_2, \ldots, x_n)) + g_2^n(x_1, x_2, \ldots, x_n) \cdot \overline{\operatorname{sg}}(h^n(x_1, x_2, \ldots, x_n)),$$

в которой функция $\operatorname{sg}(x)$ равна 1 при $x \neq 0$ и 0 при $x = 0$, а функция $\overline{\operatorname{sg}}(x)$ представляет ее дополнение: $\overline{\operatorname{sg}}(x) = 1 - \operatorname{sg}(x)$. Первую можно построить с помощью примитивной рекурсии: $\operatorname{sg}(0) = 0$, $\operatorname{sg}(x^+) = 1$, где постоянную 1 можно рассматривать как функцию двух аргументов x и $\operatorname{sg}(x)$. Аналогично можно построить и дополняющую функцию $\overline{\operatorname{sg}}(x)$: $\overline{\operatorname{sg}}(0) = 0^+$, $\overline{\operatorname{sg}}(x^+) = 0$. ▶

Замечание 9.4. Функции $\operatorname{sg}(x)$ и $\overline{\operatorname{sg}}(x)$ можно получить с помощью ***усеченной разности*** $r(x, y) = x \dot{-} y = \max\{x - y,\ 0\}$. Действительно, $\overline{\operatorname{sg}}(x) = 1 \dot{-} x$, $\operatorname{sg}(x) = \overline{\operatorname{sg}}(\overline{\operatorname{sg}}(x))$. Докажем, что усеченная разность разность есть примитивно рекурсивная функция.

Имеем $x \dot{-} 0 = x$ — примитивно рекурсивная функция. Покажем, что $\varphi(x) = x \dot{-} 1$ — тоже примитивно рекурсивная функция. Опять примитивная рекурсия: $\varphi(0) = 0$ — примитивно рекурсивна (формально), $\varphi(y^+) = y = I_1^2(y, \varphi(y))$. Теперь, возвращаясь к $r(x, y) = x \dot{-} y$, можем записать $r(x, y^+) = x \dot{-} y^+ = (x \dot{-} y) \dot{-} 1 = \varphi(r(x, y)) = I_3^3(x, y, \varphi(r(x, y)))$. Полагая $g^3(x, y, z) = I_3^3(x, y, \varphi(z))$, получим $r(x, y^+) = g^3(x, y, r(x, y))$, что соответствует определению примитивной рекурсии. #

9.2. Предикаты, простые числа и возвратная рекурсия

Рассмотренные функции $x + y$, xy и т.д. получаются с помощью алгебраических операций. Таким способом однако нельзя получить другие используемые на практике функции, например функция $p(i)$, значением которой является $(i + 1)$-е по порядку простое число. Однако и такие функции являются эффективно вычислимыми и естественен вопрос, являются ли они примитивно рекурсивными.

Выделим класс целочисленных функций, принимающих лишь значения 0, 1, т.е. отображений множества $\mathbb{N}^k$ в множество $\{0, 1\}$. Такие функции являются истинностными функциями предикатов, определенных на множестве натуральных чисел. Не разделяя предикаты и их истинностные функции, будет говорить, что ***n-местный предикат*** — это функция от n натуральных аргументов, принимающая два значения 0, 1. В этом контексте можно говорить о ***примитивно рекурсивных предикатах***. Выясним, как себя ведет условие примитивной рекурсивности при выполнении обычных логических операций.

Теорема 9.3. Если $R_1(x_1, \ldots, x_n)$ и $R_2(x_1, \ldots, x_n)$ — примитивно рекурсивные предикаты, то примитивно рекурсивными являются следующие предикаты:

$$R_1 \vee R_2, \quad R_1 \wedge R_2, \quad R_1 \to R_2, \quad R_1 \sim R_2, \quad \neg R_1.$$

◀ Достаточно выразить эти операции через уже рассмотренные:

$$x \vee y = \operatorname{sg}(x+y), \quad x \wedge y = xy, \quad \neg x = \overline{\operatorname{sg}}(x),$$
$$x \to y = \neg x \vee y = \overline{\operatorname{sg}}(x \dot{-} y), \quad x \sim y = \overline{\operatorname{sg}}((x \dot{-} y) + (y \dot{-} x)). \qquad \blacktriangleright$$

Предикаты можно получать как результат сравнения целочисленных функций. Если сравниваемые функции примитивно рекурсивны, то получаемый предикат тоже примитивно рекурсивен.

Теорема 9.4. Если функции $f(x_1, x_2, \ldots, x_n)$ и $g(x_1, x_2, \ldots, x_n)$ примитивно рекурсивны, то примитивно рекурсивными являются предикаты

$$f < g, \quad f \leqslant g, \quad f \geqslant g, \quad f < g, \quad f = g, \quad f \neq g.$$

◀ Все эти сравнения либо перестановкой двух функций, либо применением логических операций сводятся к одному сравнению $f > g$. Например, $f \geqslant g \equiv \neg(g > f)$, $(f \neq g) \equiv (f > g) \vee (g > f)$. Сравнение $f > g$ можно выразить через ранее рассмотренные функции: $(f > g) \equiv \operatorname{sg}(f \dot{-} g)$. ▶

С предикатами могут также выполняться кванторные операции.

Теорема 9.5. Если $R(x_1, x_2, \ldots, x_n)$ — примитивно рекурсивный предикат, то примитивно рекурсивны предикаты

$$R_\exists(x_1, x_2, \ldots, x_n) = \exists y(y \leqslant x_n \wedge R(x_1, x_2, \ldots, x_{n-1}, y)),$$
$$R_\forall(x_1, x_2, \ldots, x_n) = \forall y(y \leqslant x_n \to R(x_1, x_2, \ldots, x_{n-1}, y))$$

и функция

$$\mu(x_1, x_2, \ldots, x_n) = \min\{y\colon ((y < x_n) \wedge R(x_1, x_2, \ldots, y)) \vee (y = x_n)\}.$$

◀ Утверждение о предикатах вытекает из представлений

$$R_\exists(x_1, x_2, \ldots, x_n) = \operatorname{sg}\Big(\sum_{i=0}^{x_n} R(x_1, x_2, \ldots, x_{n-1}, i)\Big),$$
$$R_\forall(x_1, x_2, \ldots, x_n) = \prod_{i=0}^{x_n} R(x_1, x_2, \ldots, x_{n-1}, i)$$

и теоремы 9.1. Примитивная рекурсивность функции μ вытекает из представления

$$\mu(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{x_n}\Big((i \neq x_n) \wedge \prod_{j=0}^{i} \neg R(x_1, x_2, \ldots, x_{n-1}, j)\Big).$$

Здесь особенность — наличие переменного x_n под знаком суммы, что не предусматривается в теореме 9.1. Обойти это можно так. Функция

$$\mu_e(x_1, x_2, \ldots, x_n, y) = \sum_{i=0}^{y} f^{n+1}(x_1, x_2, \ldots, x_n, i) = \sum_{i=0}^{y}\Big((i \neq x_n) \wedge \prod_{j=0}^{i} \neg R(x_1, x_2, \ldots, x_{n-1}, j)\Big)$$

примитивно рекурсивна. Значит, и функция $\mu(x_1, x_2, \ldots, x_n) = \mu_e(x_1, x_2, \ldots, x_n, x_n)$ примитивно рекурсивна. ▶

Для кванторных операций будем использовать упрощенные обозначения:

$$\exists(y \leqslant x_n)\, R(x_1, x_2, \ldots, x_{n-1}, y), \quad \forall(y \leqslant x_n)\, R(x_1, x_2, \ldots, x_{n-1}, y).$$

С помощью этих базовых кванторных операций можно получить другие операции, также приводящие к примитивно рекурсивным предикатам, если исходные предикаты примитивно рекурсивны:

$$\nabla(y{<}x_n)\, R(x_1, x_2, \ldots, x_{n-1}, y), \quad \nabla(k{\leqslant}y{\leqslant}x_n)\, R(x_1, x_2, \ldots, x_{n-1}, y)$$

и другие, отличающиеся типом неравенства (здесь ∇ — один из двух кванторов). Аналогичное обозначение введем для операции минимизации μ:

$$\mu(y{\leqslant}x_n)R(x_1, x_2, \ldots, x_{n-1}, y) = \min\{y\colon (y < x_n \wedge R(x_1, x_2, \ldots, y)) \vee y = x_n\}.$$

Как и кванторные операции, операция минимизации допускает вариации с заменой неравенства на нестрогое или на двойное неравенство.

Предикаты позволяют ввести новые примитивно рекурсивные функции, например:

1) $x \vdots y$ — предикат „x делится на y“;

2) $\Pr(x)$ — предикат „число x простое“;

3) $\pi(i)$ — функция, значением которой является $(i+1)$-е (в порядке возрастания) простое число.

Предикаты $x \vdots y$ и $\Pr(x)$ можно записать с помощью логических и кванторных операций:

$$(x \vdots y) \equiv \exists(i{\leqslant}x)\,(x = iy), \quad \Pr(x) \equiv \neg\exists(i{<}x)((i > 1) \wedge (x \vdots i)).$$

Функцию, перечисляющую простые числа, можно задать с помощью примитивной рекурсии следующим образом:

$$\pi(0) = 2, \quad \pi(x^+) = \mu(\pi(x) < i \leqslant \pi(x)! + 1)\ \Pr(i).$$

Здесь использован элементарный факт, что число $\pi(x^+)$, т.е. первое простое число, следующее за $\pi(x)$, не превышает $\pi(x)! + 1$. Действительно, число $\pi(x)! + 1$ не делится ни на одно из простых чисел $\pi(1)$, …, $\pi(x)$. Значит, либо оно простое, либо у него есть простой делитель, отличающийся от указанных. Поэтому в интервале $[\pi(x) + 1,\ \pi(x)! + 1]$ есть простые числа. Взяв среди таких простых чисел наименьшее, получим $\pi(x^+)$.

В операторе μ, использованном при определении $\pi(x)$, задано сложное условие. Такое условие можно реализовать следующим образом. С помощью предиката $\Pr(x)$ строим функцию двух переменных $g(x, y) = \mu(i \leqslant y)(\Pr(i) \wedge (i > x))$, которая возвращает первое простое число на промежутке $x < i < y$ или число y, если на указанном промежутке нет простого числа. Тогда примитивную рекурсию для функции $\pi(x)$ можно определить так: $\pi(x^+) = g(\pi(x), \pi(x)! + 1)$. Остается убедиться в том, что $x!$ — примитивно рекурсивная функция.

Каждое натуральное (ненулевое) число x можно разложить на простые множители, т.е. представить в виде $x = p_1^{a_1} p_2^{a_2} \ldots p_k^{a_k}$. Такое разложение единственно, если простые множители p_i пронумерованы по возрастанию. Более того, такое разложение можно записать в виде бесконечного произведения

$$x = \pi(0)^{c_0}\pi(1)^{c_1} \ldots \pi(n)^{c_n} \ldots \tag{9.1}$$

в котором только конечное число показателей c_i отлично от нуля (ненулевые показатели соответствуют реальным простым множителям в разложении числа x). В результате каждому ненулевому числу поставлена в соответствие последовательность натуральных чисел, в которой только конечное число элементов ненулевое (финитная последовательность). Здесь возникают функции: c_0 как функция x, c_1 как функция x и т.д.

Введем обозначение $\mathrm{pw}_i(x)$ для коэффициента c_i в разложении (9.1) числа x. Полагаем $\mathrm{pw}_i(0) = 0$. Получаем для каждого i функцию pw_i, которая оказывается примитивно рекурсивной. Действительно,

$$\mathrm{pw}_i(x) = \mu(y{<}x)\left[(x \vdots \pi(i)^y) \wedge \neg(x \vdots \pi(i)^{y+1})\right].$$

На самом деле последняя формула показывает, что функция двух переменных $\mathrm{pw}(i,x)$ есть примитивно рекурсивная функция (правда, надо доказать, что функция x^y примитивно рекурсивна).

Примитивная рекурсия возникает как реализация метода математической индукции при определении функции. Базовый вариант метода состоит в следующем. Доказываем какое-то утверждение, которое можно рассматривать как истинность некоторого предиката $R(x)$. Если: а) $R(0)$ истинно; б) из истинности $R(n)$ следует истинность $R(n+1)$, то предикат $R(n)$ является тождественно истинным (наше утверждение верно для любого n).

Однако известны различные модификации метода математической индукции, в которых для установления истинности $R(n+1)$ используется истинность нескольких предыдущих значений предиката: истинность $R(k_1)$, $R(k_2)$, …, $R(k_s)$, где $k_1 < k_2 < \ldots < k_s \leqslant n$.

Аналогичным образом значение целочисленной функции $h(x_1, x_2, \ldots, x_n, y+1)$ может определяться на основе не только значения $h(x_1, x_2, \ldots, x_n, y)$, но и некоторых предыдущих значений $h(x_1, x_2, \ldots, x_{n-1}, k_i)$ с $k_i < y$. В этом случае говорят о ***возвратной рекурсии***. Точное определение таково. Говорят, что функция h^{n+1} получается возвратной рекурсией из функций f^n, g^{n+s+1}, t_1^1, …, t_s^1, где $t_i(y) \leqslant y$, $i = \overline{1,\, s}$, если

$$\begin{aligned} &h^{n+1}(x,0) = f^n(x),\\ &h^{n+1}(x,y+1) = g^{n+s+1}(x,y,h^{n+1}(x,t_1^1(y)),\ldots,h^{n+1}(x,t_s^1(y)) \end{aligned} \tag{9.2}$$

(здесь для упрощения через x обозначена группа переменных x_1, x_2, …, x_n).

Теорема 9.6. Функция, полученная возвратной рекурсией из примитивно рекурсивных функций, примитивно рекурсивна.

◀ Идея доказательства состоит в переходе от конструируемой функции h^{n+1} к другой в некотором роде „аккумулирующей“ функции H^{n+1}, для которой* $H(x,y)$ содержит в себе информацию о всех значениях $h(x,k)$, $0 \leqslant k \leqslant y$, причем любое значение $h(x,k)$ извлекается из $H(x,y)$ стандартной процедурой. Тогда из примитивной рекурсивности H мы сможем сделать заключение о примитивной рекурсивности h. Речь идет о кодировании нескольких чисел в одно.

Введем в рассмотрение ***производящую функцию***

$$H(x,y) = \prod_{i=0}^{y} \pi(i)^{h(x,i)}$$

Тогда

$$h(x,k) = \mathrm{pw}(k, H(x,y)), \quad k \leqslant y. \tag{9.3}$$

Докажем, что функция $H(x,y)$ примитивно рекурсивна. Имеем

$$H(x,0) = \pi(0)^{h(x,0)}, \quad H(x,y+1) = \prod_{i=0}^{y-1} \pi(i)^{h(x,i)} \cdot \pi(y+1)^{h(x,y+1)} = H(x,y)\pi(y+1)^{h(x,y+1)}.$$

Поскольку $h(x,y)$ получена возвратной рекурсией, т.е. по формуле (9.2), заключаем, что

$$\begin{aligned} h(x,y+1) = g(x,y,h(x,t_1(y)),\ldots,h(x,t_s(y)) =\\ = g(x,y,\mathrm{pw}(t_1(y),H(x,y)),\ldots,\mathrm{pw}(t_s(y),H(x,y)) = G(x,y,H(x,y)), \end{aligned}$$

где

$$G(x,y,z) = g(x,y,\mathrm{pw}(t_1(y),z),\ldots,\mathrm{pw}(t_s(y),z).$$

В результате

$$H(x,y+1) = H(x,y)\,\pi(y+1)^{G(x,y,H(x,y))}$$

и функция $H(x,y)$ получена примитивной рекурсией. Следовательно, $H(x,y)$ примитивно рекурсивна. Согласно формуле (9.3) при $k=y$, функция $h(x,y)$ также примитивно рекурсивна. ▶

*В доказательстве мы опять через x обозначаем группу неизменных аргументов x_1, x_2, …, x_n.

9.3. Частично рекурсивные функции

Оказывается, что класс примитивно рекурсивных функции не охватывает множество всех вычислимых функций (в частности, вычислимых по Тьюрингу или Маркову). Первое обстоятельство — примитивно рекурсивные функции всюду определены. Но есть и второе, более серьезное обстоятельство, связанное с ростом функций.

Функция Аккермана. Рассмотрим функцию $B(x,y)$, называемую ***функцией Аккермана***, которая определяется индуктивно следующими формулами:

$$B(0,y)=2+y,\quad B(x,0)=\operatorname{sg}(x-1),\ \ x\geqslant 1,\quad B(x+1,y+1)=B(x,B(x+1,y)),$$

Из формул видно, что для вычисления $B(x+1,y+1)$ необходимо знать $B(x+1,y)$ (это укладывается в операцию примитивной рекурсии) и все значения $B(x,z)$, поскольку мы не знаем, какое конкретно $z=B(x+1,y)$ будет использовано. В действительности вычисление значений функции идет в общем порядке возрастания пар x, y, но конкретный порядок перебора этих пар устроен весьма сложно. Тем не менее эта функция вычислима.

Непосредственно из формул вытекают соотношения

$$B(0,y)=2+y,\quad B(1,y)=2y,\quad B(2,y)=2^y,\quad B(3,y)=2^{2^{\cdot^{\cdot^{2}}}}\ \ (y\text{ раз}).$$

Видно, что с ростом x функция $B(x,y)$ по y растет все быстрее. В то же время

$$B(x,0)=\operatorname{sg}(x),\quad B(x,1)=2+\overline{\operatorname{sg}}(x),\quad B(x,2)=4, \tag{9.4}$$

т.е. при $y\leqslant 2$ функция $B(x,y)$ как функция переменного x ограничена. Однако

$$B(0,3)=5,\quad B(1,3)=6,\quad B(2,3)=8,\quad B(3,3)=16,\quad \ldots,$$

и видно, что функция $B(x,3)$ растет.

Теорема 9.7. Функция $B(x,y)$ удовлетворяет условиям

$$B(x,y)<B(x,y+1),\ \ x\in\mathbb{N},\quad B(x,y+1)\leqslant B(x+1,y),\ \ x\in\mathbb{N}.$$

◄ Сначала докажем неравенство $B(x,y)\geqslant y+2$, верное при $y\geqslant 2$. При $x=0$ оно верно согласно определению функции Аккермана. Оно также верно при любом $x>0$ и при $y=2$, поскольку $B(x,2)=4$. Предположим, что оно верно при данном значении x для любого $y\geqslant 2$ и при значениях $x+1$ и $y>2$. Тогда

$$B(x+1,y+1)=B(x,B(x+1,y))\geqslant B(x+1,y)+2\geqslant y+4>y+2.$$

Теперь докажем неравенство $B(x,y+1)>B(x,y)$. Оно очевидно при $x=0$. Оно также верно, согласно (9.4), при любом x и при $y<2$. Пусть $x>0$ и $y\geqslant 2$. Тогда $B(x,y)\geqslant y+2>2$ и

$$B(x,y+1)=B(x-1,B(x,y))\geqslant B(x,y)+2>B(x,y).$$

Наконец, покажем, что $B(x+1,y)\geqslant B(x,y+1)$ при $y\geqslant 3$.

$$B(x+1,y)=B(x,B(x+1,y-1))\geqslant B(x,y-1+2),$$

поскольку $B(x+1,y-1)\geqslant y+1$ при $y\geqslant 3$ и $B(x,z_1)>B(x,z_2)$ при $z_1>z_2$. ►

Можно показать, что каждая примитивно рекурсивная функция $f(x_1,x_2,\ldots,x_n)$ при некотором достаточно большом m удовлетворяют неравенству

$$f(x_1,x_2,\ldots,x_n)\leqslant B(m,\|x\|),\quad \|x\|>1.$$

где $\|x\| = \max\{x_1, x_2, \ldots, x_n\}$. Для этого достаточно проверить простейшие функции (для них $m = 0$) и сохранение свойства при суперпозиции и примитивной рекурсии.

Указанным свойством не обладает ***диагональная функция Аккермана*** $A(x) = B(x, x)$. Действительно, тогда должно было бы выполняться неравенство $B(x, x) \leqslant B(m, x)$. Однако это неверно при $x > m$. Значит, диагональная функция Аккермана не является примитивно рекурсивной.

Функция Аккермана — пример вычислимой функции, не являющейся примитивно рекурсивной. Существование таких функций указывает на то, что класс примитивно рекурсивных функций должен быть расширен. Это требование разрешается введением еще одной операции. Скажем, что n-арная функция h^n получена из $(n + 1)$-местного предиката P^{n+1} с помощью ***оператора минимизации*** (***μ-оператора***), если

$$h^n(x_1, x_2, \ldots, x_n) = \mu y P^{n+1}(x_1, x_2, \ldots, x_n, y),$$

где символ μ означает выбор первого номера y, при котором $P(x_1, x_2, \ldots, x_n, y)$ является истинным (т.е. равен единице).

Отметим, что оператор μ уже был использован, но с дополнительным ограничением, когда y не превышал одной из переменных (в силу этого такой вариант называют ***оператором ограниченной минимизации***). Оператор ограниченной минимизации не выводит из класса примитивно рекурсивных функций (это было установлено). Оператор общей минимизации уже нельзя выразить через операции композиции и примитивной рекурсии.

Определение 9.1. Скажем, что целочисленная функция f частично рекурсивна, если она удовлетворяет одному из условий:

1) примитивно рекурсивная функция частично рекурсивна;

2) функция, полученная с помощью оператора минимизации из частично рекурсивной функции, является частично рекурсивной.

Частичная рекурсивная функция при некоторых значениях аргументов может быть не определена. Пусть при данных $x_1, x_2, \ldots, x_n$ значение предиката $P(x_1, x_2, \ldots, x_n, y)$ равно нулю независимо от значения переменной y. Тогда оператор минимизации дает неопределенное значение. Внешне это выглядит как бесконечная проверка $y = 1, 2, 3, \ldots$, которая никогда не заканчивается — ситуация, аналогичная неприменимости нормального алгорифма или машины Тьюринга.

Если частично рекурсивная функция определена при любых значениях аргументов, ее называют ***общерекурсивной***.

Пример 9.4. Рассмотрим предикат $P(x, y, z) = (x = y + z)$. С помощью него и оператора минимизации определим функцию

$$s(x, y) = \mu z\, P(x, y, z).$$

Смысл этой функции очевиден: $s(x, y) = x - y$, если $x \geqslant y$, и $s(x, y)$ не определена, если $x < y$.

Теорема 9.8. Любая частично рекурсивная функция вычислима по Маркову (Тьюрингу).

◀ Понятия вычислимости по Маркову и Тьюрингу эквивалентны. Остановимся на вычислимости по Маркову, для чего будем опираться на сочетания нормальных алгорифмов. Надо доказать, что базисные функции вычислимы по Маркову и что операции образования новых функций сохраняют свойство вычислимости по Маркову.

Очевидно, что простейшие функции вычислимы по Маркову. Также легко установить, что композиция функций, вычислимых по Маркову, является вычислимой по Маркову: достаточно сделать соединение нормальных алгорифмов в соответствии с рис. 9.1, на котором алгоритм Z_0 удаляет лидирующий нуль в результате, представленном как γ-система.

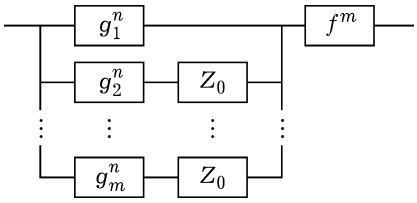
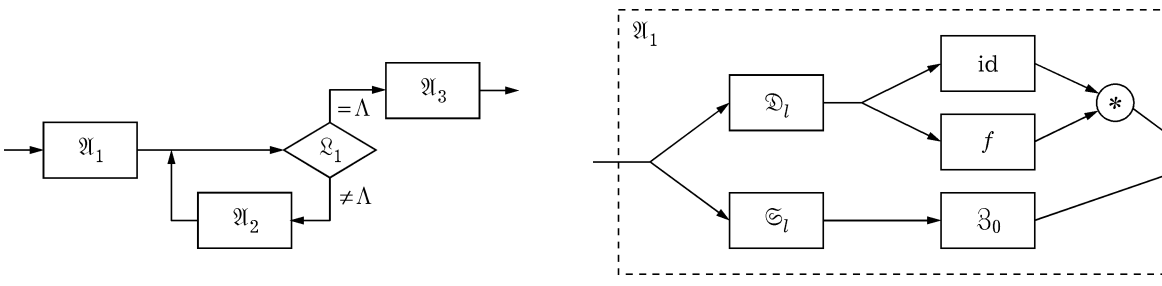


Рис. 9.1

Перейдем к оператору примитивной рекурсии, который определяет функцию h^{n+1} по функциям f^n и g^{n+2}. Чтобы вычислить функцию h^{n+1} для аргументов x_1, x_2, …, x_n, y, необходимо многократно вычислять функцию g^{n+2}, а функцию f^n использовать как начальное значение. В результате наш алгоритм реализуется как повторение с предусловием (рис. 9.2, *а*). Алгоритм $\mathfrak{A}_1$ формирует γ-систему $01^{x_1}0\ldots01^{x_n}01^i01^z01^y0$, где i — текущее значение последнего аргумента функции h^{n+1}, z — текущее значение самой функции h^{n+1}, y — счетчик рекурсии (оно же — конечное значение i). Начальное значение i нулевое, начальное значение y — то, при котором надо вычислить h^{n+1}, а начальное значение z есть $h^{n+1}(x_1, x_2, \ldots, x_n, 0) = f^n(x_1, x_2, \ldots, x_n)$. На каждом шаге мы проверяем условие $y = 0$ (алгоритм $\mathfrak{L}_1$). Если оно выполняется, выделяем из γ-системы слово z и предъявляем как результат (алгоритм $\mathfrak{A}_3$). Если же $y > 0$, то в γ-системе заменяем z на $g^{n+2}(x_1, x_2, \ldots, x_n, i, z)$, затем i на $i + 1$, а y на $y - 1$ (алгоритм $\mathfrak{A}_2$).

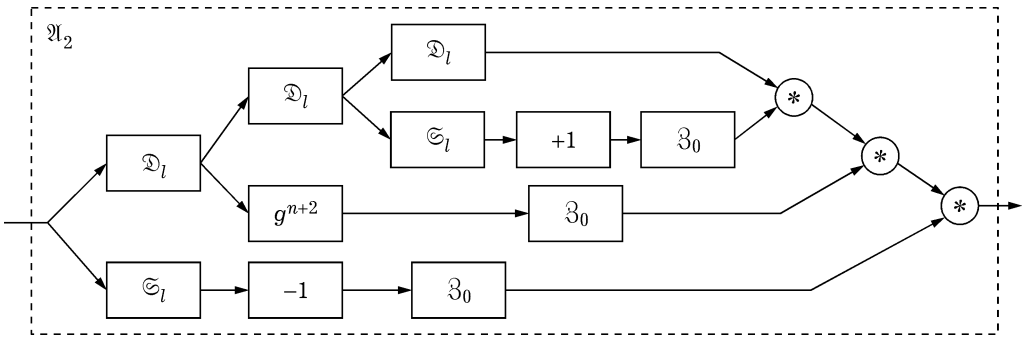
Алгоритм $\mathfrak{A}_1$ можно представить как соединение нескольких алгоритмов (рис. 9.2, *б*). В этом соединении алгоритм $\mathfrak{D}_l$ удаляет последний элемент γ-системы, алгоритм $\mathfrak{S}_l$, наоборот, извлекает из γ-системы последний элемент, алгоритм id пустой, алгоритм f реализует вычисление функции f, алгоритм $\mathfrak{Z}_0$ удаляет лидирующий нуль в γ-системе. Структура алгоритма $\mathfrak{A}_2$ представлена на рис. 9.2, *в*, алгоритмы, обозначенные „+1“ и „−1“, соответственно прибавляют к числу 1 и вычитают. Алгоритм $\mathfrak{L}_1$ реализуется как композиция двух алгоритмов: первый алгоритм $\mathfrak{S}_l$ извлекает из γ-системы параметр y, второй проверяет условие $y = 0$ и состоит из одной подстановки $00 \to\cdot\ \Lambda$. Наконец, алгоритм $\mathfrak{A}_3$ есть композиция алгоритмов $\mathfrak{D}_l$ (удаляет счетчик y) и $\mathfrak{S}_l$ (извлекает z).

Оператор минимизации также реализуется как повторение с предусловием. Формируем γ-систему $01^{x_1}0\ldots01^{x_n}01^y0$. Начальное значение y нулевое. На каждом шаге проверяем значение



а

б



в

Рис. 9.2

предиката P. Если значение 0, увеличиваем значение y на единицу и процедуру проверки повторяем. Если $P = 0$, выделяем из γ-системы последнее слово и завершаем работу алгоритма. ▶

Теорема 9.9. Любая функция, вычислимая по Тьюрингу (Маркову), частично рекурсивна.

◀ В данной теореме мы будем ориентироваться на машины Тьюринга, учитывая, что вычислимость по Тьюрингу равносильна вычислимости по Маркову. Конечно, и машины Тьюринга, и нормальные алгорифмы реализуют словарные функции; числовые функции трактуются как особый тип словарных функций. Но для доказательства теоремы такой трактовки недостаточно, поскольку в процессе работы алгоритм часто выходит за рамки двухбуквенного алфавита. Поэтому необходимо любой словарной функции придать смысл функции числовой.

Пусть $A = a_0a_1 \dots a_{m-1}$ — внешний алфавит машины Тьюринга, $Q = q_0q_1 \dots q_{n-1}$ — ее внутренний алфавит. Каждое слово в алфавите A можно интерпретировать как запись числа в системе исчисления с основанием m и цифрами a_0, a_1, …, a_{m-1}, т.е. слову $X = a_{j_1}a_{j_2} \dots a_{j_s}$ мы ставим в соответствие число

$$n(X) = j_s + j_{s-1} * m + \ldots + j_1 * m^{s-1}.$$

Каждому состоянию q_i ставим в соответствие число i. Представленная числовая интерпретация элементов машины Тьюринга называется ее ***арифметизацией***. Арифметизация машины Тьюринга позволяет любую конфигурацию Xq_ja_iY представить как совокупность четырех чисел: $n(X)$, j, i, $n(\overline{Y})$, $\overline{Y}$ — слово Y в обратном порядке. Инверсия слова Y позволяет пустые символы неограниченной ленты интерпретировать как незначащие разряды числа. Каждый шаг машины Тьюринга состоит из трех действий, зависящих от внутреннего состояния q_j и обозреваемого символа a_i: смене внутреннего состояния, замене обозреваемого символа и сдвиге читающей головки. Каждое из этих действий описывается соответствующей функцией: новое внутреннее состояние $\hat{q}(j,i)$, печатаемый символ $\hat{a}(j,i)$, сдвиг $\hat{s}(j,i)$, который можно описать тремя значениями 0 (сдвига нет), 1 (влево), 2 (вправо). Каждая из этих функций определена на конечном множестве пар натуральных чисел. Доопределив их нулевым значением вне этого множества, получим примитивно рекурсивные функции.

Каждую конфигурацию Xq_ja_iY, как сказано, можно представить как совокупность четырех чисел $u_1 = n(X)$, $u_2 = j$, $u_3 = i$, $u_4 = n(\overline{Y})$. Один шаг машины Тьюринга можно представить как преобразование этих чисел в новую четверку $u_1' = n(X')$, $u_2' = j'$, $u_3' = i'$, $u_4' = n(\overline{Y}')$. Это преобразование описывается функциями

$$u_1' = \begin{cases} u_1, & s(u_2,u_3) = 0; \\ [u_1/m], & s(u_2,u_3) = 1; \\ u_1 * m + \hat{a}(u_2,u_3), & s(u_2,u_3) = 2; \end{cases} \qquad u_2' = \hat{q}(u_2,u_3);$$

$$u_3' = \begin{cases} \hat{a}(u_2,u_3), & s(u_2,u_3) = 0; \\ u_1 \bmod m, & s(u_2,u_3) = 1; \\ u_4 \bmod m, & s(u_2,u_3) = 2; \end{cases} \qquad u_4' = \begin{cases} u_4, & s(u_2,u_3) = 0; \\ u_4 * m + \hat{a}(u_2,u_3), & s(u_2,u_3) = 1; \\ [u_4/m], & s(u_2,u_3) = 2. \end{cases}$$

Таким образом, один шаг машины Тьюринга описывается четырьмя функциями, которые в силу их представления являются примитивно рекурсивными. Для описания многошаговой работы нужно ввести еще один аргумент, играющий роль времени: $t = 0$ соответствует начальной конфигурации машины Тьюринга, $t = k$ — конфигурации, полученной после k-го шага. В результате получим функции $u_i(t)$, которые определяются следующим образом:

$$u_i(u^0, 0) = u_i^0, \quad u_i(u^0, t+1) = u_i'(u_1(u^0,t), u_2(u^0,t), u_3(u^0,t), u_4(u^0,t)), \quad i = \overline{1,\,4},$$

где $u^0 = (u_1^0,\, u_2^0,\, u_3^0,\, u_0^4)$. Это похоже на примитивную рекурсию, но все четыре функции перевязаны. Выход: упаковать все четыре функции в одну так, что каждая извлекается из

упакованного варианта с помощью примитивно рекурсивной функции (например, используя характеристическую функцию как в возвратной рекурсии).

Наконец, условие останова реализуется с помощью оператора минимизации. Сперва определяется t как функция начальной конфигурации: $t = \mu\tau(u_2(u^0, t) = 0)$ (предполагается замкнутая машина Тьюринга). Затем эта функция начальных условий подставляется в функции u_i, и мы получаем u_i как функции только начальных условий.

Можно считать, что машина Тьюринга начинает и завершает работу в стандартном положении, т.е. с положением читающей головки перед первым символом. Тогда в начале работы $u_1^0 = 0$, $u_1^0 = 1$, $u_2^0 = \overline{X} \bmod m$, $u_3^0 = [\overline{X}/m]$, в конце работы $Y = \overline{u}_4^e * m + u_3^e$. Здесь X — исходное слово, Y — результирующее слово, в которое машина Тьюринга преобразовала слово X. В результате мы получили интерпретацию работы машины как вычисление частично рекурсивной функции.

Пусть задана функция $f(x_1, x_2, \ldots, x_n)$, вычислимая по Тьюрингу. Соответствующая машина Тьюринга преобразует слово, представляющее собой запись аргументов функции в „палочной“ системе, в слово, представляющее собой запись результата в той же „палочной“ системе. Эту машину представляет некоторая функция частично рекурсивная $g(x)$ одного переменного, в которой исходное слово X и результирующее слово Y интерпретируются как запись чисел $n(X)$ и $n(Y)$ в системе исчисления с основанием m. Исходную функцию $f(x_1, x_2, \ldots, x_n)$ можно представить как композицию трех функций, первая преобразует набор чисел x_1, x_2, …, x_n в число $n(X)$, вторая — это $g(x)$, третья преобразует число $n(Y)$ в результат $f(x_1, x_2, \ldots, x_n)$. В свете этого, чтобы доказать частичную рекурсивность функции f, достаточно доказать частичную рекурсивность двух преобразований. Первое преобразование описывается формулой

$$n(X) = m + \ldots + m^{x_n} + m^{x_n+2} + m^{x_n+3} + \ldots + m^{x_n+x_{n-1}+2} + \ldots,$$

из которых видно, что это преобразование есть примитивно рекурсивная функция. Второе преобразование обратное. Его можно интерпретировать как пересчет числа в сумму его цифр. Можно показать, что она примитивно рекурсивна. С учетом этого факта заключаем, что функция $f(x_1, x_2, \ldots, x_n)$ частично рекурсивна. ►

9.4. Универсальные рекурсивные функции

Пусть дана $(n+1)$-арная частично рекурсивная функция $f(x_0, x_1, \ldots, x_n)$. Ясно, что для любого конкретного числа l функция $g_l(x_1, x_2, \ldots, x_n) = f(l, x_1, \ldots, x_n)$ является частично рекурсивной. Другими словами, каждая частично рекурсивная функция от $n+1$ аргументов порождает счетное семейство частично рекурсивных функций от n аргументов. Может ли быть такое, что это семейство накрывает множество всех частично рекурсивных функций от n аргументов? Существование универсального алгорифма наталкивает на мысль, что такое вполне возможно.

Ключевым пунктом здесь является следующая теорема.

Теорема 9.10 (*теорема Клини о нормальной форме*). Для каждого натурального $n \geqslant 1$ существуют такая примитивно рекурсивная функция F^1 и такой примитивно рекурсивный предикат D^{n+2}, что для любой n-арной частично рекурсивной функции f^n при некотором значении l имеет место представление

$$f^n(x_1, x_2, \ldots, x_n) \simeq F^1(\mu y D^{n+2}(l, x_1, x_2, \ldots, x_n, y)). \quad \#$$

Как видим, любая частично рекурсивная функция может порождаться только одним применением оператора минимизации.

Пусть $\mathcal{F}^n$ — некоторое множество n-арных частично рекурсивных функций. Скажем, что функция U^{n+1} является универсальной для множества $\mathcal{F}^n$, если оно совпадает с множеством

всех функций $f_l(x_1,\ldots,x_n) = U^{n+1}(l,x_1,\ldots,x_n)$, $l \in \mathbb{N}$. В качестве такого множества будем рассматривать множество $\mathcal{F}^n_{\text{чр}}$ всех частично рекурсивных функций, множество $\mathcal{F}^n_{\text{ор}}$ всех общерекурсивных функций и множество $\mathcal{F}^n_{\text{пр}}$ всех примитивно рекурсивных функций.

Теорема 9.11. 1. Существует универсальная частично рекурсивная функция для множества $\mathcal{F}^n_{\text{чр}}$.

2. Существует общерекурсивная функция, универсальная для множества $\mathcal{F}^n_{\text{пр}}$, причем ни одна такая функция не является примитивно рекурсивной.

3. Множество $\mathcal{F}^n_{\text{ор}}$ не имеет универсальной общерекурсивной функции.

Первое утверждение теоремы — очевидное следствие из теоремы Клини. Два других утверждения менее очевидны. Отметим, что из второго утверждения вытекает, что существуют общерекурсивные функции, не являющиеся примитивно рекурсивными.

Универсальные частично рекурсивные функции в определенном смысле имеют максимальную область определения. Скажем, что функция g^n является доопределением функции f^n, если ее область определения включает область определения f^n и на области определения f^n значения двух функций совпадают.

Теорема 9.12. Никакая универсальная частично рекурсивная функция для множества $\mathcal{F}^n_{\text{ор}}$ не может быть доопределена до общерекурсивной.

◀ Пусть U^{n+1} — универсальная частично рекурсивная функция и G^{n+1} — ее общерекурсивное доопределение. Рассмотрим функцию $g(x_1,x_2,\ldots,x_n) = G^{n+1}(x_1,x_1,x_2,\ldots,x_n)+1$. Это общерекурсивная функция n переменных и должна выражаться через универсальную функцию. Следовательно, при некотором l

$$g(x_1,x_2,\ldots,x_n) = U^{n+1}(l,x_1,x_2,\ldots,x_n) = G^{n+1}(l,x_1,x_2,\ldots,x_n).$$

Но, учитывая определение функции g, заключаем, что

$$G^{n+1}(x_1,x_1,x_2,\ldots,x_n)+1 = G^{n+1}(l,x_1,x_2,\ldots,x_n).$$

В частности,

$$G^{n+1}(l,l,\ldots,l)+1 = G^{n+1}(l,l,\ldots,l),$$

а это невозможно. ▶

Конечно, множество $\mathcal{F}^n_{\text{ор}}$ имеет универсальную частично рекурсивную функцию (таковой является любая универсальная функция для множества $\mathcal{F}^n_{\text{чр}}$). Согласно теореме 9.12 среди универсальных функций для множества $\mathcal{F}^n_{\text{ор}}$ нет общерекурсивных.

Следствие 9.1. Существует одноместный частично рекурсивный предикат, не имеющий общерекурсивных доопределений.

◀ Выберем какую-нибудь универсальную общерекурсивную функцию U^{n+1} и положим $f(x) = \overline{\text{sg}}(U^{n+1}(x,x,\ldots,x))$. Пусть эта функция имеет общерекурсивное доопределение $v(x)$. Образуем функцию n переменных, введя фиктивные переменные: $g(x_1,x_2,\ldots,x_n) = v(x_1)$. Эта функция, как частично рекурсивная, выразима через универсальную. Поэтому при некотором l

$$\overline{\text{sg}}(U^{n+1}(x,x,\ldots,x)) = v(x) = U^{n+1}(l,x,x_2,\ldots,x_n).$$

В частности,

$$\overline{\text{sg}}(U^{n+1}(l,l,\ldots,l)) = U^{n+1}(l,l,l,\ldots,l),$$

а это равенство неверно. ▶

9.5. Разрешимые и перечислимые множества

Под разрешимым понимают множество, для которого можно построить разрешающий алгоритм, т.е. алгоритм, который для любого объекта a определяет, принадлежит a множеству или нет. Мы рассматриваем подмножества множества натуральных чисел. Ясно, что среди них есть неразрешимые (на всех алгоритмов не хватает).

Каждое подмножество A в $\mathbb{N}$ можно описать ***характеристической функцией***

$$\chi_A(x) = \begin{cases} 1, & x \in A; \\ 0, & x \notin A. \end{cases}$$

Разрешимость множества равносильна вычислимости его характеристической функции. Понятие вычислимой функции мы отождествляем с понятием частично рекурсивной функции. Для целей описания множеств нужны всюду определенные, т.е. общерекурсивные функции. Это объясняет следующее определение.

Определение 9.2. ***Разрешимое множество*** (также ***рекурсивное множество***) — любое подмножество множества натуральных чисел, характеристическая функция которого является общерекурсивной. Если характеристическая функция примитивно рекурсивна, то множество называют ***примитивно рекурсивным***.

Пример 9.5. Пустое множество, множество всех натуральных чисел примитивно рекурсивны, поскольку их характеристические функции, будучи постоянными, примитивно рекурсивны. Множество четных чисел примитивно рекурсивно: его характеристическая функция описывается формулой $f(x) = 1 \dot{-} (x \bmod 2)$.

Теорема 9.13. Если множества A и B разрешимы (примитивно рекурсивны), то и множества $A \cap B$, $A \cup B$, $A \setminus B$, $\overline{A} = \mathbb{N} \setminus A$ разрешимы (примитивно рекурсивны).

◀ Утверждение теоремы вытекает из следующих представлений теоретико-множественных операций через характеристические функции:

$$\chi_{A\cap B} = \chi_A \chi_B, \quad \chi_{A\cup B} = \chi_A + \chi_B \dot{-} \chi_A \chi_B, \quad \chi_{A\setminus B} = \chi_A(1 \dot{-} \chi_B), \quad \chi_{\overline{A}} = 1 \dot{-} \chi_A. \qquad ▶$$

Пример 9.6. Одноточечное множество $A = \{a\}$ примитивно рекурсивно, поскольку его характеристическая функция $\chi_A(x) = \overline{\mathrm{sg}}(|x - a|)$ примитивно рекурсивна. Конечное множество примитивно рекурсивно как конечное объединение примитивно рекурсивных множеств.

Теорема 9.14. Если общерекурсивная (примитивно рекурсивная) функция f удовлетворяет условию $f(x) \geqslant x$, $x \in \mathbb{N}$, то множество ее значений разрешимо (примитивно рекурсивно).

◀ Суть в механизме проверки, является ли данное число x значением функции f. При $y > x$ имеем $f(y) \geqslant y > x$, так что равенство $x = f(y)$ надо проверять только для значений y, не превышающих x. Характеристическую функцию множества A значений функции f можно записать с помощью ограниченного квантора существования:

$$\chi_A(x) = \exists (y \leqslant x)(f(y) = x).$$

Из этой формулы вытекает общерекурсивность (примитивная рекурсивность) χ_A в случае, когда $f(y)$ общерекурсивна (примитивно рекурсивна). ▶

Пример 9.7. Как было показано, функция $\mathrm{Pr}(x)$, значением которой является простое число с номером x, примитивно рекурсивна. Множество ее значений — множество всех простых чисел — является примитивно рекурсивным, поскольку $\mathrm{Pr}(x) \geqslant x$, $x \in \mathbb{N}$.

Определение 9.3. Множество $A \subset \mathbb{N}$ называется ***рекурсивно перечислимым***, если оно совпадает с областью определения некоторой частично рекурсивной функции.

Определение можно переформулировать и так: множество A рекурсивно перечислимо, если существует частично рекурсивная функция f, удовлетворяющая условию: $f(x) = 1$, если $x \in A$, и $f(x)$ не определена, если $x \notin A$. Действительно, если A есть область определения частично рекурсивной функции g, то указанному требованию удовлетворяет функция $f(x) = 1(g(x))$.

Нетрудно понять, что понятие рекурсивно перечислимого множества более широкое по сравнению с понятием разрешимого множества. Действительно, если для каждого n мы можем проверить, принадлежит оно множеству или нет, то ясен алгоритм и перечисления таких чисел: последовательно просматривая натуральный ряд, мы присваиваем функции значение 1, если число принадлежит множеству, или зацикливаем алгоритм в противном случае. Докажем этот факт строго.

Теорема 9.15. Разрешимое множество рекурсивно перечислимо.

◀ Согласно условию характеристическая функция χ_A множества A общерекурсивна. Составим функцию $f_A(x) = \mu y(\chi_A(x)) + 1$. Если $\chi_A(x) = 0$, то значение оператора минимизации не определено и функция $f_A(x)$ не определена. Если $\chi_A(x) = 1$, то $\mu y(\chi_A(x)) = 0$ и значение $f_A(x)$ есть 1. Следовательно, область определения функции f_A совпадает с множеством A и это множество рекурсивно перечислимо. ▶

Теорема 9.16. Непустое множество рекурсивно перечислимо тогда и только тогда, когда оно является областью значений некоторой одноместной примитивно рекурсивной функции.

◀ Доказательство в одну сторону основано на той же идее, что и предыдущая теорема, только квантор существования уже не будет ограниченным и вместо него лучше использовать оператор минимизации. Итак, пусть A — множество значений примитивно рекурсивной функции f. Сформируем двуместный предикат $f(y) = x$ и используем оператор минимизации:

$$g(x) = \mu(y)(f(y) = x).$$

Функция $g(x)$ частично рекурсивна, так как получена с помощью оператора минимизации из примитивно рекурсивного предиката. Она не определена, если x не является значением функции f, и имеет значение наименьшего числа y, для которого $f(y) = x$, если x является значением функции f.

В другую сторону доказательство сложнее. Речь идет о перестройке процедуры, порождающей частично рекурсивную функцию. Согласно теореме Клини о нормальной форме, частично рекурсивная функция одного переменного может быть получена из примитивно рекурсивного предиката одним оператором минимизации (функция F^1 в нормальной форме не играет роли). Итак, пусть частично рекурсивная функция $f(x)$ получена с помощью примитивно рекурсивного предиката $P(x, y)$ согласно формуле $f(x) = \mu y(P(x, y))$. Согласно этой формуле, число x принадлежит области определения функции f, если для этого x при некотором y предикат $P(x, y)$ имеет значение 1. Надо пересчитать все такие x, может быть, с повторением. Для этого достаточно перенумеровать пары $(x,\ y)$ в одну последовательность, т.е. сформировать функцию $c(x, y)$. Этой функции соответствуют функции $l(n)$ и $r(n)$, вычисляющие по номеру пары левую и правую компоненты. Все три функции при подходящем выборе нумерации примитивно рекурсивны (это можно показать). Далее создаем функцию

$$g(n) = \begin{cases} l(n), & P(l(n), r(n)); \\ b, & \neg P(l(n), r(n)), \end{cases}$$

где b — один из фиксированных элементов множества, которое по условию непусто. Очевидно, что множество значений функции g совпадает с рассматриваемым множеством. ▶

Теорема 9.17. Если A и B рекурсивно перечислимы, то $A \cap B$ и $A \cup B$ рекурсивно перечислимы.

◀ Пусть A и B — области определения функций f_A и f_B. Тогда область определения, например, функции $f_A(x)f_B(x)$ есть $A \cap B$.

Отметим, что при $A = \varnothing$ или $B = \varnothing$ множество $A \cup B$ совпадает с B или A, а потому рекурсивно перечислимо. Предположим, что A И B не пусты. Тогда существуют примитивно рекурсивные функции g_A и g_B, множества значений которых совпадают с A и B. Рассмотрим функцию

$$h(x) = \begin{cases} g_A([x/2]), & x \bmod 2 = 0; \\ g_B((x/2]), & x \bmod 2 \neq 0. \end{cases}$$

Множеством значений этой функции является $A \cup B$. ▶

Как показано выше класс рекурсивно перечислимых множеств включает в себя класс разрешимых множеств. Уточним связь между двумя классами множеств.

Теорема 9.18. Существуют рекурсивно перечислимые множества, не являющиеся разрешимыми. Существуют рекурсивно перечислимые множества, дополнения к которым не являются рекурсивно перечислимыми.

◀ Область определения A предиката f из следствия 9.1 — рекурсивно перечислимое множество. Рассмотрим функцию $h(x) = 1(f((x))$, равную 1 на множестве A и не определенную вне A. Если бы множество A было разрешимо, существовал бы предикат χ_A, равный 1 на A и 0 вне A, т.е. функция h имела бы общерекурсивное доопределение. Но тогда и функция f имеет общерекурсивное доопределение:

$$\tilde{f}(x) = \begin{cases} f(x), & \chi_A(x); \\ 0, & \neg\chi_A(x). \end{cases}$$

Поскольку в силу следствия 9.1 функция f не имеет общерекурсивного доопределения, то и предикат χ_A не существует, а множество не является разрешимым.

Пусть D^3 — предикат из нормальной формы Клини для $n = 1$. Рассмотрим функцию $\varphi(x) = \mu y D^3(x, x, y)$. Поскольку φ образована из примитивно рекурсивного предиката оператором минимизации, она частично рекурсивна. Ее область определения $B = \{x \in \mathbb{N}\colon \exists y D(x, x, y)\}$. Пусть X — произвольное рекурсивно перечислимое множество. Согласно теореме о нормальной форме его можно определить как область определения функции $\mu y D^3(l, x, y)$ при некотором значении l. Выясняется, что если $l \in B$, т.е. $\exists y D(l, l, y)$, то $l \in X$, а если $l \notin B$, то $l \notin X$. Значит, произвольно взятое рекурсивно перечислимое множество не может быть дополнением к B и дополнение к B не является рекурсивно перечислимым. ▶

Теорема 9.19 (*теорема Поста*). Если множество A и его дополнение рекурсивно перечислимы, то множество A разрешимо.

◀ Можно выделить случай, когда либо A, либо его дополнение пусто. В этом случае утверждение тривиально. Будем считать, что $A \neq \varnothing$, $\overline{A} \neq \varnothing$. Существуют примитивно рекурсивная функция f_1, множество значений которой есть A, и примитивно рекурсивная функция f_2, множество значений которой есть $\overline{A}$. Рассмотрим функцию $f(x) = \mu y((f_1(y) = x) \vee (f_2(y) = x))$. Покажем, что она общерекурсивна. Это значит, что для любого x существует такое y, что либо $f_1(y) = x$, либо $f_2(y) = x$. Пусть x произвольно. Если $x \in A$, то $\exists y(f_1(y) = x)$ (т.е. x есть значение f_1), а если $x \in \overline{A}$, то y — значение f_2, т.е. $\exists y(f_2(y) = x)$.

Таким образом, функция f является общерекурсивной. Рассмотрим общерекурсивный предикат $v(x) = (f_1(f(x)) = x)$. Если $x \in A$, то $\exists y(f_1(y) = x)$ и $\nexists y(f_2(y) = x)$. Значит, значением $f(x)$ является первое же y, для которого $f_1(y) = x$. Но тогда $f_1(f(x)) = x$ и $v(x) = 1$. Пусть $x \notin A$. Тогда $\nexists y(f_1(y) = x)$ и $\exists y(f_2(y) = x)$. Следовательно, значение $f_1(f(x) \neq x$, каково бы ни было значение y. Поэтому $v(x) = 0$. Мы доказали, что общерекурсивный предикат v является характеристической функцией множества A. Следовательно, A разрешимо. ▶

10. НЕРАЗРЕШИМЫЕ АЛГОРИТМИЧЕСКИЕ ПРОБЛЕМЫ

Одна из причин разработки теории алгоритмов — проверка существования алгоритмов, решающих те или иные задачи. Если заявлен некоторый алгоритм, решающий ту или иную задачу, то проверить его качества — в основном техническая задача. Но если есть подозрение, что такого алгоритма не существует (например, алгоритма проверки формулы логики предикатов на истинность), обосновать это подозрение можно только тогда, когда алгоритм из интуитивного понятия трансформирован в строгое математическое понятие.

Разумеется, нельзя доказать, что та или иная формализация понятия алгоритма охватывает все мыслимые алгоритмы. В этом смысле тезис Черча (или Тьюринга, Маркова) — неразрешимая проблема.

Выделим так называемые ***массовые алгоритмические проблемы***. Массовой называют проблему существования алгоритма, решающего бесконечную серию однотипных задач. Если удается арифметизировать массовую проблему, т.е. перенумеровать все частные случаи серии однотипных задач и представить в символической форме возможные решения каждой задачи, то задача сводится к построению некоей функции h, которая каждой задаче с номером n ставит в соответствие ее решение $h(n)$. Символические записи всех возможных решений тоже можно перенумеровать. В этом смысле функция h будет функцией натурального аргумента. Существование алгоритма вычисления этой функции и есть математическая интерпретация разрешимости массовой проблемы. Дадим точное определение.

Пусть X — счетное множество, которое мы можем интерпретировать как множество слов в некотором алфавите A. Полагаем, что есть свойство, которым обладает или нет каждый элемент $x \in X$. Это свойство можно описать некоторым предикатом $P(x)$, определенным на множестве X. Массовая алгоритмическая проблема состоит в построении алгоритма $\mathfrak{A}$ в алфавите A, который применим к любому элементу $x \in X$, причем $\mathfrak{A}(x) = \Lambda$, если $P(x)$ истинно, и $\mathfrak{A}(x) \neq \Lambda$, если $P(x)$ ложно.

Массовая алгоритмическая проблема неразрешима, если такого алгоритма нет. Примером таких проблем является неразрешимость ряда формальных аксиоматических теорий (в частности, исчисления высказываний), т.е. отсутствие алгоритма, который по заданной формуле устанавливает, выводима эта формула в данной теории или нет.

Мы остановимся на одной внутренней проблематике теории алгоритмов.

10.1. Проблема распознавания самоприменимости

Если алфавит A содержит не менее двух букв, любой нормальный алгорифм можно записать в виде слова $\mathfrak{A}^3$ в том же алфавите A. Для этого достаточно в алфавите A выделить подалфавит A_0 из двух букв и перевести изображение алгоритма $\mathfrak{A}$ в алфавит A_0. То, что получится, назовем ***записью нормального алгорифма***. Эта запись может быть исходным словом для самого нормального алгорифма. Можно поставить вопрос, применим ли нормальный алгорифм к своей записи? Может быть да, а может быть и нет. Например, алгоритм, добавляющий к исходному слову первую букву алфавита A, применим к любому слову, в том числе и к своей записи. А если нормальный алгорифм неприменим к словам, имеющим некоторые сочетания букв (например, к словам, содержащим первую букву подалфавита A_0), то он может быть неприменимым к своей записи.

Будем говорить, что ***алгоритм $\mathfrak{A}$ самоприменим***, если $!\mathfrak{A}(\mathfrak{A}^3)$. Если же $\neg!\mathfrak{A}(\mathfrak{A}^3)$, то говорим, что ***алгоритм $\mathfrak{A}$ несамоприменим***. Можно ли построить алгоритм, проверяющий, является ли данный алгоритм $\mathfrak{A}$ самоприменимым?

Теорема 10.1. Пусть алфавит A содержит не менее двух букв. Не существует нормального алгорифма $\mathfrak{R}$ в алфавите A, который для любого нормального алгорифма $\mathfrak{A}$ в алфавите A применим к $\mathfrak{A}^3$ тогда и только тогда, когда нормальный алгорифм $\mathfrak{A}$ несамоприменим.

◀ Предположим, что алгоритм $\mathfrak{R}$ существует. Тогда в силу своего определения $!\mathfrak{R}(\mathfrak{R}^3)$, если $\mathfrak{R}$ несамоприменим. Но эти два утверждения взаимно исключающие. ▶

Доказанная, в общем-то тривиальная, теорема ограничивается лишь частным случаем нормальных алгорифмов, не использующих специальных символов. Однако общий случай можно свести к этому частному. Пусть алфавит A содержит не менее 4-х букв. Зафиксируем подалфавит A_0 из двух букв, которые используем для записи нормальных алгорифмов в алфавите A, и выделим подалфавит $A_1 \subset A$ из четырех букв, включающий в себя A_0.

Теорема 10.2. Пусть A содержит не менее четырех букв и $A_0 \subset A$ — двухбуквенный подалфавит. Не существует нормального алгорифма $\mathfrak{R}$ над алфавитом A_0, который для любого нормального алгорифма $\mathfrak{A}$ в алфавите A применим к $\mathfrak{A}^3$ тогда и только тогда, когда $\mathfrak{A}$ несамоприменим.

◀ Пусть такой алгоритм существует. Согласно теореме приведения, можно построить нормальный алгоритм $\mathfrak{R}_1$ в четырехбуквенном алфавите A_1, вполне эквивалентный алгоритму $\mathfrak{R}$ относительно алфавита A_0. Это значит, что для любого слова $X \in A_0^*$ имеем $!\mathfrak{R}(X) \Leftrightarrow !\mathfrak{R}_1(X)$. Для алгоритма $\mathfrak{R}_1$ можно построить его естественное расширение $\mathfrak{R}_2$ на алфавит A, так что для любого слова $X \in A_0^*$ имеем $!\mathfrak{R}_1(X) \Leftrightarrow !\mathfrak{R}_2(X)$. Мы получили, что алгоритм $\mathfrak{R}_2$ применим к тем и только к тем записям в A_0 нормальных алгорифмов в алфавите A, которые являются несамоприменимыми. Но такого нормального алгорифма не существует. Значит, и нормального алгорифма типа алгоритма $\mathfrak{R}$ тоже не существует. ▶

Существование нормального алгорифма со свойством, описанным выше, все равно не решило бы проблему распознавания самоприменимости (несамоприменимости), поскольку для несамоприменимых алгоритмов наш алгоритм $\mathfrak{R}$ давал бы ответ, а для самоприменимых — нет, так как для таких алгоритмов он будет работать неограниченно долго. Модифицируем постановку задачи. ***Массовой проблемой распознавания несамоприменимости*** назовем проблему существования алгоритма $\mathfrak{P}$, применимого к записи в A_0 любого нормального алгорифма в алфавите A и удовлетворяющего условию: $\mathfrak{P}$ преобразует запись нормального алгорифма в пустое слово, если этот нормальный алгорифм несамоприменим, и в непустое слово в простивном случае. Оказывается, что эта массовая алгоритмическая проблема неразрешима.

Теорема 10.3. Пусть A содержит не менее четырех букв и $A_0 \subset A$ — двухбуквенный подалфавит. Не существует нормального алгорифма $\mathfrak{P}$ над алфавитом A_0, который дает пустое значение на слове $X \in A_0^*$ тогда и только тогда, когда X есть запись несамоприменимого нормального алгорифма.

◀ На самом деле это модификация предыдущей теоремы. Действительно, пусть нормальный алгорифм $\mathfrak{P}$ существует. Тогда его легко модифицировать, зацикливая в тех случаях, когда результатом является непустое слово (достаточно взять композицию $\mathfrak{P}$ с нормальным алгорифмом, описываемым групповой подстановкой $\xi \to \xi$, где ξ пробегает весь алфавит алгоритма $\mathfrak{P}$). В результате получим нормальный алгорифм с теми же свойствами, что и алгоритм $\mathfrak{R}$ из теоремы 10.2. Но такого алгоритма не существует. Значит, не существует и алгоритма $\mathfrak{P}$. ▶

Доказанная теорема, в частности, утверждает и отсутствие нормального алгорифма, применимого ко всем записям нормальных алгорифмов в алфавите[*] A, но дающего в результате пустое

[*]В соответствии с формулировкой теоремы нормальный алгорифм $\mathfrak{P}$ применим к записям всех несамоприменимых алгорфмов и дает при этом пустое слово, а к записям других нормальных алгорифмов он может быть и неприменим, но если применим, то дает в результате непустое слово.

слово только для несамоприменимых нормальных алгорифмов. Это и означает, что массовая проблема распознавания несамоприменимости неразрешима. Нетрудно также сделать вывод о неразрешимости ***массовой проблемы распознавания самоприменимости***, поскольку из соответствующего алгоритма легко получить алгоритм распознавания несамоприменимости.

10.2. Проблема распознавания применимости алгоритма к слову

Схожая проблема — существование алгоритма, распознающего применимость данного алгоритма к произвольному слову. Пусть дан нормальный алгорифм $\mathfrak{A}$ над алфавитом A. Поставим вопрос: существует ли нормальный алгорифм $\mathfrak{P}$, который применим к любому слову в алфавите A, и дающий в результате пустое слово тогда и только тогда, когда к исходному слову применим (неприменим) нормальный алгорифм $\mathfrak{A}$?

Суть проблемы в следующем. Конечно, можно применить сам алгоритм $\mathfrak{A}$ к слову X. Однако это даст ответ только в случае, когда алгоритм $\mathfrak{A}$ применим к этому слову X. Неприменимость означает, что $\mathfrak{A}$ будет работать вечно, а мы на каждом шаге не знаем, остановится он через пару шагов или нет.

Разумеется, есть алгоритмы, для которых ответ очевиден. Например, тождественный алгоритм применим к любому слову, построение для него распознающего алгоритма тривиально: распознающий алгоритм должен всегда возвращать Λ. Но для всех ли алгоритмов имеется такое распознавание?

Теорема 10.4. Существует нормальный алгорифм $\mathfrak{C}$ над пятибуквенным алфавитом A_2, обладающий свойством: не существует нормального алгорифма $\mathfrak{P}$ над A_2, который преобразует в Λ те и только те слова в алфавите A_2, к которым неприменим $\mathfrak{C}$.

◀ Доказательство основано на существовании универсального алгорифма. Пусть $A_2 = a_0a_1\alpha\beta\delta$, полагаем $A_0 = a_0a_1$, $A_1 = A_0\alpha\beta$. Рассмотрим универсальный алгорифм $\mathfrak{R}$ для алфавита A_2. Мы можем его трасформировать в нормальный алгорифм $\mathfrak{C}$, в котором используются не изображения нормальных алгорфмов, а их записи в алфавите A_0. Действительно, нетрудно построить нормальный алгорифм $\mathfrak{S}_з$, переводящий запись нормального алгорифма в его изображение. Тогда композиция $\mathfrak{C} = \mathfrak{R} \circ \mathfrak{S}_з$ будет обладать свойством $\mathfrak{C}(\mathfrak{A}^з\delta X) \simeq \mathfrak{R}(\mathfrak{A}^и\delta X)$.

Пусть для нормального алгорифма $\mathfrak{C}$ существует распознающий нормальный алгорифм $\mathfrak{P}$, т.е. $\mathfrak{P}(Y) = \Lambda \Leftrightarrow \neg!\mathfrak{C}(Y)$. В качестве Y рассмотрим слова вида $\mathfrak{A}^з\delta\mathfrak{A}^з$ по всем нормальным алгорифмам $\mathfrak{A}$ в алфавите A_2. Мы заключаем, что $\mathfrak{P}(\mathfrak{A}^з\delta\mathfrak{A}^з) = \Lambda \Leftrightarrow \neg!\mathfrak{C}(\mathfrak{A}^з\delta\mathfrak{A}^з)$. Но $\mathfrak{C}(\mathfrak{A}^з\delta\mathfrak{A}^з) \simeq$ $\simeq \mathfrak{A}(\mathfrak{A}^з)$. Поэтому $\mathfrak{P}(\mathfrak{A}^з\delta\mathfrak{A}^з) = \Lambda \Leftrightarrow \neg!\mathfrak{A}(\mathfrak{A}^з)$. Трансформируем алгоритм $\mathfrak{P}$, взяв его композицию с алгоритмом $\mathfrak{S}_2$, который удваивает запись нормального алгорифма: $\mathfrak{S}_2(\mathfrak{A}^з) = \mathfrak{A}^з\delta\mathfrak{A}^з$. Получим нормальный алгорифм $\mathfrak{P}_1 = \mathfrak{P} \circ \mathfrak{S}_2$, обладающий свойством: $\mathfrak{P}_1(\mathfrak{A}^з) = \Lambda \Leftrightarrow \neg!\mathfrak{A}(\mathfrak{A}^з)$, т.е. нормальный алгорифм $\mathfrak{P}_1$ распознает свойство несамоприменимости. Согласно теореме 10.3 такого алгоритма не существует. Значит, предположение о существовании нормального алгорифма $\mathfrak{P}$ неверно, что и доказывает теорему. ▶

Замечание 10.1. Построенный нами нормальный алгорифм $\mathfrak{C}$ имеет большой алфавит: кроме основного алфавита A_2 он использует значительное количество спецсимволов. Однако его можно перевести в эквивалентный нормальный алгорифм в двухбуквенном алфавите, переводя также и исходные слова. В результате мы получаем нормальный алгорифм в двухбуквенном алфавите, применимость которого к словам в этом алфавите не распознается.

Из доказанной теоремы выекают следующие утверждения.

Следствие 10.1. Существует нормальный алгорифм $\mathfrak{C}$ над пятибуквенным алфавитом A_2, обладающий свойством: не существует нормального алгорифма $\mathfrak{P}$ над A_2, применимого ко

всякому слову в A_2 и преобразующего в Λ те и только те слова в алфавите A_2, к которым неприменим $\mathfrak{C}$.

Следствие 10.2. Существует нормальный алгорифм $\mathfrak{C}$ над пятибуквенным алфавитом A_2, обладающий свойством: не существует нормального алгорифма $\mathfrak{P}$ над A_2, применимого ко всякому слову в A_2 и преобразующего в Λ те и только те слова в алфавите A_2, к которым применим $\mathfrak{C}$.

Эти следствия устанавливают неразрешимость проблемы распознавания применимости (неприменимости) алгоритма к слову. Отметим еще один аспект, связанный с этими следствиями. Любой нормальный алгорифм естественно ассоциировать с частично рекурсивной функцией (такая ассоциация возникает в результате процесса арифметизации нормальных алгоритмов). При этом нормальный алгорифм, применимый к любому слову и дающий результат в виде пустого или непустого слова, ассоциируется с общерекурсивным предикатом. При такой трактовке нетрудно увидеть следующую интерпретацию сформулированных следствий: существует рекурсивно перечислимое множество (область применения нормального алгорифма $\mathfrak{C}$), дополнение к которому не является разрешимым, и существует рекурсивно перечислимое множество, не являющееся разрешимым.

Близка к рассмотренной и проблема распознавания аннулирования, состоящая в следующем: существует ли для данного алгоритма $\mathfrak{A}$ над алфавитом A такой нормальный алгорифм $\mathfrak{R}$, который применим к любому слову в A и аннулирующий (преобразующий в пустое слово) те и только те слова алфавита A, которые аннулирует $\mathfrak{A}$.

Простой механизм разветвления позволяет алгоритм $\mathfrak{C}$ над алфавитом A_2 превратить в алгоритм $\mathfrak{B}$, который аннулирует те и только те слова, для которых $\mathfrak{C}$ применим (т.е. либо дает пустое слово, либо неприменим). Достаточно взять композицию алгоритма $\mathfrak{C}$ с алгоритмом, аннулирующим любое слово. В результате проблема распознавания аннулирования для такого алгоритма будет равносильна проблеме распознавания применимости, а последняя не имеет решения.

Множество всех частично рекурсивных функций счетно. Это можно усмотреть разными способами. Например, можно вспомнить о существовании универсальных функций, которое вытекает из теоремы Клини о нормальной форме. Ограничимся функциями одного переменного. Существует такая частично рекурсивная функция U^2, что множество функций $U^2(n,\cdot)$ совпадает с множеством всех частично рекурсивных функций одного переменного. Универсальную функцию можно рассматривать как нумерующую: каждому номеру соответствует некоторая функция и таким способом охватываются все частично рекурсивные функции. Отметим, что этот способ нумерации не гарантирует единственность номера.

Однако важно не только по номеру вычислять функцию, но и наоборот, по функции находить один из ее номеров. А как идентифицировать функцию? Мы не можем этого сделать по ее действию, поскольку для этого надо выполнить бесконечное количество проверок. Два этих момента связаны.

Идентифицировать функции можно следующим образом. Каждую функцию можно связать с нормальным алгорифмом. Нормальный алгорифм в свою очередь можно записать в двухбуквенном алфавите. Действительно, можно ограничиться счетным набором букв и считать, что алфавиты всех схем черпаются из этого множества. Все эти буквы можно закодировать в двухбуквенном алфавите. В результате все нормальные алгорифмы окажутся представленными словами в двухбуквенном алфавите. С каждым нормальным алгорифмом связана частично рекурсивная функция. В результате перенумерации всех нормальных алгорифмов получим нумерацию частично рекурсивных функций. Можно показать, что существует алгоритм, который по номеру определяет нормальный алгорифм, затем соответствующую частично рекурсивную функцию и вычисляет ее значение, т.е. мы имеем алгоритм вычисления функции двух переменных, которая оказывается универсальной.

Чтобы перенумеровать все алгоритмы, достаточно устроить нумерацию всех слов по мере возрастания их длины, а путем проверки, что это запись нормального алгорифма (по формальным признакам), пропустить все лишние слова и тем самым получить нумерацию всех нормальных алгорифмов. В дальнейшем будем рассматривать именно такие нумерации.

Теорема 10.5. Для любой частично рекурсивной функции f^2 существует такая общерекурсивная функция $s(x)$, что $f(x,y) = U^2(s(x), y)$.

◀ Существует нормальный алгорифм $\mathfrak{A}$, вычисляющий функцию $f(x,y)$. Идея такая. Для каждого x мы имеем функцию одного переменного $f(x,\cdot)$. Модификацией алгоритма $\mathfrak{A}$ можно получить нормальный алгорифм, вычисляющий функцию $f(x,\cdot)$: достаточно к исходному слову 01^y0 слева добавить аргумент x и применить алгоритм[**] $\mathfrak{A}$. В результате мы получаем алгоритм, который по первому аргументу x определяет номер соответствующего алгоритма и тем самым номер $s(x)$. ▶

Следствие 10.3. Существует общерекурсивная функция $s(n,x)$, для которой

$$U^3(n,x,y) = U^2(s(n,x), y),$$

где U^3 — нумерация всех частично рекурсивных функций двух переменных.

◀ Алгоритм, который по $f(x,y)$ вычисляет ее функцию номера $s(x)$, несложно модифицировать следующим образом: к числу y добавляем пару чисел n и x и применяем алгоритм вычисления U^3. Эту процедуру оформляем в виде записи и по ней определяем номер $s(n,x)$ значение y на самом деле здесь не нужно. В результате есть алгоритм, вычисляющий $s(n,x)$, а это значит, что эта функция общерекурсивна. ▶

Установим следующий факт.

Теорема 10.6 (Клини). Для любой частично рекурсивной функции $g(x)$ существует число a такое, что $U^2(g(a),x) = U^2(a,x)$.

◀ Рассмотрим вспомогательную функцию $f(y,x) = U^2(g(s(y,y)),x)$, где s — функция, существование которой утверждается следствием 10.3. Поскольку $f(y,x)$ частично рекурсивна, существует номер n, при котором $f(y,x) = U^3(n,y,x)$. В результате $U^2(g(s(y,y)),x) = U^3(n,y,x)$. Согласно следствию 10.3 $U^3(n,y,x) = U^2(s(n,y),x)$. Поэтому $U^2(g(s(y,y)),x) = U^2(s(n,y),x)$. Положив $y = n$ и обозначив $a = s(n,n)$ с учетом общерекурсивности s получаем $U^2(g(a),x) = U^2(a,x)$. ▶

Доказанная теорема позволяет установить удивительный факт. Предположим, что есть свойство P, которым может обладать или не обладать частично рекурсивная функция одного переменного. Опустим очевидные случаи, когда свойством обладают все функции или, наоборот, ни одна из функций одного переменного. Поставим вопрос: существует ли алгоритм, который по номеру l устанавливает, обладает ли свойством P функция $U^2(l,\cdot)$? Оказывается, что эта алгоритмическая проблема неразрешима.

Тезис о свойстве здесь можно опустить: свойство определяет некоторое подмножество в множестве всех частично рекурсивных функций одного переменного, причем это подмножество собственное (не пусто и не есть все множество). Существование распознающего алгоритма равносильно существованию общерекурсивного предиката, вычисляемого алгоритмом, который для данного x принимает значение 1, если функция с номером x обладает свойством P, и значение 0 в противном случае. Иначе говоря, разрешимость поставленной проблемы равносильна тому, что множество номеров всех частично рекурсивных функций, обладающих свойством

[**]Правда, здесь есть тонкость: есть часть алгоритмов, работающих с системами чисел и тем самым вычисляющих функцию, а есть процедура арифметизации, которая каждый алгоритм трансформирует в функцию. Надо, чтобы при этом характер первых алгоритмов не изменился.

P, разрешимо. Отметим, что множество самих частично рекурсивных функций может быть разрешимым, например, если оно конечно.

Теорема 10.7 (Райс). Пусть P — собственное подмножество множества всех частично рекурсивных функций одного переменного. При любой нумерации частично рекурсивных функций множество номеров функций, принадлежащих P, не разрешимо.

◀ Предположим, что множество номеров $N(P)$ функций, принадлежащих P, разрешимо. Тогда и дополнение $\overline{N(P)}$ разрешимо, т.е. обе характеристические функции χ_N и $\chi_{\overline{N}}$ множеств $N(P)$ и $\overline{N(P)}$ являются общерекурсивными. По условию оба множества не пусты, поэтому можно выбрать $\alpha \in N(P)$ и $\beta \notin N(P)$. Рассмотрим функцию

$$g(x) = \begin{cases} \alpha, & x \notin N(P); \\ \beta, & x \in N(P). \end{cases}$$

В силу представления $g(x) = \alpha\chi_{\overline{N}}(x) + \beta\chi_N(x)$ делаем заключение, что функция $g(x)$ общерекурсивна.

По функции $g(x)$ выберем число a в соответствии с теоремой Клини и рассмотрим функцию $f(x) = U(g(a), x)$, которая, очевидно, частично рекурсивна. Интересно, принадлежит ли число a множеству $N(P)$?

Предположим, что $a \in N(P)$. Тогда $g(a) = \beta \notin N(P)$. В результате, с одной стороны, $U(a, \cdot) \in P$, а с другой, $U(a, \cdot) = U(g(a), \cdot) = U(\beta, \cdot) \notin P$. Предположение $a \notin P$ также приводит к противоречию. Следовательно, исходное предположение о разрешимости $N(p)$ неверно. ▶

11. СЛОЖНОСТЬ АЛГОРИТМОВ

Богатая практика программирования в наше время естественным образом формирует представления о сложности программы. Мы бы сказали, что более сложной является программа, потребляющая большее количество ресурсов. Ключевыми ресурсами в современной вычислительной практике являются память и время. Это и ложится в основу понятий со сложности алгоритмов.

Заметим, что используемую программой память можно разделить на две части: память для размещения самой программы и память для размещения данных. Мы сосредоточимся только на 1-й части, т.е. на оценке емкости самой программы.

Разумеется, используемые ресурсы зависят от того, в какой среде реализуется программа. В контексте теории алгоритмов можно сказать так: сложность алгоритма зависит от используемой вычислительной модели. Мы коснемся двух вычислительных моделей: нормальных алгорифмов и одноленточных машин Тьюринга.

Рекурсивные функции стоят здесь особняком, поскольку механизм рекурсии можно уподобить неалгоритмическому описанию соответствующего преобразования. Подобные описания нашли свое применение в практике программирования. Эту вычислительную модель в контексте сложности мы обсуждать не будем.

11.1. Сложность нормальных алгорифмов

Сложностью нормального алгорифма $S(\mathfrak{A})$ называется количество знаков в изображении этого алгорифма, т.е. $S(\mathfrak{A}) = |\mathfrak{A}^{\text{и}}|$. ***Сложность времени работы*** $t_{\mathfrak{A}}(X)$ алгоритма $\mathfrak{A}$ над словом X — это число шагов работы алгоритма для данного слова X. Если алгоритм неприменим к слову X, считаем, что $t_{\mathfrak{A}}(X) = \infty$.

Рассмотрим проблему распознавания свойства P, которым могут обладать или не обладать слова в алфавите A. Такая проблема в целом неразрешима. Однако разрешимой может быть ***ограниченная проблема*** — проблема существования алгоритма, распознающего свойство P для слов длины не более n. Отсутствие алгоритма для всего множества слов, в частности, вытекает из того, что сложность алгоритма, решающего ограниченную проблему, растет с ростом предельной длины слова.

Назовем верхней оценкой сложности распознавания свойства P такую функцию $f(n)$, что для каждого n существует нормальный алгорифм, решающий n-ограниченную проблему распознавания свойства P и имеющий сложность не выше $f(n)$. Нижней оценкой назовем такую функцию $g(n)$, что ни один алгоритм, решающий n-ограниченную проблему распознавания, не может иметь сложность менее $g(n)$.

Простенькая теорема о сложности распознавания самоприменимости показывает, почему эта массовая проблема неразрешима.

Теорема 11.1. Пусть N — натуральное число и $\mathfrak{R}$ — нормальный алгорифм в алфавите A, имеющем не менее четырех букв, применимый к записи нормального алгорифма $\mathfrak{A}$ в алфавите A сложности $S(\mathfrak{A}) \leqslant N$ тогда и только тогда, когда $\mathfrak{A}$ несамоприменим. Тогда $S(\mathfrak{R}) > N$.

◀ Если $S(\mathfrak{R}) \leqslant N$, то его можно применить к своей записи. При этом согласно условию, если он несамоприменим, то он применим к своей записи, т.е. самоприменим и, наоборот, если он самоприменим, то он неприменим к своей записи, т.е. несамоприменим. Противоречие показывает, что на самом деле $S(\mathfrak{R}) > N$. ▶

Переходим к оценке сложности проблемы распознавания несамоприменимости. Зафиксируем четырехбуквенный алфавит $A_1 = abcd$. Для произвольного нормального алгорифма $\mathfrak{A}$ в алфавите A_1 построим алгоритм $\mathfrak{B}$ в алфавите $A_1 e$ со схемой

$$\begin{cases} e\xi \to\cdot\ \Lambda, \\ \xi e \to\cdot\ \Lambda, \\ e \to e, \\ \mathfrak{A}_e, \end{cases}$$

где $\mathfrak{A}_e$ — нормальный алгорифм, полученный из замыкания $\mathfrak{A}$ заменой каждой терминальной подстановки $P \to\cdot\ Q$ подстановкой $P \to eQ$.

Нетрудно увидеть, что явно построена композиция двух алгоритмов, причем второй есть „зацикливатель“ $\xi \to\cdot\ \Lambda$, $\Lambda \to \Lambda$. Алгоритм $\mathfrak{B}$ применим к слову X в A_1 тогда и только тогда, когда $\mathfrak{A}(X) \neq \Lambda$. Устроим перевод $\mathfrak{B}^\tau$ алгоритма $\mathfrak{B}$ в алфавит A_1, используя в качестве базы $A_0 = ab$ и переводя остальные буквы по правилам $c \to cdc$, $d \to cd^2c$, $e \to cd^3c$. Оценим сложность $\mathfrak{B}^\tau$ через сложность $\mathfrak{A}$, полагая, что схема алгоритма $\mathfrak{A}$ имеет k подстановок.

Число букв алфавита A_1 в схеме $\mathfrak{A}$ можно подсчитать, вычтя все служебные символы: k символов α или β и $k+1$ символ γ. Получается $S(\mathfrak{A}) - (2k+1)$. Схема алгоритма $\mathfrak{A}_e$ содержит дополнительную подстановку, но число символов алфавита A_1 будет то же самое. При переходе к переводу этого алгорифма каждая буква алфавита A_1 заменяется словом длины не более 4. Кроме того, в этом переводе будет до $k+1$ букв e, каждая из которых заменяется словом длины 5. В результате получим оценку

$$4(S(\mathfrak{A}) - (2k+1)) + 5(k+1) + 2k + 3 = 4S(\mathfrak{A}) - k + 4.$$

Первые три строки (фактически это 9 подстановок) добавляют еще $(5 \cdot 4 + 4 + 3 + 1 + 1 + 4) \cdot 2 + {} + 11 + 9 = 86$ букв. В результате

$$S(\mathfrak{B}^\tau) \leqslant 4S(\mathfrak{A}) - k + 90 \leqslant 4S(\mathfrak{A}) + 89. \tag{11.1}$$

Полученная оценка позволяет доказать следующую теорему.

Теорема 11.2. Пусть натуральное число N и нормальный алгорифм $\mathfrak{C}$ в A_1 таковы, что $\mathfrak{C}$ применим к записи всякого алгоритма $\mathfrak{A}$ в A_1 с $S(\mathfrak{A}) \leqslant N$, причем $\mathfrak{C}(\mathfrak{A}^3) = \Lambda$ тогда и только тогда, когда $\mathfrak{A}$ несамоприменим. Тогда $S(\mathfrak{C}) > \dfrac{N-89}{4}$.

◀ По алгоритму $\mathfrak{C}$ построим алгоритм $\mathfrak{B}^\tau$, как описано выше. Тогда алгоритм $\mathfrak{B}^\tau$ будет неприменим к записи алгоритма $\mathfrak{A}$ с $S(\mathfrak{A}) \leqslant N$ тогда и только тогда, когда $\mathfrak{C} = \Lambda$, т.е. когда $\mathfrak{A}$ несамоприменим. По доказанному в теореме 11.1 $S(\mathfrak{B}^\tau) > N$, а согласно неравенству (11.1) получаем

$$S(\mathfrak{C}) \geqslant \frac{S(\mathfrak{B}^\tau) - 89}{4} > \frac{N-89}{4}.$$

что и требовалось доказать. ▶

Замечание 11.1. Из доказанной теоремы немедленно вытекает, что общая проблема несамоприменимости неразрешима: для разрешающего алгоритма $\mathfrak{C}$, если он существует, получаем, что $S(\mathfrak{C}) > \dfrac{N-89}{4}$ для любого натурального N, а это невозможно.

Замечание 11.2. Аналогичным образом можно получить оценку для алгоритмов, решающих ограниченную проблему самоприменимости. Для этого взамен алгоритма $\mathfrak{B}$ строим алгоритм $\mathfrak{B}'$ со схемой

$$\begin{cases} e\xi \to e\xi, \\ \xi e \to \xi e, \\ e \to\cdot\ \Lambda, \\ \mathfrak{A}_e, \end{cases}$$

Первые четыре строки в схеме дают $[2\cdot(5\cdot 4+4+3+1+1)+4]\cdot 2+6+9=139$ символов (было 86). Поэтому, повторяя рассуждения, заключаем, что нормальный алгорифм $\mathfrak{C}'$, решающий N-ограниченную проблему самоприменимости, имеет оценку сложности

$$S(\mathfrak{C}')>\frac{N-142}{4}.$$

Скажем, что нормальный алгорифм $\mathfrak{C}$ над алфавитом A решает N-ограниченную проблему применимости нормального алгорифма $\mathfrak{A}$ к словам в алфавите A, если для всякого слова $X\in A^*$ длины не более N, во-первых, $!\mathfrak{C}(X)$, а во-вторых, $\mathfrak{C}(X)=\Lambda\Leftrightarrow\,!\mathfrak{A}(X)$.

Напомним, что проблема применимости была сведена к проблеме несамоприменимости. Уточним это сведение, подсчитав соответствующее количество символов. Напомним, что, видоизменив универсальный алгорифм $\mathfrak{R}$, мы построили нормальный алгорифм $\mathfrak{R}_1$ над алфавитом A_1e, который удовлетворяет условию $\mathfrak{R}_1(\mathfrak{A}^з eX)\simeq\mathfrak{A}(X)$. Очевидная модификация с соединением приводит к нормальному алгорифму $\mathfrak{R}_2$, для которого $\mathfrak{R}_2(X)\simeq\mathfrak{R}_1(XeX)$. Для этого алгоритма, в частности, $\mathfrak{R}_2(\mathfrak{A}^з)\simeq\mathfrak{A}(\mathfrak{A}^з)$, т.е. алгоритм $\mathfrak{R}_2$ распознает самоприменимость алгоритма. Осталось перевести его в алгоритм $\mathfrak{R}_3$ в алфавите A_1, имея в виду, что записи нормальных алгорифмов используют только буквы a и b.

При отсутствии ограничений на длину алгоритма эта конструкция приводила к противоречию, поскольку допускалось рассматривать применимость построенного алгоритма к самому себе. Однако введение на ограничение длины записи эту проблему снимает и мы лишь можем утверждать, что запись построенного алгоритма не проходит по длине.

Теорема 11.3. Пусть N — произвольное натуральное число. Каков бы ни был алгоритм $\mathfrak{C}$ в алфавите A_1, решающий N-ограниченную проблему неприменимости алгоритма $\mathfrak{R}_3$ к словам в алфавите A_0, верна оценка

$$S(\mathfrak{C})>\frac{N}{36}-\frac{45}{2}.$$

◀ Если нормальный алгорифм $\mathfrak{C}$ решает проблему неприменимости алгоритма $\mathfrak{R}_3$ к словам в алфавите A_0, то, в частности, он применим к записи любого алгоритма с длиной записи не более N, причем $\mathfrak{C}(\mathfrak{A}^з)=\Lambda\Leftrightarrow\neg!\mathfrak{A}(\mathfrak{A}^з)$.

Отметим, что для любого нормального алгорифма $\mathfrak{A}$ в алфавите A_1 имеем $|\mathfrak{A}^з|\leqslant 9|\mathfrak{A}^и|$, поскольку изображение строится в семибуквенном алфавите $abcd\alpha\beta\gamma$ и при переводе семи букв в двухбуквенный алфавит каждая будет записана максимум 9 буквами.

Пусть $\mathfrak{A}$ — нормальный алгорифм в a_1 с длиной изображения (сложностью) не более $\left[\frac{N}{9}\right]$ (целой части дроби). Тогда $|\mathfrak{A}^з|\leqslant N$. Следовательно, $!\mathfrak{C}(\mathfrak{A}^з)$ и $\mathfrak{C}(\mathfrak{A}^з)=\Lambda\Leftrightarrow\neg!\mathfrak{A}(\mathfrak{A}^з)$. Мы видим, что $\mathfrak{C}$ решает K-ограниченную проблему несамоприменимости с $K=\left[\frac{N}{9}\right]$. Поэтому

$$S(\mathfrak{C})>\frac{K-89}{4}\geqslant\frac{1}{4}\Big(\frac{N}{9}-1-89\Big)=\frac{N}{36}-\frac{45}{2},$$

что и требовалось доказать. ▶

Замечание 11.3. Учитывая замечание 11.2, нетрудно получить оценку сложности для алгоритма, решающего N-ограниченную проблему применимости к слову:

$$S(\mathfrak{C}')>\frac{N}{36}-\frac{143}{4}.$$

Теорема 11.3 и замечание 11.3 устанавливают нижнюю оценку для ограниченной проблемы неприменимости или применимости к слову. Существуют и верхние оценки, имеющие тот порядок роста, что и нижние оценки.

Теорема 11.4. Каков бы ни был нормальный алгорифм $\mathfrak{R}$ над алфавитом A_0, существует такая постоянная k, что для любого N существует нормальный алгорифм $\mathfrak{C}$ сложности не более $N+k$, решающий N-ограниченную проблему применимости $\mathfrak{R}$ к словам алфавита A_0.

11.2. Сложность машин Тьюринга

Пусть задана машина Тьюринга T с внешним алфавитом $A = a_0a_1 \dots a_n$, и внутренним алфавитом $Q = q_0q_1 \dots q_m$. ***Сложностью*** $S(T)$ ***машины Тьюринга*** T называется максимальная емкость программы, равная $m(n+1)$. ***Временно́й сложностью*** (сложностью времени работы) $t_T(K)$ на конфигурации K называется количество шагов, сделанных машиной Тьюринга начиная с конфигурации K до завершения работы. Если машина Тьюринга неприменима к данной конфигурации, то считаем, что $t_T(K) = \infty$.

Для данной вычислимой словарной функции f можно указать много машин Тьюринга, ее вычисляющих. В каких пределах меняется сложность этих машин? Назовем число l ***верхней оценкой сложности*** для функции f, если существует машина Тьюринга, вычисляющая f, сложности не выше l. Число k называется ***нижней оценкой сложности***, если любая машина Тьюринга, вычисляющая f, имеет сложность не менее k.

При отсутствии алгоритма, решающего ту или иную массовую проблему, можно рассматривать серию алгоритмов, решающих ограниченную массовую проблему. Допустим, что решение массовой проблемы выражается как вычисление некоторой словарной функции $f(X)$. Отсутствие алгоритма означает, что эта функция невычислима. Рассмотрим N-ограниченную проблему, состоящую в вычислении функции $f(X)$ для слов длины $|X| \leqslant N$. Таких слов в данном алфавите конечное число и, следовательно, ограниченная проблема разрешима. Разрешающих алгоритмов может быть много. Скажем, что функция $h(N)$ есть верхняя оценка сложности вычисления функции $f(X)$, если для любого N существует машина Тьюринга, вычисляющая $f(X)$ при $|X| \leqslant N$ и имеющая сложность не выше $h(N)$. Нижней оценкой сложности вычисления функции $f(X)$ называется любая функция $g(N)$, такая, что для любая машина Тьюринга, решающая N-ограниченную проблему вычисления $f(X)$, имеет сложность не ниже $g(N)$. Аналогичные определения вводятся и для временной сложности.

По поводу временно́й сложности отметим следующий факт.

Теорема 11.5 (теорема Блюма об ускорении). Пусть r — общерекурсивная функция. Существует общерекурсивная функция f, такая, что для любой машины Тьюринга T, вычисляющей f, существует машина Тьюринга T_1, также вычисляющая f, для которой $r(t_{T_1}(n)) < t_T(n)$ начиная с некоторого номера n.

Пусть $r(n) = 2^n$ (это, как легко показать, общерекурсивная функция). Существует функция f, обладающая следующим свойством. Если T — машина Тьюринга, вычисляющая f, то существует машина Тьюринга T_1, также вычисляющая f, для которой $t_{T_1}(n) < \log_2 t_T(n)$ при больших n. Далее, существует машина Тьюринга T_2, также вычисляющая f, для которой $t_{T_2}(n) < \log_2 t_{T_1}(n) < \log_2 \log_2 t_T(n)$ при больших n. Этот процесс можно продолжить.

11.3. Классы сложности P и NP

Теорема Блюма ставит крест на попытке ввести понятие оптимального, т.е. наименьшего по сложности, алгоритма. Развитие теории сложности пошло по пути выделения классов сложности алгоритмов. Во всех этих задачах рассматриваются так называемые ***распознающие алгоритмы***, которые применимы к любому слову и дают два возможных ответа 1 (да) и 0 нет. Этим алгоритмам соответствует понятие общерекурсивный предикат. Им можно придать различную трактовку: распознавание свойства, которым может обладать слово в данном алфавите, распознавание принадлежности слова множеству, проверка тех или иных утверждений о словах в данном алфавите и т.п.

Можно в самом общем виде так поставить массовую проблему. Есть бесконечное (счетное) множество однотипных задач, зависящих от некоторого набора параметров. Мы кодируем эти параметры с помощью некоторого набора символов (алфавита) и тем самым ставим каждой конкретной задаче слово в зафиксированном алфавите. Решение задачи — ответ „да“ или

„нет“. Мы приходим в результате к задаче вычислимости некоторого словарного предиката в заданном алфавите.

В значительной части литературы по этой тематике используют такую трактовку. Любое множество слов в данном алфавите называют ***языком***. Иначе говоря, язык в алфавите A — это любое подмножество $L \subset A^*$. Алгоритм T (машина Тьюринга или другая вычислительная модель) распознает язык L, если он применим к любому слову в данном алфавите и $T(X) = 1$ тогда и только тогда, когда $X \in L$. Напомним, что в этом контексте можно ограничиться двухбуквенным алфавитом.

Пусть задан алфавит A и словарная функция $f\colon A^* \to A^*$ вычислима по Тьюрингу. Скажем, что машина Тьюринга T, вычисляющая f, полиномиальная (вычисляет f за полиномиальное время), если существует такой полином $Q(t)$, что для любого слова $X \in A^*$ имеем $t_T(X) \leqslant Q(|X|)$, где $|X|$ — длина слова X. ***Словарная функция f полиномиальная*** (вычислима за полиномиальное время), если существует машина Тьюринга, вычисляющая f за полиномиальное время. Множество таких функций обозначим через Π.

Среди словарных функций выделим ***словарные предикаты*** — всюду определенные функции, принимающие лишь два значения 0 и 1. Словарный предикат представляет собой характеристическую функцию некоторого множества слов в рассматриваемом алфавите.

Скажем, что машина Тьюринга T распознает язык L за полиномиальное время, если она вычисляет характеристическую функцию этого языка за полиномиальное время. ***Язык L распознаваем за полиномиальное время***, если его характеристическая функция полиномиальная. Класс таких языков обозначим через P.

Разумеется, вычислимость словарного предиката не зависит от выбора вычислительной модели, но сложность для каждой модели своя. Например, можно поставить вопрос о временной сложности вычисления данного предиката в рамках нормальных алгорифмов и в рамках машин Тьюринга какой-либо модификации. Однако можно показать, что перенос алгоритма из одной вычислительной модели в другую (среди известных) приводит к полиномиальному увеличению сложности. Это значит, что класс P, формально вводимый для каждой вычислительной модели, на самом деле одинаков для широкого круга таких вычислительных моделей.

В качестве эталонных при оценке сложности выбирают одноленточные машины Тьюринга как более элементарные, например, по сравнению с нормальными алгорифмами. Поэтому те или иные оценки для машин Тьюринга получить легче.

Говорят о задачах полиномиальной сложности, а также о задачах экспоненциальной сложности — задачах, сложность которых не может быть оценена сверху никаким полиномом. Отметим, что к экспоненциальным относят, например, и задачи, временная сложность которых растет как $e^{\ln^2 n}$ (т.е. чуть выше полиномиальной). С практической точки зрения проблемы полиномиальной сложности отличает то, что при увеличения быстродействия в несколько раз, сложность задачи, решаемой за приемлемое время также повышается в несколько раз, т.е. здесь зависимость мультипликативная. При экспоненциальной сложности зависимость аддитивная.

Рассмотрим, к примеру, задачу проверки выполнимости КНФ. Можно выполнить 2^n операций вычисления значения КНФ на всевозможных наборах булевых переменных. И длина КНФ, записанной в том или ином алфавите, и алгоритм вычисления ее значения на конкретном наборе значений переменных полиномиально зависят от количества переменных. А в целом рассматриваемый алгоритм перебора значений имеет экспоненциальную сложность. Алгоритм полиномиальной сложности для этой задачи не известен. Однако ситуация меняется, если количество переменных в каждой элементарной дизъюнкции ограничено. Например, если в каждой элементарной дизъюнкции всего две переменных, то метод резолюций приводит к перебору не более $\frac{n(n+1)}{2}$ пар и даст ответ за полиномиальное время.

Еще одна задача — проверка, является ли отношение, заданное на конечном множестве, отношением эквивалентности. Здесь надо проверить три условия: симметричность, рефлексивность и транзитивность. Проверка каждого из этих условий имеет полиномиальную временную сложность. Таким образом, эта задача — из класса P.

Класс сложности NP аналогичен классу сложности P, но за основу взяты так называемые ***недетерминированные машины Тьюринга***. Такая машина устроена в целом так же, как и обычная, но для каждого состояния она позволяет несколько вариантов перехода, т.е. паре „состояние — обозреваемый символ“ соответствует несколько команд, и она выбирает любую.

Для недетерминированной машины Тьюринга для каждого слова (или для всех) есть как бы несколько вариантов ответа. Считаем, что результат ее работы 1, если хотя бы один ответ 1, иначе 0. Временем работы недетерминированной машины на данном слове называется либо шаг первого появления результата 1, либо шаг завершения последнего варианта, если все ответы 0.

Детерминированную машину Тьюринга можно рассматривать как частный случай недетерминированной. Из того, что представляет собой результат работы недетерминированной машины Тьюринга, заключаем, что речь идет не о выборе случайного варианта работы (для этого есть вероятностные машины), а об анализе все вариантов работы машины. Можно показать, что работу любой недетерминированной машины можно реализовать с помощью соответствующей детерминированной машины, которая при появлении нескольких вариантов на ленте дублирует информацию и параллельно ведет каждый из этих вариантов. Другими словами, недетерминированная машина — это теоретическое описание переборного алгоритма.

Хотя класс разрешимых проблем для детерминированной и недетерминированной машин совпадают, эти машины различаются по сложности вычислений. Класс NP — это класс проблем, для которых существует разрешающая недетерминированная машина Тьюринга полиномиальной временно́й сложности. Очевидно, что $P \subset NP$. Но какое это включение строгое или нет? Ответ не известен. Анализу этой проблемы посвящено довольно много усилий.

Недетерминированная машина — удобный способ описания переборных алгоритмов, таких как поиск в глубину в графе. При линейном переборе мы имеем дело с полиноимальной задачей. Но если перебор идет по дереву (например, выводимость в исчислении высказываний), то сложность возрастает до экспоненциальной.

Отметим альтернативное определение класса NP. Язык $L \in A^*$ принадлежит классу NP, если существует полином $p(x)$ и полиномиальная распознающая детерминированная машина Тьюринга T, такие, что $X \in L$ тогда и только тогда, когда существует слово Y, $|Y| \leqslant p(|X|)$, для которого $T(Y + X) = 1$.

Пусть $L \in NP$. Если $X \in L$, существует вариант работы недетерминированной машины, приводящий к значению 1. Достаточно записать в качестве слова Y информацию о правильный выборе при разветвлении в недетерминированной машине, а затем повторить ее работу по единственной ветке. Наоборот, пусть детерминированный алгоритм работает, если исходное слово „подправлено“. Тогда можно составить недетерминированный алгоритм полиномиальной сложности, формирующий „подсказку“.

Например, задача проверки выполнимости КНФ относится к классу NP, так как достаточно добавить конкретный набор значений переменных, при которых КНФ истинна, а затем запустить полиномиальный алгоритм вычисления значения КНФ на известном наборе значений переменных.

Принадлежность той или иной задачи данному классу можно попытаться установить непосредственно. Но чаще выгоднее свести такую задачу к принадлежности классу другого алгоритма. Будем говорить о языках. Скажем, что язык $L_1 \in A^*$ сводим к языку $L_2 \in A^*$, если существует вычислимая словарная функция f полиномиальной сложности, удовлетворяющая условию: $X \in L_1 \Leftrightarrow f(X) \in L_2$ (f — сводящая функция). Обозначаем $L_1 \preccurlyeq L_2$.

Если для L_2 существует распознающий алгоритм полиномиальной сложности, то для любого слова X мы сперва вычисляем $f(X)$, затрачивая полиномиальное время, а затем применяем распознающий алгоритм. Получаем распознающий L_1 алгоритм полиномиальной сложности. Другими словами, если $L_2 \in P$ и $L_1 \preccurlyeq L_2$, то и $L_1 \in P$. То же, очевидно, относится и к классу NP.

Отношение сводимости, очевидно, рефлексивное и транзитивное, но не антисимметричное (существуют разные языки, сводимые друг к другу). Скажем, что языки L_1 и L_2 эквивалентны, если $L_1 \preccurlyeq L_2$ и $L_2 \preccurlyeq L_1$. Получим отношение эквивалентности.

Язык $L \in NP$ называется NP-полным, если к нему сводится любой язык класса NP. Вопрос: существуют ли NP-полные проблемы, принадлежащие классу P? Этот вопрос эквивалентен равенству $P = NP$. Действительно, если существует NP-полный язык из P, то автоматически любой язык из NP попадает в P как сводимый к языку из P. Обратное утверждение очевидно. Пока таких языков не нашли. Существуют ли вообще NP-полные языки? Ответ да. Один из таких возникает в задаче проверки выполнимости КНФ. Множество выполнимых КНФ (а точнее их записей) обозначим SAT. Полнота этого языка объясняется тем, что любую переборную задачу можно оформить как проверку выполнимости (истинности хотя бы в одном варианте) некоторой КНФ.

ОГЛАВЛЕНИЕ