Министерство науки и высшего образования
Российской Федерации

Тверской государственный
университет

М. И. Дехтярь
С. М. Дудаков
Б. Н. Карлов

ЛЕКЦИИ ПО ДИСКРЕТНОЙ МАТЕМАТИКЕ

Учебник

Издание второе, переработанное и дополненное

Тверь — 2019

УДК 510 + 519
ББК 22.12 + 22.176 я73-1
Д 39

Тверской государственный университет
Факультет прикладной математики и кибернетики

Дехтярь М. И., Дудаков С. М., Карлов Б. Н.

Д 39 Лекции по дискретной математике : Учебник. — Издание второе, переработанное и дополненное. — Тверь : Твер. гос. ун-т, 2019. — 512 с.

Учебник содержит лекционный материал по дисциплине «Дискретная математика», а также примеры задач с решениями и задачи для самостоятельной работы. Основные разделы учебника: множества, математическая индукция, комбинаторика, булевы функции, логика высказываний и предикатов, графы, автоматы и формальные языки, алгоритмы.

Учебник адресован, прежде всего, студентам младших курсов, обучающихся по направлениям укрупненных групп 01.03.00 «Математика и механика», 02.03.00 «Компьютерные и информационные науки», 09.03.00 «Информатика и вычислительная техника».

УДК 510 + 519
ББК 22.12 + 22.176 я73-1

Посвящается памяти
нашего общего
учителя и коллеги
Михаила Иосифовича
Дехтяря

С. М. Дудаков
Б. Н. Карлов

Оглавление

Предисловие

Ко второму изданию

Данный учебник базируется на книге [7] «Лекции по дискретной математике», написанной Михаилом Иосифовичем Дехтярём в 2002–2007 гг. Прошло уже более 10 лет с момента её издания и в связи с опытом применения назрела необходимость внесения некоторых изменений.

В основном материал излагается в той же манере и последовательности, что и в [7]. Если не считать локальных редакторских и методологических изменений (иногда — значительных), то были внесены следующие правки:

1) Более подробно написан раздел, посвящённый множествам и их мощности.

2) Добавлен материал в раздел о логике высказываний и представлении информации с помощью неё.

3) Изложен популярный «табличный» метод построения многочленов Жегалкина.

4) Упрощено изложение логики предикатов за счёт исключения функций.

5) Общая глава 9, посвящённая ориентированным и неориентированным графам, разбита на две отдельные. В связи с этим изменилась нумерация следующих глав.

6) Добавлен материал об эйлеровых и плоских графах, раскраске графов.

7) Добавлен параграф о построении регулярного выражения по конечному автомату, что делает законченным доказательство эквивалентности этих классов языков.

8) Добавлен материал об общих свойствах операции кодирования, в связи с чем гомоморфизмы, гомоморфные образы и прообразы

языков стали всего лишь частным случаем. Включены параграфы, связанные с однозначностью и построением оптимального префиксного кодирования методом Хаффмана.

9) Вместо структурированных программ рассмотрены программы с метками. За счёт этого сильно упрощено доказательство эквивалентности программной вычислимости и вычислимости по Тьюрингу, а также появилась возможность лаконично доказать частичную рекурсивность программно вычислимых функций.

10) Добавлен материал об универсальной машине Тьюринга.

11) Часть материала из последней главы об эквивалентности различных формализаций алгоритма перенесена в предыдущие разделы.

12) Более подробно написаны параграфы об алгоритмически неразрешимых проблемах.

13) Для многих задач добавлены ответы, решения либо указания к решениям.

Программа годового курса, которой придерживаются авторы, включает материал глав 1–5, 7, 9–12, 15–17, 20–21, в связи с чем именно материал этих глав и подвергся наиболее глубокой переработке. Изменение состава этих глав, по сравнению с первым изданием было продиктовано следующими соображениями.

1) Неумение студентов первого курса корректно обращаться с простейшими конструкциями, описываемыми логикой предикатов (например, что будет отрицанием предложения «для всякого x найдётся y такой, что ...»?), потребовало уделить больше внимания этому вопросу.

2) Материал о схемах из функциональных элементов (глава 13) обязательно изучается в курсах наподобие «Архитектура ЭВМ», поэтому нет нужды его дублировать.

3) Материал главы 14 о бинарных диаграммах решений может быть изложен в курсах по искусственному интеллекту или автоматизированному принятию решений.

4) Материал главы 18 о программах с метками фактически дублирует материал других курсов.

5) Частично рекурсивные функции (глава 19) являются довольно специфическим методом определения алгоритмической вычислимости, который применяется в основном в теоретической информатике и математической логике. Поэтому, как нам представляется, изучать его всем студентам независимо от направления подготовки необязательно. Для тех, кто обучается по фундаментальным «компьютерным» или математическим направлениям, этот материал всё равно рассказывается далее в курсе «Математическая логика и теория алгоритмов» или подобном ему.

Разумеется, эти соображения являются личным видением авторов и у пользователей учебника могут быть совсем другие приоритеты.

Все замечания и предложения следует отправлять авторам по адресам электронной почты:

sergeydudakov@yandex.ru
bnkarlov@gmail.com

В качестве предисловия к первому изданию включено оригинальное предисловие М. И. Дехтяря к [7], в него внесены лишь редакторские правки.

К первому изданию

Данная книга содержит материал лекций курсов «Дискретная математика» и «Основы дискретной математики», читаемых на кафедре информатики Тверского государственного университета студентам первого курса, которые обучаются по направлению 510200 «Прикладная математика и информатика» (специальность 010200 «Прикладная математика и информатика») и по направлению «Математические методы и исследование операций в экономике» (специальность 061800 «Математические методы и исследование операций в экономике»).

Первый из них читается в течение двух семестров, второй — одного. Главная задача этих курсов — обучение студентов методам мышления, характерным для дискретной математики, основным понятиям таких её дисциплин, как булевы функции, логика предикатов, графы, конечные автоматы и алгоритмы. Поскольку обе указанные специальности предполагают умение эффективно решать задачи с помощью компьютера, была также поставлена цель развить у студентов навыки алгоритмического мышления на примерах решения задач из разных разделов дискретной математики. Программы обоих курсов были разработаны в конце 1990-х годов профессором М. А. Тайцлиным и затем подверглись некоторой корректировке автором книги, читавшим их в 2002–2007 гг.

Отметим, что, кроме переизданий известного учебника С. В. Яблонского [20], в последние годы было выпущено не менее дюжины книг отечественных и зарубежных авторов, в названиях которых фигурирует «Дискретная математика» (например, [9, 14]). Зачем же понадобилась ещё одна? Основная причина достаточно проста — хотелось снабдить студентов одним пособием, ориентированным на начальный уровень, то есть не требующим никаких специальных предварительных знаний, и полностью включающим материал всех изучаемых тем. Ещё одна цель — объединить в одной книге теоретический материал с достаточным количеством упражнений и задач, чтобы обеспечить проведение семинарских занятий и дать студентам пищу для самостоятельной работы. К сожалению, ни одно из имеющихся изданий этим условиям полностью не соответствует. Многие из них (особенно зарубежные) ориентированы на изучение дискретной математики на втором или третьем годах учёбы и требуют предварительного знания некоторых разделов высшей математики и программирования. В других нет достаточного количества задач, третьи — включают, в основном, теоретический материал, не доведённый до уровня алгоритмов и процедур. Но главное — ни одно из изданий не содержит всех существенных для нашей программы разделов. Например, во многих отсутствуют основы теории конечных автоматов

и алгоритмов, по которым в дальнейшем по данным специальностям не предусмотрены специальные курсы.

Содержание книги

Книга состоит из двадцати лекций. В лекции 1 приведены все необходимые для чтения и понимания книги сведения из элементарной теории множеств. Во второй кратко изложен метод математической индукции, который является основным методом доказательства в дискретной математике и интенсивно используется в остальных главах. В частности, отмечены особенности его применения для доказательства свойств объектов из индуктивно определённых классов (формул, выражений, программ и т. п.). В этой же лекции приводятся начальные сведения из комбинаторики: выводятся формулы для перестановок, размещений и сочетаний. Отдельный пункт посвящён доказательству принципа включений и исключений (в дальнейшем материале этот принцип явно не используется, и этот пункт можно опустить или перенести на практические занятия).

В лекциях 3–6 рассмотрен самый простой и важный класс дискретных функций — булевы функции. В лекции 3 описаны разные их представления: геометрическое, табличное и с помощью формул. На примерах показана связь между булевыми функциями и логикой высказываний. Лекция 4 посвящена эквивалентности формул. В ней приводятся основные логические тождества («законы логики») и показано, как они применяются для доказательства эквивалентности формул. Определены специальные классы формул для представления булевых функций: дизъюнктивные и конъюнктивные нормальные формы и многочлены Жегалкина. Описан метод Блейка для построения сокращённой дизъюнктивной нормальной формы и два способа построения многочленов Жегалкина: по произвольной формуле с помощью эквивалентных преобразований и по таблице с помощью метода неопределённых коэффициентов. В лекции 5 вводится понятие полной системы функций и доказывается теорема Поста, дающая необходимое и достаточное условие полноты. Лекция 6 содержит

менее традиционный для курсов дискретной математики материал: в ней рассматривается специальный подкласс булевых формул — хорновские формулы. Задача выводимости для формул этого класса, в отличие от формул общего вида, решается эффективно (за линейное время). Она нашла приложение во многих разделах информатики: в теории баз данных, логическом программировании, автоматическом синтезе программ, экспертных системах. Учитывая ориентацию курса «Основы дискретной математики» на математиков-экономистов, мы сформулировали её «экономическую» версию — задачу о возможности производства заданной продукции (набора товаров) из некоторого множества исходных продуктов (товаров, сырья) с помощью технологических процессов, задаваемых списками исходных и результирующих продуктов. Приведены два варианта алгоритма прямого вывода (от данных), эффективно решающего эту задачу: первый более прост для понимания, а второй его уточняет и работает в линейное время.

Лекция 7 представляет краткое введение в логику предикатов. Её главная цель — ознакомить читателей с этим языком (его синтаксисом и семантикой) и научить формально выражать на нём содержательные утверждения о свойствах объектов и отношений между ними. Лекция 8 посвящена связям между логикой предикатов и реляционными базами данных: языком запросов SQL и основными видами ограничений целостности.

Лекции 9–11 образуют введение в теорию графов[1]. В лекции 9 вводятся основные понятия теории графов. В частности, определены три представления графов: матрица смежности, матрица инцидентности и списки смежности, описаны процедуры построения графа достижимости (транзитивного замыкания), компонент сильной связности и всех баз ориентированного графа. В лекции 10 определены классы неориентированных и ориентированных деревьев. Отмечена связь деревьев и формул (выражений), описаны основные способы обхода

[1]Ещё раз напоминаем, что все лекции из [7], начиная с десятой, имеют в настоящей книге номер на единицу больший *(прим. авт.)*.

деревьев. В лекции 11 рассмотрены и проанализированы три классические задачи теории графов: построение минимального остова, обход всех вершин графа (задача о лабиринте) и задача о кратчайших путях. Для решения первой из них приведён простой вариант алгоритма Крускала, для второй — алгоритм поиска в глубину, позволяющий эффективно обходить все вершины графа, для третьей — алгоритм Дейкстры, строящий по заданной вершине нагруженного ориентированного графа дерево кратчайших путей из неё в остальные вершины.

Лекции 12 и 13 знакомят с двумя представлениями булевых функций с помощью специальных классов ориентированных графов без циклов: логическими схемами (схемами из функциональных элементов) (лекция 12) и упорядоченными бинарными диаграммами решений (УБДР) (лекция 13). Первое из них достаточно хорошо описано в отечественной литературе. Мы не включили никаких общих теорем о схемах, а лишь установили их связь с линейными программами и проиллюстрировали на примерах схем, реализующих функцию сложения по модулю 2 и сложение двоичных чисел. Хотя второе представление — с помощью УБДР — также весьма элементарно, оно появилось и исследовалось лишь в 90-х годах и, по-видимому, в русскоязычной учебной литературе пока подробно не описывалось.[2] Его включение в курс дискретной математики объясняется тем, что сейчас УБДР являются одним из основных средств реализации булевых функций от большого числа переменных в задачах искусственного интеллекта, проверки правильности электронных схем, программ, протоколов и т. п. В книге для них рассмотрена задача построения сокращённой (минимальной) УБДР по произвольной и задача синтеза сокращённой УБДР по формуле, представляющей функцию.

В лекциях 14–16 изложены основы теории конечных автоматов. Основные разновидности конечных автоматов — преобразователи и распознаватели (детерминированные и недетерминированные) — определяются в лекции 14. Там же приводится процедура детерминизации недетерминированных автоматов. Лекция 15 посвящена

[2]Краткое описание имеется в [10] *(прим. М. И. Дехтяря).*

языку регулярных выражений и его связи с конечными автоматами. Установлена теорема синтеза конечного автомата по заданному регулярному выражению. В лекции 16 рассмотрены свойства замкнутости класса автоматных языков относительно теоретико-множественных операций, конкатенации, итерации, гомоморфизмов и их обращений. Доказана теорема о разрастании для автоматных языков, и с её помощью продемонстрирована неавтоматность ряда языков.

Лекции 17–20 образуют краткое введение в теорию алгоритмов. В них определяются и сравниваются три формальных модели описания алгоритмов. В начале в лекции 17 вводится один из простейших формальных алгоритмических языков — простые структурированные программы. Он часто используется для изучения свойств программ в курсах информатики (см. [8, 17]). Две другие модели — это классические модели теории алгоритмов: описания частично рекурсивных функций (лекция 18) и машины Тьюринга (лекция 19). Приводятся типовые приёмы «программирования» и примеры вычислений арифметических функций в каждой из этих моделей. В лекции 20 посредством взаимного моделирования устанавливается, что все три модели определяют один и тот же класс функций. Это позволяет сформулировать тезис Тьюринга-Чёрча о совпадении этого класса функций с классом функций, вычислимых с помощью каких-либо алгоритмов. Объясняется, как строгое понятие алгоритма может позволить установить алгоритмическую неразрешимость некоторой проблемы. Доказывается неразрешимость проблем самоприменимости, останова, эквивалентности и некоторых других проблем, относящихся к свойствам структурированных программ.

О задачах и упражнениях

В книге помещено около 200 задач и упражнений, многие из которых состоят из нескольких независимых подзадач. Как правило, они расположены в отдельных параграфах в конце лекций. Условно эти задачи можно отнести к следующим категориям.

1) Упражнения на закрепление основного материала, в частности, задачи на «прокрутку» тех или иных процедур и алгоритмов, выполнение стандартных преобразований.
2) Задачи на доказательство (как правило, не очень сложное) отдельных утверждений из теорем и лемм основного текста. Желателен их разбор на практических занятиях.
3) Задачи на доказательство не отмеченных в основном тексте свойств определённых в нём понятий.
4) Задачи на установление свойств новых, не определённых в основном тексте понятий. Эти задачи расширяют материал книги и предназначены для самых любопытных читателей. Решения многих из них можно найти в литературе, рекомендуемой к соответствующим главам.

Хотя формально эта классификация в тексте не представлена, отнесение конкретной задачи к той или иной категории не должно вызывать трудностей. Точно так же не сделано никакое разбиение задач по степени сложности их решения. К задачам, казавшимся более сложными, помещены указания. Приводимые в большом количестве в основном тексте примеры должны служить образцами решения типовых задач первой категории.

О процедурах и алгоритмах

Решения большинства проблем, рассматриваемых в пособии, являются конструктивными. Как правило, они представлены в виде соответствующих процедур и алгоритмов. Для их описания использована достаточно неформальная версия псевдокода. При этом в первых главах шаги и действия процедур описываются, в основном, фразами на русском языке, а далее часто применяются программные конструкции:

- условный оператор «**Если** условие **то** действия **Иначе** действия»;
- цикл «**Пока** условие **выполнять** действия»;

- цикл «**Для всех** переменная и множество (диапазон) **выполнять** действия».

С одной стороны, эти конструкции интуитивно понятны, с другой, — к моменту появления они уже должны быть изучены в курсе информатики.

О литературе

В список литературы помещены издания, которые, во-первых, были использованы при написании данного пособия и, во-вторых, содержат дополнительный материал к указанным после них главам данной книги. Этот список не претендует на полноту. В частности, в него не включены зарубежные источники. Отметим среди них монографию Meinel C., Theobald T. Algorithms and data structures in VLSI design. OBDD foundations and applications. Berlin, Springer-Verlag, 1998, которая послужила основой для лекции 13.

О программах курсов

Как мы уже отметили, содержание книги обеспечивает преподавание семестрового курса «Основы дискретной математики» и годового курса «Дискретная математика». Семестровый курс обычно включает материал лекций 1–11 (при этом некоторые более сложные доказательства и алгоритмы опускаются). Годовой курс содержит основной материал лекций 1–5 и 9–20. Вместе с тем, большинство глав достаточно независимы, что позволяет выбрать и другой состав и порядок изучения.

Глава 1

Множества и отношения

Краткое содержание: множества и операции над ними, как доказывать равенство множеств, отношения и функции, отношения эквивалентности и частичного порядка, мощность множеств, слова и языки.

Ключевые слова: множество, отношение принадлежности, подмножество, пустое множество, объединение, пересечение, разность, декартово произведение, декартова степень, бинарное отношение, n-местное отношение, эквивалентность, частичный порядок, линейный порядок, функция, взаимно однозначная функция, перестановка, мощность, счётное множество, алфавит, слово, язык.

§ 1.1. Множества

М н о ж е с т в о — это одно из основных понятий математики, как дискретной, так и непрерывной. Оно не определяется через другие понятия. Содержательно, под множеством понимается некоторая совокупность каких-либо объектов. Основное отношение между объектами и множествами — это о т н о ш е н и е п р и н а д л е ж н о с т и объекта множеству. Оно обозначается знаком $\in$: $x \in A$ означает, что

объект x принадлежит множеству A. В этом случае x называют элементом множества A. Запись $x \notin A$ означает, что объект x не входит в множество A.

Два множества считаются равными, если они содержат одни и те же элементы, это утверждение называется аксиомой экстенсиональности.

Множество, не содержащее ни одного элемента, называется пустым и обозначается $\varnothing$.

Теорема 1. *Пустое множество единственно.*

Доказательство. В самом деле, все пустые множества содержат одни и те же элементы (никаких не содержат), по аксиоме экстенсиональности они равны. □

Множества обозначаются с помощью пары фигурных скобок, в которые заключены их элементы. Небольшие множества задаются прямым перечислением всех элементов. Например, множество простых чисел, не превосходящих 10, — это $\{2, 3, 5, 7\}$; множество (имён) летних месяцев: {июнь, июль, август}. В описаниях «больши́х» конечных множеств используют многоточие. В них часто указывается несколько первых элементов и последний элемент множества. Например, множество целых чисел, не превосходящих 100, записывают как $\{0, 1, 2, \ldots, 100\}$, множество всех месяцев года — как {январь, февраль, ..., декабрь}. Такое задание требует определённой аккуратности. Например, если некоторое множество A задано как $\{3, 5, 7, \ldots, 19\}$, то не ясно, является ли A множеством нечётных чисел, лежащих в интервале от 3 до 19, или это множество простых чисел из того же интервала (возможны и другие его «расшифровки»). Перечисления элементов бесконечных множеств начинаются несколькими начальными элементами, а завершаются многоточием. При этом часто указывают общий вид элемента задаваемого множества. Основное бесконечное множество, рассматриваемое в дискретной математике, — это множество всех натуральных чисел $\omega = \{0, 1, 2, 3, \ldots\}$. Множество всех квадратов этих чисел можно записать, например, так: $\{0, 1, 4, 9, \ldots, n^2, \ldots\}$.

Элементами множеств могут быть другие множества, например, множество $\{1, \{2, 3\}, \omega\}$ содержит единицу, множество, состоящее из 2 и 3, а также множество натуральных чисел. Не следует путать пустое множество $\varnothing$ и множество $\{\varnothing\}$, составленное из пустого: в первом множестве нет ни одного элемента, во втором — в точности один, а именно — $\varnothing$.

Множества, элементами которых являются только другие множества, часто называют с е м е й с т в а м и.

Как мы уже отметили, больши́е множества не всегда можно точно определить, используя перечисление с многоточием. Основной способ их описания имеет вид:

$$\{e \in A : \text{условие на } e\}.$$

Такая запись означает множество, составленное из тех элементов A, которые удовлетворяют условию. Например,

$$\{n \in \omega : 10 \leqslant n \leqslant 1000\}$$

это множество натуральных чисел на отрезке от 10 до 1000,

$$\{x \in \omega : x \text{ — простое число}\}$$

это множество всех простых чисел. Если заранее определено некоторое «универсальное» множество U и все рассматриваемые множества составляются только из элементов U, то часто ограничиваются записью вида

$$\{e : \text{условие на } e\},$$

подразумевая, что e берутся из U. Так, например, поступают в математическом анализе, когда рассматривают множества действительных чисел, в качестве U тогда выступает множество всех действительных чисел $\mathbb{R}$.

Совсем не указывать, из какого множества следует брать элементы, нельзя, так как попытка составить множество из всех объектов, которые удовлетворяют какому-то условию, может привести к противоречию.

Теорема 2 (Парадокс Рассела)**.** *Не существует множества, которое содержало бы все x, удовлетворяющие условию $x \notin x$.*

Доказательство. Допустим, что такое множество R существует. Попробуем ответить на вопрос: принадлежит ли R само себе, $R \in R$?

Если ответ «да», то, поскольку все $x \in R$ удовлетворяют условию $x \notin x$, из $R \in R$ получаем, что и R должен этому условию удовлетворять: $R \notin R$. Получили противоречие, следовательно, такой случай невозможен.

Если ответ «нет», то есть $R \notin R$, то поскольку R содержит все x, удовлетворяющие условию $x \notin x$, и само R этому условию удовлетворяет, то получаем $R \in R$. Снова получили противоречие, такой вариант тоже невозможен.

Итак, любой ответ на вопрос «$R \in R$?» приводит к противоречию, значит, такого множества R существовать не может. □

Подобные совокупности объектов, которые могут быть «слишком велики», чтобы содержаться в множествах, называют *классами*.

Ещё один способ определения множеств — получение его элементов из элементов другого множества по какому-либо закону S. Запись

$$\{S(e) : e \in A\},$$

означает, что элементы этого множества получены применением S ко всем элементам A. Например, $\{n^2 : n \in \omega\}$ — это множество квадратов натуральных чисел. Оба способа задания множеств можно совместить, скажем,

$$\{n^2 : n \in \omega \text{ и } n \text{ — простое число}\}$$

означает множество квадратов простых чисел.

Определение 1 (Подмножество)**.** *Если каждый элемент множества A является также элементом множества B, то множество A называется подмножеством множества B, а множество B — надмножеством A. Так же говорят, что B включает в себя A. Этот факт записывают в виде $A \subseteq B$.*

Если $A \subseteq B$ и $B \subseteq A$, то $A = B$, то есть множества A и B равны. Если $A \subseteq B$ и $A \neq B$, то A называется собственным (или строгим) подмножеством множества B, в этом случае иногда пишут $A \subset B$.

Определение 2 (Множество подмножеств)**.** *Множество (семейство) всех подмножеств множества A обозначается через $\mathsf{P}(A)$, то есть*

$$\mathsf{P}(A) = \{B : B \subseteq A\}.$$

Например, если $A = \{0, 1, \{2, 3\}\}$, то

$$\mathsf{P}(A) = \{\varnothing, \{0\}, \{1\}, \{\{2,3\}\}, \{0,1\}, \{0,\{2,3\}\}, \{1,\{2,3\}\}, \{0,1,\{2,3\}\}\},$$

а для пустого множества $\varnothing$ семейство его подмножеств состоит только из него самого: $\mathsf{P}(\varnothing) = \{\varnothing\}$.

§ 1.2. Операции над множествами

Имеется целый ряд операций, позволяющих получать одни множества из других. Рассмотрим основные из них.

Определение 3 (Объединение)**.** *Для множеств A и B их объединением называется множество*

$$A \cup B = \{x : x \in A \text{ или } x \in B\}.$$

Объединением семейства множеств A_i, $i \in I$, называется множество

$$\bigcup_{i \in I} A_i = \{x : \text{существует такое } i_0 \in I, \text{ что } x \in A_{i_0}\}.$$

Если семейство I пусто, то условие $i_0 \in I$ не может быть выполнено ни для какого i_0, поэтому $\bigcup\limits_{i \in I} A_i$ тоже пусто.

Определение 4 (Пересечение). *Пересечением двух множеств A и B называется множество*

$$A \cap B = \{x : x \in A \text{ и } x \in B\}.$$

Пересечением семейства множеств A_i, $i \in I$, $I \neq \varnothing$, называется множество

$$\bigcap_{i \in I} A_i = \{x : \text{для всех } i \in I \text{ выполнено } x \in A_i\}.$$

Условие $I \neq \varnothing$ для пересечения семейства множеств необходимо, так как при $I = \varnothing$ мы бы получили, что на x не наложено вообще никаких условий. Это приводит к такому же противоречию, как в парадоксе Рассела.

Из определения объединения и пересечения непосредственно следует, что они обладают свойствами ассоциативности:

$$A \cup (B \cup C) = (A \cup B) \cup C,$$
$$A \cap (B \cap C) = (A \cap B) \cap C$$

и коммутативности:

$$A \cup B = B \cup A, \quad A \cap B = B \cap A.$$

Следующей операцией над множествами, которую мы рассмотрим, будет разность.

Определение 5 (Разность). *Разностью множеств A и B называется множество*

$$A \setminus B = \{x : x \in A \text{ и } x \notin B\}.$$

Как мы уже говорили, часто все рассматриваемые множества являются подмножествами некоторого «универсального» множества U. В этом случае разность $U \setminus A$ называется дополнением множества A (в U) и обозначается через $\bar{A}$. Ясно, что $A \cup \bar{A} = U$ и $A \cap \bar{A} = \varnothing$.

Определение 6 (Симметрическая разность)**.** *Симметрической разностью множеств A и B называется множество*

$$A \div B = (A \setminus B) \cup (B \setminus A).$$

Иногда симметрическую разность множеств называют дизъюнктивной суммой и обозначают $A \oplus B$ или $A \nabla B$.

Определение 7 (Декартово произведение)**.** *Пусть даны n множеств $A_1, \ldots, A_n$, $n > 0$. Их декартовым произведением называется множество упорядоченных n-ок, в которых i-й по порядку элемент принадлежит A_i:*

$$A_1 \times \cdots \times A_n = \{(a_1, \ldots, a_n) : a_1 \in A_1, \ldots, a_n \in A_n\}.$$

Если $A_1 = \cdots = A_n = A$, то $A_1 \times \cdots \times A_n$ называется декартовой степенью множества A и обозначается через A^n.

Пример 1. *Пусть заданы множества $A = \{0, 1, \ldots, n\}$ и $B = \{0, 1, \ldots m\}$, где $n \in \omega$ и $m \in \omega$ — натуральные числа и $n < m$. Тогда*

$$\begin{aligned} &A \cup B = B;\\ &A \cap B = A;\\ &A \setminus B = \varnothing;\\ &B \setminus A = \{n+1, \ldots, m\};\\ &A \times B = \{(i, j) : 0 \leqslant i \leqslant n, 0 \leqslant j \leqslant m\}. \end{aligned}$$

§ 1.3. Как доказывать равенство множеств

Многие математические утверждения, в том числе и многие теоремы в этой книге, имеют следующую форму:

> Даны разные определения двух множеств A и B. Требуется доказать, что $A = B$.

Согласно аксиоме экстенсиональности нам достаточно будет показать, что A и B содержат одни и те же элементы. Точнее, мы должны

установить, что все элементы A принадлежат B, а все элементы B принадлежат A. Но первое означает $A \subseteq B$, а второе — $B \subseteq A$.

Поэтому стандартный способ доказательства утверждения вида $A = B$ для множеств A и B состоит в доказательстве двух утверждений о включениях:

1) $A \subseteq B$ и

2) $B \subseteq A$.

Доказательства этих включений проводятся по такой схеме: берётся произвольный элемент, удовлетворяющий определению меньшего множества (слева от знака $\subseteq$), и устанавливается, что он удовлетворяет также определению большего множества (справа от $\subseteq$).

В качестве примера докажем одно из свойств (законов) дистрибутивности для операций объединения и пересечения:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$

1) Пусть a — произвольный элемент из $A \cup (B \cap C)$. Тогда по определению операции $\cup$ имеем $a \in A$ или $a \in (B \cap C)$. В первом случае из того же определения выводим, что $a \in (A \cup B)$ и $a \in (A \cup C)$. Но тогда по определению операции $\cap$ получаем, что $a \in (A \cup B) \cap (A \cup C)$. Во втором случае из определения $\cap$ следует, что $a \in B$ и $a \in C$. Из этого и из определения $\cup$ снова следует, что $a \in (A \cup B)$ и $a \in (A \cup C)$ и $a \in (A \cup B) \cap (A \cup C)$. Таким образом, мы установили, что $A \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C)$.
2) Пусть теперь $a \in (A \cup B) \cap (A \cup C)$. Тогда по определению операции $\cap$ имеем $a \in (A \cup B)$ и $a \in (A \cup C)$. Если $a \in A$, то оба эти включения выполнены. Но тогда $a \in A \cup (B \cap C)$. Если же $a \notin A$, то из первого включения следует, что $a \in B$, а из второго — $a \in C$. Следовательно, $a \in (B \cap C)$ и $a \in A \cup (B \cap C)$. Таким образом, $(A \cup B) \cap (A \cup C) \subseteq A \cup (B \cap C)$ и наше утверждение доказано.

Используя эту же схему, можно установить много других свойств введённых выше операций над множествами и связей между ними (см. задачи 2 на стр. 39 и 6 на стр. 39).

§ 1.4. Отношения и функции

Важными частными случаями множеств являются отношения и функции.

Определение 8 (Бинарное отношение)**.** *Пусть A и B — множества. Бинарным (или двухместным) отношением между элементами A и B называется любое подмножество R их декартова произведения $A \times B$.*

Говорят также, что R является отношением из A в B. При $A = B$ отношение R называется бинарным отношением на множестве A. Вместо $(x, y) \in R$ часто пишут $x\ R\ y$. Например, для отношений порядка на множестве натуральных чисел ω используют записи вида $3 \leqslant 7$, $x \geqslant 23$, $z > y$ и т. д.

Определение 9 (Тождественное отношение)**.** *Тождественным отношением на множестве A называется отношение*

$$\mathrm{id}_A = \{(x, x) : x \in A\}.$$

Его обозначают знаком равенства «=».

С бинарным отношением R связаны его область определения $\operatorname{dom} R$ и область значений $\operatorname{rng} R$:

$$\operatorname{dom} R = \{x : (x, y) \in R \text{ для некоторого } y\}$$
$$\operatorname{rng} R = \{y : (x, y) \in R \text{ для некоторого } x\}.$$

Обратным отношением для бинарного отношения R называется множество пар

$$R^{-1} = \{(x, y) : (y, x) \in R\}.$$

Образом множества X относительно бинарного отношения R называется множество

$$R(X) = \{y : (x, y) \in R \text{ для некоторого } x \in X\},$$

прообразом X относительно R называется $R^{-1}(X)$.

Композицией отношений $R_1 \subseteq A \times B$ и $R_2 \subseteq B \times C$ называется следующее отношение $R_1 \circ R_2 \subseteq A \times C$:

$$R_1 \circ R_2 = \{(x, z) : \text{существует } y \in B \text{ такой, что } (x, y) \in R_1 \text{ и } (y, z) \in R_2\}.$$

Важную роль среди бинарных отношений играют отношения эквивалентности.

Определение 10 (Отношение эквивалентности)**.** *Бинарное отношение R на множестве A называется отношением эквивалентности, если для него выполнены следующие условия:*

1) *рефлексивность: для всех $a \in A$ выполнено $(a, a) \in R$;*
2) *симметричность: для всех $a, b \in A$ из $(a, b) \in R$ следует $(b, a) \in R$;*
3) *транзитивность: для всех $a, b, c \in A$, если $(a, b) \in R$ и $(b, c) \in R$, то и $(a, c) \in R$.*

Примером отношения эквивалентности на множестве натуральных чисел ω является равенство остатков при делении на некоторое фиксированное положительное число n: $a \equiv b \pmod{n}$.

С каждым отношением эквивалентности $\equiv$ на множестве A связано разбиение A на непересекающиеся подмножества — классы эквивалентности.

Определение 11 (Класс эквивалентности)**.** *Для каждого $a \in A$ его класс эквивалентности $[a]_{\equiv}$ — это множество всех эквивалентных a элементов:*

$$[a]_{\equiv} = \{b \in A : a \equiv b\}.$$

Из определения эквивалентности непосредственно следует, что, если $a \equiv b$, то $[a]_{\equiv} = [b]_{\equiv}$, а если $a \not\equiv b$, то $[a]_{\equiv} \cap [b]_{\equiv} = \varnothing$. Таким образом, разбиение A на классы эквивалентности не зависит от выбора конкретных представителей этих классов в качестве их имён.

Если в приведённом выше примере в качестве n взять, например, 5, то все числа из ω разобьются на 5 классов эквивалентности: N_0, N_1, N_2, N_3, N_4, где в класс N_i, $i = 0, 1, 2, 3, 4$, войдут числа, дающие при делении на 5 остаток i.

Ещё один важный класс отношений — отношения порядка.

Определение 12 (Отношение частичного порядка)**.** *Бинарное отношение R на множестве A называется отношением частичного порядка, если для него выполняются следующие два условия:*

1) *антисимметричность: для всех $a, b \in A$, если $(a, b) \in R$ и $(b, a) \in R$, то $a = b$;*
2) *транзитивность.*

Если при этом для отношения R выполнено ещё свойство рефлексивности, то такое отношение порядка называют нестрогим, а если свойство антирефлексивности: $(a, a) \notin R$ для всех $a \in A$, то — строгим.

Примером строго отношения частичного порядка является отношение строгого включения на множестве $\mathsf{P}(A)$ всех подмножеств некоторого множества A. Нестрогий порядок — это, например, отношение делимости $|$ на множестве натуральных чисел.

Обычное отношение строгого порядка $<$ на ω также удовлетворяет условиям строгого частичного порядка. Но для него выполнено ещё одно существенное условие — линейность.

Определение 13 (Линейный порядок)**.** *Отношение частичного порядка называется линейным, если для всех $a, b \in A$ выполнено хотя бы одно из трёх: $(a, b) \in R$, $(b, a) \in R$ или $a = b$.*

Упомянутое выше отношение делимости $|$ не является линейным, так как 2 и 3 не делятся друг на друга. Отношение строгого включения $\subset$ на $\mathsf{P}(A)$ будет линейным только для множеств A, содержащих не более одного элемента.

Определение 14 (Функция)**.** *Говорят, что бинарное отношение f является функцией из A в B, если* $\operatorname{dom} f = A$, $\operatorname{rng} f \subseteq B$ *и для всех x, y_1, y_2 из того, что $(x, y_1) \in f$ и $(x, y_2) \in f$, следует, что $y_1 = y_2$. Записывается этот факт в виде $f : A \to B$.*

В качестве синонимов термина «функция» часто используются слова отображение или преобразование. Если f функция, то вместо $(x, y) \in f$ пишем $f(x) = y$ (иногда — просто $fx = y$) и называем y значением f на аргументе x.

Определение 15 (Разнозначная, сюръективная, взаимно однозначная функция)**.** *Функция $f : A \to B$ называется разнозначной (или 1-1-функцией, обратимой, инъективной), если для всех x_1, x_2, y из того, что $f(x_1) = y$ и $f(x_2) = y$, следует, что $x_1 = x_2$.*

Функция $f : A \to B$ называется сюръективной (или покрытием), если $\operatorname{rng} f = B$.

Функция $f : A \to B$ называется взаимно однозначной, если она является одновременно разнозначной и сюръективной. В этом случае часто пишут $f : A \leftrightarrow B$.

Взаимно однозначная функция $f : A \to A$ называется перестановкой множества A.

Тождественное отношение id_A является примером взаимно однозначной функции на множестве A.

Важным видом функций являются характеристические.

Определение 16 (Характеристическая функция)**.** *Пусть $A \subseteq B$. Характеристическая функция подмножества A (в множестве B) — это функция $I_A : B \to \{0, 1\}$ такая, что $f(x) = 1$ при $x \in A$ и $f(x) = 0$ при $x \notin A$.*

Например, характеристическая функция пустого множества $\varnothing$ всегда равна нулю: $I_\varnothing(x) = 0$, так как для всех x выполнено $x \notin \varnothing$.

Определения бинарных отношений и функций с одним аргументом естественным образом обобщаются на многоместные отношения и функции.

Определение 17 (n-местное отношение, n-местная функция)**.** *n-местным (или n-арным) отношением на множествах $A_1, \ldots, A_n$ называется любое подмножество $A_1 \times \cdots \times A_n$. Функцию $f : A_1 \times \ldots \times A_n \to B$ называют n-местной (или n-арной) и записывают $f(x_1, \ldots, x_n) = y$, когда $((x_1, \ldots, x_n), y) \in f$.*

Чаще всего мы будем рассматривать n-местные функции для $A_1 = \ldots = A_n = A$. В этом случае $f : A^n \to B$ будем называть n-местной функцией из A в B.

§ 1.5. Мощность множеств

В ряде случаев бывает необходимо сравнивать множества по количеству элементов. Если элементы можно пересчитать, то задача несложна, но для бесконечных множеств такой способ не годится.

Чтобы выяснить, в каком множестве A или B больше элементов, можно воспользоваться следующим методом, который подходит для любых множеств. Будем выбирать из множеств A и B по одному элементу, образуя пары (a_1, b_1), (a_2, b_2) и т. д., до тех пор, пока мы полностью не исчерпаем одно из этих множеств, например, A. В этом случае A, очевидно, не может содержать больше элементов, чем B, а множество построенных пар образует разнозначную функцию из A в B. Если же множества A и B исчерпаны одновременно, то количества их элементов следует считать равными, а построенная функция будет взаимно однозначной.

Определение 18 (Мощность множеств)**.** *Множества A и B называют равномощными, если между ними существует взаимно однозначная функция. Равномощность множеств A и B записывается в виде $|A| = |B|$.*

Если из A в B существует разнозначная функция, то говорят, что мощность A меньше или равна мощности B и записывают это в виде $|A| \leqslant |B|$.

Если $|A| \leqslant |B|$ и $|A| \neq |B|$, то A менее мощно чем B, что записывается в виде $|A| < |B|$.

Неформально можно считать, что мощность множества — это количество элементов в нём. Например, для натуральных n мощность множества $N_n = \{0, 1, \ldots, n-1\}$ равна n. В частности, для пустого множества $|\varnothing| = 0$.

Определение 19 (Конечные и счётные множества)**.** *Множество называется конечным, если оно для некоторого $n \in \omega$ равномощно множеству N_n.*

Множество, мощность которого меньше или равна мощности множества натуральных чисел ω, называется счётным.

Если оно при этом не является конечным, то оно называется счётно бесконечным.

Необходимый и достаточный критерий счётности таков.

Теорема 3. *Непустое множество A счётно тогда и только тогда, когда все его элементы можно пронумеровать натуральными числами (возможно, с повторениями: один элемент может получить несколько разных номеров).*

Доказательство. Если множество A счётно, то существует разнозначная функция $f : A \to \omega$, для которой есть обратная f^{-1}. Поскольку A не пусто, то найдётся $a \in A$. Тогда все элементы A можно пронумеровать так:

$$a_i = \begin{cases} f^{-1}(i), & \text{если } i \in \operatorname{rng} f, \\ a & \text{в противном случае.} \end{cases}$$

Пусть теперь все элементы A пронумерованы: $A = \{a_i : i \in \omega\}$. Тогда определим функцию $f : A \to \omega$ следующим образом:

$$f(a) = \min \underbrace{\{i \in \omega : a_i = a\}}_{X_a}.$$

Поскольку каждый элемент $a \in A$ имеет хотя бы один номер, то множество $X_a \subseteq \omega$ непусто, а в непустом множестве натуральных чисел всегда есть наименьший элемент. Функция f будет разнозначной, так как для разных a множества X_a попарно не пересекаются, поэтому и наименьшие их элементы различны. □

Пример 2. *Пусть* $\mathsf{P}_f(A)$ *означает множество всех конечных подмножеств* A*. Докажем, что* $\mathsf{P}_f(\omega)$ *счётно бесконечно.*

Для этого рассмотрим такую функцию $\rho : \mathsf{P}_f(\omega) \to \omega$*:*

$$\rho(X) = \sum_{i \in X} 2^i.$$

Нетрудно видеть, что значением $\rho(X)$ *будет натуральное число, в двоичной записи которого единица стоит на* i*-м разряде (начиная с младших) тогда и только тогда, когда* $i \in X$*. В силу существования и единственности двоичной записи для каждого натурального числа эта функция* ρ *будет взаимно однозначной.*

Существуют и несчётные множества.

Теорема 4 (Теорема Кантора)**.** *Для любого множества* A *имеет место* $|A| < |\mathsf{P}(A)|$.

Доказательство. Неравенство $|A| \leqslant |\mathsf{P}(A)|$ легко получается с помощью построения разнозначной функции $f : A \to \mathsf{P}(A)$, достаточно положить $f(a) = \{a\}$ для всех $a \in A$.

Чтобы доказать неравенство $|A| \neq |\mathsf{P}(A)|$, допустим противное: $|A| = |\mathsf{P}(A)|$. Это означает наличие взаимно однозначной функции $f : A \to \mathsf{P}(A)$. Дальнейшие рассуждения будут напоминать доказательство парадокса Рассела. Рассмотрим множество R:

$$R = \{a \in A : a \notin f(a)\}. \tag{1}$$

Тогда $R \subseteq A$. Поскольку функция f является взаимно однозначной, то $f(x) = R$ для некоторого $x \in A$. Возможно два случая: $x \in R$ или $x \notin R$. Покажем, что оба они ведут к противоречию, что означает неверность исходного предположения о равномощности множеств A и $\mathsf{P}(A)$.

Если $x \in R$, то, как и для любого элемента R, для x должно выполняться $x \notin f(x)$, но так как $f(x) = R$ мы получаем $x \notin R$, противоречие.

Если $x \notin R$, то из $f(x) = R$ получаем $x \notin f(x)$, поэтому x должен войти в R согласно (1), то есть $x \in R$, противоречие. □

В частности, при $A = \omega$ получаем

Следствие 5. *$|P(\omega)| > |\omega|$, то есть $\mathsf{P}(\omega)$ несчётно.*

Другим примером несчётного множества является множество действительных чисел $\mathbb{R}$.

Теорема 6. $|\mathbb{R}| = |\mathsf{P}(\omega)|$.

Доказательство. Сначала рассмотрим функцию

$$f(x) = \frac{\operatorname{arctg} x + {}^{\pi}/_{2}}{\pi}.$$

Она взаимно однозначно отображает $\mathbb{R}$ на интервал $(0;1)$. Следовательно, $|\mathbb{R}| = |(0;1)|$.

Далее покажем, что $|(0;1)| = |[0;1]|$. Для этого рассмотрим функцию g определённую так: $g({}^1/_2) = 1$, $g({}^1/_3) = 0$, $g({}^1/_n) = {}^1/_{(n-2)}$ для $n > 3$, $g(x) = x$ для всех остальных чисел интервала $(0,1)$. Тогда g взаимно однозначно отображает интервал $(0;1)$ на отрезок $[0;1]$, то есть их мощности равны.

Теперь продемонстрируем $|[0;1]| \leqslant |\mathsf{P}(\omega)|$. Для этого каждое число x из отрезка $[0;1]$ представим в виде двоичной дроби: $x = 0, x_0x_1x_2\ldots$, где $x_i \in \{0,1\}$ — $(i+1)$-я двоичная цифра после запятой. Заметим, что единица тоже так представляется: $1 = 0,1111\ldots$ Некоторые числа можно записать в виде двоичной дроби разными способами, например,

$${}^1/_2 = 0,1000\ldots = 0,0111\ldots$$

В таких случаях будем считать, что зафиксирован один из этих способов. Определим функцию $h(x) = A_x$, где $A_x = \{i \in \omega : x_i = 1\}$. Тогда h разнозначно отображает $[0;1]$ в $\mathsf{P}(\omega)$, то есть $|[0;1]| \leqslant |\mathsf{P}(\omega)|$.

Наконец, покажем $|\mathsf{P}(\omega)| \leqslant |[0;1]|$. Для этого каждому подмножеству $A \subseteq \omega$ сопоставим число x_A, двоичная запись которого имеет следующий вид: $0, x_0\bar{x}_0x_1\bar{x}_1x_2\bar{x}_2\ldots$ Здесь $x_i = 1$, если $i \in A$, $x_i = 0$, если $i \notin A$, $\bar{x}_i = 1 - x_i$. Тогда функция $f(A) = x_A$ является разнозначной, поэтому $|\mathsf{P}(\omega)| \leqslant |[0;1]|$.

Подводя итог, мы видим: $|\mathbb{R}| = |(0;1)| = |[0;1]|$, $|[0;1]| \leqslant |\mathsf{P}(\omega)|$ и $|\mathsf{P}(\omega)| \leqslant |[0;1]|$. Следовательно,

$$|\mathbb{R}| = |(0;1)| = |[0;1]| = |\mathsf{P}(\omega)|. \qquad \square$$

Если $A \subseteq B$, то, очевидно, $|A| \leqslant |B|$, потому что тождественная функция id_A одновременно будет разнозначным отображением A в B. Верно ли аналогичное утверждение для строгого включения: если $A \subset B$, то $|A| < |B|$?

Если множество A конечно, то ответ утвердительный: при добавлении элементов к конечному множеству его мощность, конечно же, возрастает.

Для бесконечных множеств это уже не так и собственное подмножество может иметь ту же мощность, что и всё множество.

Пример 3. *Множество натуральных чисел ω является собственным подмножеством целых $\mathbb{Z}$: $\omega \subset \mathbb{Z}$. Но тем не менее $|\omega| = |\mathbb{Z}|$, как показывает следующая взаимно однозначная функция $f : \omega \to \mathbb{Z}$,*

$$f(x) = (-1)^x \cdot \left[\frac{x+1}{2}\right].$$

Рассмотрим ещё один пример, покажем, что единичный квадрат

$$Q = \{(x, y) : x, y \in [0; 1)\}$$

имеет ту же мощность, что и его сторона $L = [0; 1)$. Поскольку $L \subseteq Q$, то $|L| \leqslant |Q|$. Чтобы построить разнозначное отображение $f : Q \to L$, поступим следующим образом. Запишем любую пару чисел $x, y \in [0; 1)$ в десятичной системе:

$$x = 0, x_0x_1x_2\ldots; \quad y = 0, y_0y_1y_2\ldots$$

и положим

$$f(x, y) = 0, 0x_0y_00x_1y_10x_2y_2\ldots,$$

где x_i и y_i — цифры десятичной записи чисел x и y соответственно. Указанная функция f является разнозначной, поэтому $|Q| \leqslant |L|$.

Окончательно делаем вывод, что $|Q| = |L|$.

Замечание 1 (Как сравнивать мощность множеств?)**.** Читатель, вероятно, обратил внимание вот на что. Выше в нескольких случаях мы вместо того, чтобы доказывать равномощность множеств: $|A| = |B|$, доказывали два неравенства: $|A| \leqslant |B|$ и $|B| \leqslant |A|$. Строго говоря, нужно обосновать, что из этих двух неравенств следует равенство мощностей, ведь неравенство означает по определению существование разнозначной функции, а равенство — взаимно однозначной. Почему из наличия двух разнозначных

функций $f : A \to B$ и $g : B \to A$ следует существование взаимно однозначной $h : A \leftrightarrow B$?

На самом деле, это утверждение справедливо и носит название теоремы Кантора-Бернштейна. Доказательство её не очень сложно, но выходит за рамки нашего курса. Интересующиеся могут обратиться к [2].

Используя мощностные соображения, можно доказывать существование объектов, не строя их. Рассмотрим такой пример. Число называется алгебраическим, если оно является корнем ненулевого многочлена с целыми коэффициентами. Например, $1/2$ — алгебраическое число, так как является корнем уравнения $2x - 1 = 0$. Аналогично, числа $\pm\sqrt{2}$ алгебраические, так как являются корнями $x^2 - 2 = 0$. Вопрос заключается в том, существуют ли неалгебраические числа (они называются трансцендентными)? Мы дадим лёгкое доказательство того, что они действительно есть.

Теорема 7. *Трансцендентные числа существуют.*

Доказательство. Из теоремы 6 на стр. 34 мы знаем, что существует несчётно много действительных чисел. Но алгебраических чисел существует счётно много (задача 21 на стр. 41). Следовательно, алгебраических чисел меньше, чем действительных, и должны существовать действительные числа, которые алгебраическими не являются. □

Заметим, что хотя существование трансцендентных чисел доказывается легко, но доказать для какого-то конкретного числа x, что оно трансцендентно, задача весьма сложная, а для многих известных констант — до сих пор не решённая.

Другой интересный вопрос заключается в следующем. Как мы видели $\omega \subset \mathbb{R}$ и при этом $|\omega| < |\mathbb{R}|$. Поэтому если брать «промежуточные» множества A: $\omega \subseteq A \subseteq \mathbb{R}$, то будет выполнено $|\omega| \leqslant |A| \leqslant |\mathbb{R}|$. Вопрос заключается в том, а существуют ли множества A, для которых оба неравенства строгие: $|\omega| < |A| < |\mathbb{R}|$, то есть A содержит больше элементов, чем существует всех натуральных чисел, но меньше, чем всех действительных? Утверждение о том, что таких множеств не существует, называется «континуум-гипотезой».

Оказывается, что выяснить справедливость континуум-гипотезы невозможно: положительный ответ на вопрос о существовании «промежуточного» множества A, $|\omega| < |A| < |\mathbb{R}|$, столь же непротиворечив, как и отрицательный. Таким образом, континуум-гипотеза является примером неразрешимого в классической математике утверждения, то есть такого, которое нельзя ни доказать, ни опровергнуть.

Замечание 2 (Так что же такое «мощность»?)**.** Несмотря на то, что параграф назван «Мощность множеств», мы точного определения понятия «мощность» до сих пор не дали. Вместо этого мы определили два отношения для множеств, которые обозначили $|A| = |B|$ и $|A| \leqslant |B|$. Фактически эти отношения являются более фундаментальными и важными, чем точное определения понятия мощности. Кроме того, точное определение понятия «мощность» требует гораздо более глубокого знакомства с теорией множеств, чем это сделано в настоящей главе.

Мы попытаемся неформально описать определение этого понятия. Среди всех равномощных множеств выделяется одно, «эталонное», оно и считается мощностью всех этих множеств. Эти «эталонные» множества называют кардиналами, тогда мощностью произвольного множества A является равномощный ему кардинал и обозначается $|A|$. Например, кардиналами являются определённые выше множества N_n, поэтому мощностью конечного множества A, содержащего n элементов, является N_n: $|A| = N_n$. Обычно N_n отождествляют с самим натуральным числом n, тогда получаем $|A| = n$. Следующим кардиналом является всё множество натуральных чисел ω, поэтому для счётно бесконечных множеств A выполнено $|A| = \omega$.

В нашем курсе мы будем в основном рассматривать только счётные множества, а также — отношения и функции на таких множествах. Отметим, что многие объекты, изучаемые в дискретной математике, являются частными случаями отношений и функций на конечных множествах. К ним относятся, в частности, слова.

§ 1.6. Слова и языки

Введём ещё несколько понятий, которые нам в дальнейшем будут постоянно встречаться.

Определение 20 (Слова в алфавите). *Пусть дан алфавит — произвольное конечное множество $\Sigma = \{a_1, \ldots, a_m\}$, элементы которого называются символами (или буквами). Слово в алфавите Σ — это конечная последовательность символов этого алфавита: $w = a_{i_1} \ldots a_{i_n}$, $a_{i_j} \in \Sigma$ при $j = 1, \ldots, n$, то есть упорядоченная n-ка, элемент декартовой степени Σ^n. Количество букв в этой последовательности называется длиной слова и обозначается $|w|$. Имеется пустое слово длины ноль, которое не содержит ни одного символа. Будем обозначать его через ε.*

Нетрудно понять, что слова длины n взаимно однозначно соответствуют функциям вида $f : \{1, \ldots, n\} \to \Sigma$. А именно, слову $w = a_{i_1} \ldots a_{i_n}$, соответствует функция $f_w(j) = a_{i_j}$, $j = 1, \ldots, n$.

Определение 21 (Язык). *Языком в алфавите Σ называется произвольное множество слов этого алфавита.*

На языках, как и на любых множествах, определены операции объединения, пересечения и разности. Язык, содержащий все слова в алфавите Σ (в том числе и пустое), обозначается через Σ^*. Дополнением языка $L \subseteq \Sigma^*$ называется разность $\bar{L} = \Sigma^* \setminus L$.

На словах определена операция приписывания одного слова после другого, называемая конкатенацией.

Определение 22 (Конкатенация слов). *Если даны два слова $u = a_{i_1} \ldots a_{i_n}$ и $v = a_{k_1} \ldots a_{k_\ell}$, то их конкатенацей будет слово $w = u \,\&\, v = a_{i_1} \ldots a_{i_n} a_{k_1} \ldots a_{k_\ell}$ длины $n + \ell$.*

Обычно знак конкатенации & будем опускать и писать просто uv (по аналогии со знаком умножения в алгебре).

Пустое слово — это единственное слово такое, что для любого слова w справедливы равенства $w\varepsilon = \varepsilon w = w$.

Операция конкатенации ассоциативна: для любых трёх слов w, v и u, очевидно, имеет место равенство: $(wv)u = w(vu)$. Поэтому скобки при записи конкатенации нескольких слов будем опускать. Для представления конкатенаций слова с собой же используют сокращённую,

«степенную», форму записи:

$$w^0 = \varepsilon; \quad w^1 = w; \quad \ldots; \quad w^{i+1} = w^i w.$$

Например, $a^3b^4c^2$ — это сокращённая запись слова *aaabbbbcc*.

Определение 23 (Префикс, суффикс)**.** *Если $w = uv$, то слово u называется* п р е ф и к с о м*, а v — с у ф ф и к с о м слова w. Если при этом $u \neq w$ (соответственно $v \neq w$), то префикс (суффикс) называют* с о б с т в е н н ы м*.*

Задачи

1. Доказать следующие включения:
(а) $A \cap B \subseteq A \subseteq A \cup B$; (б) $A \setminus B \subseteq A$. ▼

2. Доказать следующие тождества:
(а) $A \cup A = A \cap A = A$;
(б) $A \cap (B \cup C) = (A \cap B) \cup (A \cap B)$;
(в) $(A \cup B) \cap A = A = (A \cap B) \cup A$;
(г) $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$;
(д) $A \setminus (B \setminus C) = (A \setminus B) \cup (A \cap C)$;
(е) $A \cup \varnothing = \varnothing \cup A = A$;
(ж) $A \cap \varnothing = \varnothing \cap A = \varnothing$;
(з) $A \cup \varnothing = \varnothing \cup A = A$;
(и) $A \dot{-} \varnothing = \varnothing \dot{-} A = A$ и $A \dot{-} A = \varnothing$;
(к) $A \dot{-} B = (A \cup B) \setminus (A \cap B)$. ▼

3. Найти все подмножества множеств $\varnothing$, $\{\varnothing\}$, $\{1, 2, 3\}$, $\{a, \{1, 2\}, \varnothing\}$. ▼

4. Доказать, что $A \subseteq B$ тогда и только тогда, когда $\mathsf{P}(A) \subseteq \mathsf{P}(B)$. ▼

5. Пусть $A = \{0, 1\}$, $B = \{a, b, c\}$. Определить множества $A \times B$ и $B \times A$. ▼

6. Доказать, что
(а) $A \times (B \cup C) = (A \times B) \cup (A \times C)$;
(б) $A \times (B \cap C) = (A \times B) \cap (A \times C)$;
(в) $A \times (B \setminus C) = (A \times B) \setminus (A \times C)$;
(г) если $A \subseteq B$ и $C \subseteq D$, то $(A \times C) = (A \times D) \cap (B \times C)$. ▼

7. Для каждого из следующих отношений определить $\operatorname{dom} R$, $\operatorname{rng} R$, R^{-1}, $R \circ R$, $R \circ R^{-1}$:
(а) $R = \{(x, y) : x, y \in \omega \text{ и } x \mid y\}$;
(б) $R = \{(x, y) : x, y \in \omega \text{ и } x + y \leqslant 10\}$;
(в) $R = \{(x, y) : x, y \in \omega \text{ и } y = 3x + 1\}$;
(г) $R = \{(x, x^2) : x \in \omega \text{ и } x \leqslant 10\}$;
(д) $R = \{(a, b), (b, c), (b, d), (c, d), (d, b)\}$. ▼

8. Пусть множество $S = \{(i, j) : 1 \leqslant i, j \leqslant 8\}$ задаёт клетки шахматной доски. Описать следующие бинарные отношения на S:
(а) $L = \{(a, b)$: ладья за один ход может перейти с клетки a на клетку $b\}$;

(б) $K = \{(a, b) :$ конь за один ход может перейти с клетки a на клетку $b\}$.

Будут ли эти отношения эквивалентностями? Описать отношение $L \circ L$. ▼

9. Пусть П — множество прямых на плоскости. Будут ли следующие отношения отношениями эквивалентности на П:

(а) параллельность прямых;

(б) перпендикулярность прямых. ▼

10. Пусть П — множество многоугольников на плоскости. Будут ли следующие отношения отношениями эквивалентности на П:

(а) x и y возможно совместить;

(б) x и y подобны;

(в) x и y имеют одинаковый угол;

(г) x и y пересекаются;

(д) x и y имеют одинаковую площадь;

(е) x и y имеют общую вершину;

(ж) x и y равносоставлены. ▼

11. Пусть $\Sigma = \{a_1, \ldots, a_m\}$ — произвольный конечный алфавит. Обозначим через Σ^n множество слов длины n в алфавите Σ (это обозначение согласовано с тем же обозначением декартовой степени Σ, так как степень Σ^n состоит из всех последовательностей элементов Σ длины n).

(а) Определим следующее отношение R_1 на словах из Σ^n. Пусть даны два слова: $v = a_{i_1} a_{i_2} \ldots a_{i_n}$ и $w = a_{j_1} a_{j_2} \ldots a_{j_n}$. Тогда $(v, w) \in R_1$, если для всех k от 1 до n выполнено неравенство $i_k \leqslant j_k$ и для некоторого такого k выполнено строгое неравенство $i_k < j_k$, то есть номер каждой буквы слова v не больше номера той же буквы в слове w и хотя бы у одной из букв он меньше. Является ли это отношение R_1 отношением частичного (линейного) порядка?

(б) В условиях пункта (а) полагаем $(v, w) \in R_2$, если существует такое k в интервале от 1 до n, что при $\ell < k$ выполнено $i_\ell = j_\ell$ и $i_k < j_k$ или $n < r$ и первые n символов w совпадают со словом v. Является ли это отношение R_2 отношением частичного (линейного) порядка? ▼

Замечание 3 (Упорядочение наборов)**.** Определённое в пункте (а) отношение R_1 называется отношением п о к о о р д и н а т н о г о порядка, а отношение R_2 из пункта (б) — отношением л е к с и к о г р а ф и ч е с к о г о порядка. В соответствии с лексикографическим порядком упорядочены, например, слова в словарях и энциклопедиях.

12. Доказать, что в условиях предыдущей задачи на множестве Σ^k лексикографический порядок расширяет покоординатный, то есть из $R_1 \subseteq R_2$. ▼

13. Доказать, что если $X \subseteq Y$ — конечные подмножества ω, то $\rho(X) \leqslant \rho(Y)$, где функция ρ — из примера 2 на стр. 33. Продемонстрировать, что обратное неверно. ▼

14. Доказать, что если множества A и B конечны, то

(а) $|A \times B| = |A| \cdot |B|$; (б) $|A \cup B| = |A| + |B| - |A \cap B|$. ▼

15. Доказать счётность множества пар ω^2. ▼

16. Доказать счётность множества упорядоченных n-ок ω^n. ▼

17. Доказать, что объединение счётного числа счётных множеств снова будет счётным. ▼

18. Доказать счётность множества рациональных чисел $\mathbb{Q}$. ▼

19. Доказать счётность множества многочленов n-й степени с целыми коэффициентами для фиксированного n. ▼

20. Доказать счётность множества всех многочленов с целыми коэффициентами. ▼

21. Доказать счётность множества алгебраических чисел. ▼

22. С помощью теоремы Кантора-Бернштейна доказать, что множество S последовательностей действительных чисел $(a_i)_{i\in\omega}$, $a_i \in \mathbb{R}$, равномощно $\mathbb{R}$. ▼

23. Доказать, что всякий язык в конечном алфавите счётен. ▼

24. Найти $L_1 \cap L_2$, $L_2 \setminus L_1$, $\bar{L}_1$ для следующих языков в алфавите $\Sigma = \{a, b\}$: ▼

$$L_1 = \{w \in \Sigma^* : \text{количества букв } a \text{ и } b \text{ в } w \text{ совпадают}\};$$
$$L_2 = \{a^i b^j : i, j \in \omega\}.$$

25. Найти все префиксы и суффиксы слова a^2b^2c. ▼

Глава 2

Индукция и комбинаторика

Краткое содержание: метод математической индукции, индукция по структуре объекта, комбинаторика: число размещений, перестановок и сочетаний, принцип включения и исключения.

Ключевые слова: математическая индукция, базис индукции, шаг индукции, размещение с повторением и без повторений, перестановка, сочетание, принцип включения и исключения, треугольник Паскаля, биномиальный коэффициент.

§ 2.1. Метод математической индукции

Математическая индукция — это весьма общий метод, который позволяет доказывать утверждения, зависящие от целочисленных параметров. Его можно описать следующим образом.

Пусть $P(n)$ — это некоторое утверждение, зависящее от целочисленного параметра n. Пусть, во-первых, утверждение $P(n_0)$ справедливо для некоторого целого числа n_0 и, во-вторых, для каждого $k > n_0$ из справедливости $P(\ell)$ для всех $\ell \in [n_0; k)$ следует справедливость $P(k)$. Тогда утверждение $P(n)$ справедливо для всех $n \geqslant n_0$.

В самом деле, если $P(k)$ не выполнено для какого-то $k \geqslant n_0$, то выберем k_0 — наименьшее из таких k. Поскольку $P(n_0)$ выполнено, то $k_0 > n_0$. Так как k_0 — наименьшее, то для всех $\ell \in [n_0; k_0)$ выполнено $P(\ell)$. Но тогда должно быть выполнено $P(k_0)$, что противоречит предположению.

Таким образом доказательство «по индукции» состоит из двух этапов.

1) Б а з и с и н д у к ц и и состоит в доказательстве утверждения $P(n_0)$ для некоторого начального значения n_0 (обычно $n_0 = 0$ или $n_0 = 1$, но это не обязательно).

2) Ш а г и н д у к ц и и состоит в предположении справедливости $P(\ell)$ для всех $\ell = n_0, \ldots, k-1$ (это называется и н д у к ц и о н н ы м п р е д п о л о ж е н и е м) и доказательстве из этого предположения справедливости утверждения $P(k)$.

   В частном случае $P(k)$ может следовать просто из $P(k-1)$. Тогда будет достаточно доказать, что для всякого $k \geqslant n_0$ из $P(k)$ следует $P(k+1)$. В данном случае индукционное предположение заключается в $P(k)$.

Рассмотрим несколько примеров применения метода математической индукции.

Пример 4. *Доказать, что для натуральных n выполнено*

$$1^3 + 2^3 + \ldots + n^3 = \frac{(n(n+1))^2}{4}.$$

1) *Б а з и с и н д у к ц и и. При $n = 1$ имеем $1^3 = \frac{(1(1+1))^2}{4}$.*

2) *Ш а г и н д у к ц и и. Индукционное предположение: допустим, что при $n = k$ выполнено*

$$1^3 + 2^3 + \ldots + k^3 = \frac{(k(k+1))^2}{4}.$$

Докажем тогда, что при $n = k+1$ будет выполняться

$$1^3 + 2^3 + \ldots + k^3 + (k+1)^3 = \frac{((k+1)(k+2))^2}{4}.$$

Действительно,

$$1^3+2^3+\ldots+k^3+(k+1)^3=\frac{(k(k+1))^2}{4}+(k+1)^3=$$
$$=(k+1)^2\cdot\left(\frac{k^2}{4}+k+1\right)=(k+1)^2\cdot\frac{(k+2)^2}{4}=$$
$$=\frac{((k+1)(k+2))^2}{4}.$$

Таким образом, наше утверждение выполнено при всех $n\geqslant 1$.

Пример 5. *Доказать, что для всякого действительного* $x>-1$, $x\neq 0$, *и всякого натурального* $n\geqslant 2$ *выполнено неравенство* $(1+x)^n>1+nx$ *(это неравенство называют неравенством Бернулли).*

1) *Базис индукции. При* $n=2$, *учитывая, что* $x^2>0$, *имеем* $(1+x)^2=1+2x+x^2>1+2x$.
2) *Шаг индукции. Индукционное предположение: допустим, что при* $n=k$ *неравенство справедливо, то есть* $(1+x)^k>1+kx$. *Покажем, что тогда оно выполнено и при* $n=k+1$. *Действительно, так как* $1+x>0$, *то умножив обе части на* $1+x>0$, *получим*

$$(1+x)^k(1+x)>(1+kx)(1+x),$$

и далее:

$$(1+x)^{k+1}=(1+x)^k(1+x)>(1+kx)(1+x)=$$
$$=1+(k+1)x+kx^2>1+(k+1)x,$$

что и требовалось.

Когда индукционный переход осуществляется от меньших параметров к бóльшим, индукция называется прямой. В некоторых случаях применяется обратная индукция, когда параметр уменьшается. Тогда базисом индукции будет утверждение $P(n_0)$ для наибольшего возможного n_0, а индукционным шагом — доказательство $P(k-1)$ при предположении $P(k)$ (или $P(\ell)$ для всех $\ell=k,\ldots,n_0$). В этом случае утверждение $P(n)$ будет верным для всех $n\leqslant n_0$.

Рассмотрим интересный пример, в котором прямой и обратный индукционный шаги чередуются.

Пример 6. *Доказать, что для всех действительных $x_1, \dots, x_n > 0$, $n \geqslant 2$ выполнено неравенство*

$$\sqrt[n]{x_1 \cdots x_n} \leqslant \frac{x_1 + \cdots + x_n}{n},$$

то есть среднее геометрическое чисел не превосходит их среднего арифметического.

1) *Базис индукции. Пусть $n = 2$. Тогда получаем*

$$0 \leqslant (\sqrt{x_1} - \sqrt{x_2})^2 = x_1 + x_2 - 2\sqrt{x_1 x_2},$$

откуда сразу выводим требуемое:

$$\sqrt{x_1 x_2} \leqslant \frac{x_1 + x_2}{2}.$$

2) *Шаг индукции 1. Индукционное предположение: пусть при $n = 2, \dots, k$ неравенство верно. Докажем его для $n = 2k$. Положим*

$$y_1 = \sqrt[k]{x_1 \cdots x_k}, \qquad y_2 = \sqrt[k]{x_{k+1} \cdots x_{2k}},$$
$$z_1 = \frac{x_1 + \cdots + x_k}{k}, \qquad z_2 = \frac{x_{k+1} + \cdots + x_{2k}}{k}.$$

По индукционному предположению $y_1 \leqslant z_1$ и $y_2 \leqslant z_2$. Далее получаем

$$\begin{aligned}\sqrt[2k]{x_1 \cdots x_{2k}} &= \sqrt{\sqrt[k]{x_1 \cdots x_k}\,\sqrt[k]{x_{k+1} \cdots x_{2k}}} = \\ &= \sqrt{y_1 y_2} \leqslant \frac{y_1 + y_2}{2} \leqslant \frac{z_1 + z_2}{2} = \\ &= \frac{\dfrac{x_1 + \cdots + x_k}{k} + \dfrac{x_{k+1} + \cdots + x_{2k}}{k}}{2} = \frac{x_1 + \cdots + x_{2k}}{2k}.\end{aligned}$$

3) *Шаг индукции 2. Индукционное предположение: допустим, что при $n = k + 1$ неравенство справедливо, докажем его для $n = k$. Положим $x_{k+1} = \dfrac{x_1 + \cdots + x_k}{k}$. Заметим, что тогда*

$$x_1 + \cdots + x_k + x_{k+1} = \frac{k+1}{k}(x_1 + \cdots + x_k).$$

По индукционному предположению

$$\sqrt[k+1]{x_1 \cdots x_k x_{k+1}} \leqslant \frac{x_1 + \cdots + x_k + x_{k+1}}{k+1},$$

возведя обе части в степень $k+1$, *получим*

$$x_1 \cdots x_k x_{k+1} \leqslant \left(\frac{x_1 + \cdots + x_k + x_{k+1}}{k+1} \right)^{k+1}$$

или

$$x_1 \cdots x_k \frac{x_1 + \cdots + x_k}{k} \leqslant \frac{(x_1 + \cdots + x_k)^{k+1}}{k^{k+1}}.$$

Сокращая на $\dfrac{x_1 + \cdots + x_k}{k}$, *имеем*

$$x_1 \cdots x_k \leqslant \frac{(x_1 + \cdots + x_k)^k}{k^k}.$$

Извлекая корень k*-й степени, получаем нужное нам:*

$$\sqrt[k]{x_1 \cdots x_k} \leqslant \frac{x_1 + \cdots + x_k}{k}.$$

В обычном понимании слово «индукция» означает переход от частных случаев к некоторому общему утверждению, а «дедукция» — получение результатов для частных случаев из некоторых общих утверждений, законов. В этом смысле метод математической индукции является дедуктивным, с его помощью доказываются общие утверждения (равенства, неравенства и т. д.). Он не даёт способа для выдвижения общей гипотезы или угадывания общего правила или формулы по наблюдениям за отдельными частными случаями. Но этот метод позволяет проверять выдвинутые гипотезы. Неверная гипотеза провалится при проверке базиса или на шаге индукции.

В дискретной математике и в информатике многие классы объектов определяются индуктивно. В таких определениях явно или неявно участвует некоторая функция, задающая «сложность» объекта, и индукция идёт по значениям этой функции. На базисном шаге определяются объекты минимальной сложности (обычно они имеют сложность ноль или один), а индукционный шаг определения заключается в том, что из объектов меньшей сложности с помощью некоторых операций (операторов, конструкций) строятся объекты большей сложности.

Примерами таких классов объектов являются, в частности, префиксные булевы формулы и формулы логики высказываний в главе 3, формулы логики предикатов в главе 7, ориентированные деревья в главе 11, регулярные выражения в главе 16 и частично рекурсивные функции в главе 19.

Для доказательства некоторого свойства объектов индуктивно определённого класса метод математической индукции применяется в следующем виде.

1) Б а з и с и н д у к ц и и состоит в проверке требуемого свойства у объектов минимальной сложности.

2) Ш а г и н д у к ц и и состоит в предположении справедливости доказываемого свойства для всех объектов класса, имеющих сложность меньше k, и проверке того, что все получаемые из них объекты сложности k также обладают требуемым свойством.

Рассмотрим эту схему на примере простых арифметических выражений.

Пример 7. *Пусть* $\boldsymbol{V} = \{x, y, z\}$ *— множество переменных, а* $\boldsymbol{O} = \{+, -, *, /\}$ *— множество операций. Определим индуктивно множество* $\mathcal{A}$ *слов в объединённом алфавите* $\Sigma = \boldsymbol{V} \cup \boldsymbol{O} \cup \{(,)\}$*, называемых* а р и ф м е т и ч е с к и м и ф о р м у л а м и*. Одновременно будем определять натуральное число — меру сложности этих формул, называемую их* г л у б и н о й*. Глубину формулы* φ *обозначим через* $\mathrm{depth}(\varphi)$*.*

1) *Б а з и с и н д у к ц и и. Каждая переменная* $v \in \boldsymbol{V}$ *является арифметической формулой глубины 0, то есть* $v \in \mathcal{A}$ *и* $\mathrm{depth}(v) = 0$*.*

2) *Ш а г и н д у к ц и и. Пусть* φ_1 *и* φ_2 *— арифметические формулы глубины* $\mathrm{depth}(\varphi_1)$ *и* $\mathrm{depth}(\varphi_2)$ *соответственно. Тогда следующие четыре слова*

   (а) $(\varphi_1 + \varphi_2)$, (б) $(\varphi_1 - \varphi_2)$, (в) $(\varphi_1 * \varphi_2)$, (г) $(\varphi_1 \,/\, \varphi_2)$,

   также являются арифметическими формулами из $\mathcal{A}$ *и каждая из этих формул имеет глубину* $\max\{\mathrm{depth}(\varphi_1), \mathrm{depth}(\varphi_2)\} + 1$*.*

Пусть $w = w_1 w_2 \ldots w_n$ — произвольное слово в алфавите Σ. Скажем, что скобки в w р а с с т а в л е н ы п р а в и л ь н о, если для каждого $i \leqslant n$ число левых скобок в слове $w^{(i)} = w_1 w_2 \ldots w_i$ не меньше числа правых скобок, а во всём слове w число левых скобок равно числу правых.

Пример 8. *Докажем, что в каждой арифметической формуле из $\mathcal{A}$ скобки расставлены правильно.*

1) *Базис индукции:* $\operatorname{depth}(\varphi) = 0$. *Формула глубины ноль является переменной $v \in \boldsymbol{V}$. В ней нет скобок и поэтому они расставлены правильно.*

2) *Шаг индукции. Пусть утверждение справедливо для всех формул из $\mathcal{A}$ глубины меньшей k и φ — произвольная формула глубины k. Тогда она имеет одну из четырёх форм (а), (б), (в) или (г).*

   Рассмотрим случай, когда $\varphi = (\varphi_1 + \varphi_2)$. Тогда из определения глубины следует, что $\operatorname{depth}(\varphi_1) < k$ и $\operatorname{depth}(\varphi_2) < k$. По индукционному предположению в обеих формулах φ_1 и φ_2 скобки расставлены правильно. Покажем, что и в φ скобки расставлены правильно. Пусть $\varphi_1 = v_1 v_2 \dots v_{m_1}$ и $\varphi_2 = w_1 w_2 \dots w_{m_2}$. Тогда

$$\varphi = t_1 t_2 \dots t_M = (v_1 v_2 \dots v_{m_1} + w_1 w_2 \dots w_{m_2}),$$

   здесь $M = m_1 + m_2 + 3$, все символы v_i и w_j принадлежат Σ.

   Для каждого $1 < i \leqslant m_1 + 1$ число левых скобок в $t_1 \dots t_i$ на 1 больше числа левых скобок в $v_1 \dots v_{i-1}$, и следовательно, больше числа правых скобок в этом слове, так все они входят в $v_1 \dots v_{i-1}$. Это же справедливо для слова $t_1 t_2 \dots t_{m_1+2}$, заканчивающегося символом $+$. При $m_1 + 2 < i < M$ разница между числом левых и правых скобок в $t_1 \dots t_i$ не меньше 1, так как $t_1 =$ (, а в φ_1 и φ_2 скобки расставлены правильно. Во всём слове φ число левых и правых скобок совпадает, так как к скобкам φ_1 и φ_2 добавилась одна левая и одна правая скобка. Таким образом, в φ скобки расставлены правильно.

   Случаи (б), (в) и (г) рассматриваются аналогично.

§ 2.2. Размещения, перестановки, сочетания

Комбинаторика — раздел математики, изучающий вопросы о том, сколько различных комбинаций, подчинённых тем или иным условиям, можно составить из заданных объектов.

Многие классические задачи комбинаторики являются задачами определения числа способов размещения некоторых объектов по одному в каждом из какого-то количества «ящиков» так, чтобы выполнялись определённые ограничения. Более формально такие задачи

можно сформулировать так. Даны множества X, Y, причём $|X| = n$, $|Y| = m$. Сколько существует функций $f : X \to Y$, удовлетворяющих заданным ограничениям? Здесь элементы Y — объекты, а элементы X — ящики, а каждая функция $f : X \to Y$ определяет для каждого ящика $x \in X$, какой объект $y \in Y$ в него попал.

Итак, размещение множества объектов Y по множеству «ящиков» X — это функция $f : X \to Y$. Заметим, что среди размещаемых объектов могут быть одинаковые, то есть $f(x_1) = f(x_2)$. Например, в несколько ячеек мы можем записать один и тот же символ.

Рассмотрим вначале простой случай, когда на размещения не накладывается никаких ограничений. Такая ситуация возникает, если количество объектов каждого вида может быть сколь угодно большим (точнее — не меньше количества ящиков, чтобы содержимое одних ящиков не ограничивало содержимое других). В этом случае говорят о размещениях с повторениями, так как разные «ящики» могут содержать один и тот же «объект».

Обозначим через F_n^m число всех функций из m-элементного множества X в n-элементное множество Y. Это число называется числом размещений с повторениями из n по m.

Теорема 8. *Если $|Y| = n$ и $|X| = m$, то количество всех функций $f : X \to Y$ равно $F_n^m = n^m$.*

Доказательство. Используем для доказательства индукцию по m. Пусть $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_n\}$. Тогда каждая функция $f : X \to Y$ однозначно определяется последовательностью своих значений $(f(x_1), \ldots, f(x_m))$, а F_n^m — это количество всех таких функций или последовательностей.

База индукции. Ясно, что при $m = 0$ имеется ровно одна функция, нигде не определённая, или, что тоже самое, одна пустая последовательность $()$, то есть $F_n^0 = 1$.

Шаг индукции. Предположим, что при $m = k$ выполнено равенство $F_n^k = n^k$. Докажем, что тогда $F_n^{k+1} = n^{k+1}$.

Действительно, при $m = k + 1$ каждая функция $f : X \to Y$ однозначно задаётся последовательностью $(f(x_1), \ldots, f(x_k), f(x_{k+1}))$. Если положить $X' = \{x_1, \ldots, x_k\}$, то f можно рассматривать как функ-

цию $f' : X' \to Y$, заданную последовательностью $(f(x_1), \ldots, f(x_k))$, которая дополнена одним новым значением $f(x_{k+1})$. Так как $|X'| = k$, то по предположению количество таких функций f' равно $F_n^k = n^k$. Каждая из них имеет ровно n возможных расширений $f(x_{k+1}) = y_i$, $i = 1, \ldots, n$. Поэтому $F_n^{k+1} = F_n^k \cdot n = n^{k+1}$. □

Следствие 9. *Если $|X| = n$, то число всех подмножеств множества X равно $|\mathsf{P}(X)| = 2^n$.*

Доказательство. Пусть $X = \{x_1, \ldots, x_n\}$. Сопоставим каждому подмножеству $A \subseteq X$ его характеристическую функцию I_A в X (определение 16 на стр. 30). Ясно, что это сопоставление взаимно однозначное. Действительно, если $A' \neq A''$, то имеется элемент $x_i \in A' \div A''$ и тогда $f_{A'}(x_i) \neq f_{A''}(x_i)$. Таким образом, число всех подмножеств X равно числу всех характеристических функций на множестве X, то есть функций вида $f : X \to \{0, 1\}$. По доказанной теореме это число равно 2^n. □

Следствие 10. *Число всех слов длины m в алфавите из n символов равно n^m.*

Доказательство. В данном случае мы размещаем с повторениями n символов на m мест. □

Найдём теперь число размещений, для которых все «ящики» содержат разные «объекты». Эта ситуация возникает, когда все размещаемые «объекты» уникальны, то есть, поместив «объект» в один «ящик», мы лишаемся возможности поместить такой же «объект» в другой «ящик». Такие размещения соответствуют разнозначным функциям и называются размещениями без повторений, так как содержимое «ящиков» (то есть значения функции) не могут повторяться.

Обозначим через A_n^m количество всех разнозначных функций из m-элементного множества X в n-элементное множество Y. Эти функции называются размещениями без повторений из n по m, а число A_n^m — их количеством.

Теорема 11. *Если $|Y| = n$, $|X| = m$ и $n \leqslant m$, то число всех разнозначных функций $f : X \to Y$ равно*

$$A_n^m = n(n-1)\cdots(n-m+1) = \frac{n!}{(n-m)!}. \tag{2}$$

Доказательство. Используем индукцию по m (для каждого фиксированного n).

Базис индукции. Поскольку при $m = 0$ снова имеется одна, нигде не определённая функция, и мы имеем $A_n^0 = 1 = {}^{n!}/_{(n-0)!}$.

Шаг индукции. Предположим, что при $m = k$ выполнено равенство $A_n^k = n(n-1)\cdots(n-k+1)$. Докажем, что тогда

$$A_n^{k+1} = A_n^k \cdot (n-k) = n(n-1)\cdots(n-k+1)(n-k).$$

Действительно, как и в предыдущей теореме, каждая разнозначная функция $f : X \to Y$ является расширением некоторой разнозначной функции $f' : X' \to Y$ значением $f(x_{k+1})$ (напомним, что $X' = X \setminus \{x_{k+1}\}$). При этом в качестве этого значения можно взять любой элемент Y, не являющийся значением f', то есть любой элемент из множества $Y \setminus \{f(x_1), \ldots, f(x_k)\}$. При $k < m$ количество таких элементов равно $m - k$, так как все $f(x_1), \ldots, f(x_k)$ попарно различны. Тогда каждую разнозначную функцию $f' : X' \to Y$ можно расширить $n - k$ способами и, следовательно, $A_n^{k+1} = A_n^k(n-k)$. □

При $m > n$ разнозначных функций $f : X \to Y$ не существует (почему?) и $A_n^m = 0$, но в этом случае средняя часть формулы (2) также справедлива, поскольку один из сомножителей в ней равен 0. Правая часть формулы (2) в этом случае не имеет смысла, так как для отрицательных чисел факториал неопределён.

В качестве простого следствия доказанной теоремы получаем формулу для числа перестановок.

Теорема 12. *Если $|X| = n$, то число всех перестановок $f : X \to X$ равно $n!$.*

Ещё одним важным комбинаторным приёмом является нахождение подмножеств с заданным количеством элементов. Они называются сочетаниями.

Количество всех k-элементных подмножеств n-элементного множества обозначим через C_n^k (часто используется также обозначение $\binom{n}{k}$).Это число называется числом сочетаний из n по k. Такой объект возникает, когда мы хотим разместить в «ящики» несколько

одинаковых «объектов»: мы должны выбрать из n «ящиков» k штук, куда попадёт по одному «объекту». Из определения сразу получаем

Следствие 13. *$C_n^k = 0$ при $k < 0$ или $k > n$.*

В остальных случаях верно следующее.

Теорема 14. *При $n \geqslant k \geqslant 0$*

$$C_n^k = \frac{A_n^k}{k!} = \frac{n!}{k!(n-k)!}$$

Доказательство. При $n = k = 0$ у пустого множества имеется одно (пустое) подмножество. Поэтому

$$C_0^0 = 1 = \frac{0!}{0! \times 0!}$$

(напомним, что $0! = 1$).

Пусть теперь $|Y| = n \geqslant 1$. Тогда каждая разнозначная функция $f : \{1, \ldots, k\} \to Y$ определяет k-элементное подмножество

$$\operatorname{rng} f = \{f(1), \ldots, f(k)\} \subseteq Y.$$

При этом одно и тоже такое подмножество получается при любой перестановке элементов $\operatorname{rng} f$. Всего таких перестановок имеется $k!$ (по предыдущей теореме), а разнозначных функций $f : \{1, \ldots, k\} \to Y$ по теореме 11 на стр. 50 существует A_n^k. Отсюда получаем

$$C_n^k = \frac{A_n^k}{k!} = \frac{n(n-1)\ldots(n-k+1)}{k!} = \frac{n!}{k!(n-k)!}. \qquad \square$$

Непосредственным следствием этой теоремы является свойство «симметричности» сочетаний: $C_n^k = C_n^{n-k}$, а также рекуррентная формула

$$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}, \tag{3}$$

позволяющая организовать их эффективное вычисление путём последовательного получения элементов треугольника Паскаля:

$$\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
\ldots & & \ldots & & \ldots & & \ldots & & \ldots & & \ldots
\end{array}$$

В n-й строке этого треугольника стоят числа C_n^0, C_n^1, …, C_n^k, …, C_n^n и каждое из них является суммой двух стоящих над ним чисел предыдущей строки. Эти числа называются биномиальными коэффициентами, так как входят в формулу бинома Ньютона, выражающую n-ю степень бинома $x + y$:

$$(x+y)^n = \sum_{k=0}^{n} C_n^k x^k y^{n-k}.$$

Справедливость этой формулы следует из того, что коэффициент при $x^k y^{n-k}$ равен количеству способов, с помощью которых из n сомножителей $(x+y)(x+y)\ldots(x+y)$ можно выбрать k множителей, дающих x (остальные $n-k$ множителей автоматически дают y).

Укажем несколько простых следствий этой формулы. Положив в ней $x = 1$, $y = 1$, получаем:

$$\sum_{k=0}^{n} C_n^k = 2^n.$$

Так как сумма слева определяет число всех подмножеств n-элементного множества, то это ещё одно доказательство следствия 9 на стр. 50.

При $x = -1$, $y = 1$ бином Ньютона даёт равенство числа подмножеств чётной и нечётной мощности:

$$\sum_{\substack{k=0,\ldots,n \\ k \text{ чётное}}} C_n^k = \sum_{\substack{k=0,\ldots,n \\ k \text{ нечётное}}} C_n^k.$$

§ 2.3. Принцип включения и исключения

Во многих ситуациях для подсчёта числа объектов, обладающих тем или иным набором свойств, используется следующий принцип включения и исключения.

Пусть имеется N объектов, каждый из которых может обладать (или не обладать) свойствами $p_1, p_2, \ldots, p_n$. Через p_i' будем обозначать свойство, дополнительное к свойству p_i, то есть объект обладает свойством p_i' в точности тогда, когда он не обладает свойством p_i. Если P — множество этих свойств, то с помощью $N(P)$ обозначим количество объектов, обладающих всеми свойствами из множества P, другие свойства при этом в расчёт не принимаются, обладают ли ими объекты — неважно. Например, $N(\{p_1, p_3', p_4\})$ — это число объектов, обладающих свойствами p_1 и p_4 и не обладающих свойством p_3, а $N(\varnothing) = N$. Круглые скобки при наличии фигурных будем опускать: $N\{p_1, p_3', p_4\}$.

Теорема 15 (Принцип включения и исключения)**.** *Число объектов, не обладающих ни одним из свойств $p_1, \ldots, p_n$, равно*

$$N\{p_1', \ldots, p_n'\} = \sum_{P \subseteq \{p_1, \ldots, p_n\}} (-1)^{|P|} N(P) = \\ = N - \sum_{i=1}^{n} N\{p_i\} + \sum_{1 \leqslant i < j \leqslant n} N\{p_i, p_j\} - \ldots$$

Доказательство. Проведём доказательство индукцией по числу свойств, независимо от самих этих свойств.

Базис индукции. При $n = 1$ очевидно $N\{p_1'\} = N - N\{p_1\}$.

Шаг индукции. Предположим, что для k свойств теорема справедлива, то есть

$$N\{p_1', \ldots, p_k'\} = \sum_{P \subseteq \{p_1, \ldots, p_k\}} (-1)^{|P|} N(P). \qquad (4)$$

Рассмотрим теперь только те объекты, которые обладают свойством p_{k+1}. Применим к ним эту же формулу:

$$N\{p'_1,\dots,p'_k,p_{k+1}\} = \sum_{P\subseteq\{p_1,\dots,p_k\}} (-1)^{|P|} N(P\cup\{p_{k+1}\}). \qquad (5)$$

Заметим, что поскольку в (5) множество P может содержать только $p_1,\dots,p_k$ и не может p_{k+1}, то $|P\cup\{p_{k+1}\}| = |P|+1$, поэтому получаем

$$N\{p'_1,\dots,p'_k,p_{k+1}\} = -\sum_{P\subseteq\{p_1,\dots,p_k\}} (-1)^{|P\cup\{p_{k+1}\}|} N(P\cup\{p_{k+1}\})$$

или

$$N\{p'_1,\dots,p'_k,p_{k+1}\} = -\sum_{\substack{P\subseteq\{p_1,\dots,p_{k+1}\}\\ p_{k+1}\in P}} (-1)^{|P|} N(P). \qquad (6)$$

Вычтем (6) из (4):

$$N\{p'_1,\dots,p'_k\} - N\{p'_1,\dots,p'_k,p_{k+1}\} = \\ = \sum_{P\subseteq\{p_1,\dots,p_k\}} (-1)^{|P|} N(P) + \sum_{\substack{P\subseteq\{p_1,\dots,p_{k+1}\}\\ p_{k+1}\in P}} (-1)^{|P|} N(P).$$

Нетрудно видеть, что левая часть в последнем равенстве равняется $N\{p'_1,\dots,p'_k,p'_{k+1}\}$, а правая:

$$\sum_{P\subseteq\{p_1,\dots,p_{k+1}\}} (-1)^{|P|} N(P),$$

что и требуется. □

В качестве примера рассмотрим следующую задачу.

Пример 9. *В студенческой группе* 25 *студентов. Из них* 15 *знают язык Pascal,* 10 *— язык C и* 14 *— язык Basic. Кроме того,* 7 *студентов знают Pascal и C,* 10 *студентов — Pascal и Basic,* 8 *студентов — C и Basic, а* 5 *студентов знают все три языка. Сколько студентов не знают ни одного из трёх языков программирования?*

Обозначив через p_1 — свойство «знать Pascal», через p_2 — свойство «знать C» и через p_3 — свойство «знать Basic», мы можем записать данные

задачи следующим образом: $N = 25$, $N\{p_1\} = 15$, $N\{p_2\} = 10$, $N\{p_3\} = 14$, $N\{p_1, p_2\} = 7$, $N\{p_1, p_3\} = 10$, $N\{p_2, p_3\} = 8$, $N\{p_1, p_2, p_3\} = 5$. *Тогда по формуле включения и исключения число студентов, не знающих ни одного языка программирования, равно*

$$N\{p_1', p_2', p_3'\} = 25 - (15 + 10 + 14) + (7 + 10 + 8) - 5 = 6.$$

Задачи

26. Доказать, что $1^4 + 2^4 + \ldots + n^4 = \frac{1}{30} n(n+1)(2n+1)(3n^2+3n-1)$.

27. Доказать, что $1 \cdot 2 + 2 \cdot 3 + \ldots + n(n+1) = \frac{n(n+1)(n+2)}{3}$.

28. Доказать, что $1 + x + x^2 + x^3 + \ldots + x^n = \frac{x^{n+1}-1}{x-1}$.

29. Доказать, что n различных прямых на плоскости разбивают её на области, которые можно закрасить белой и чёрной красками так, что смежные области будут закрашены разными красками.

30. Найти ошибку в следующем «доказательстве по индукции» утверждения: для всех $n \geqslant 1$ справедливо неравенство $3^n > 3(n+1) + 1$.

Пусть для некоторого $k \geqslant 1$ неравенство справедливо, то есть

$$3^k > 3(k+1) + 1. \tag{7}$$

Докажем, что оно верно и для $n = k+1$, то есть $3^{k+1} > 3(k+2) + 1$. Для этого заметим, что для всех $k \geqslant 1$ верно неравенство $2 \cdot 3^k > 3$. Прибавив его левую и правую часть к соответствующим частям неравенства (7), получим $3^k + 2 \cdot 3^k > 3(k+1) + 1 + 3$ или $3^{k+1} > 3(k+2) + 1$, что и требовалось.

Установите, при каких n на самом деле справедливо это неравенство.

31. Найти ошибку в следующем «доказательстве по индукции» утверждения: в каждом стаде из n коров все животные одного цвета.

Базис: если $n = 1$, то стадо состоит из одной коровы, поэтому все коровы одного цвета. Шаг: предположим, что для n утверждение доказано. Рассмотрим стадо из $n+1 > 1$ коровы. Удалим из этого стада корову x, по индукционному предположению все оставшиеся n коров будут одного цвета. Удалим из этого стада другую корову y, по индукционному предположению все оставшиеся n коров снова будут одного цвета. Тогда получаем, что цвета коров x и y совпадают с цветами всех остальных коров, поэтому все коровы в стаде одного цвета.

32. Числа Фибоначчи F_i, $i \in \omega$, определяются следующим образом: $F_i = i$ при $i < 2$, $F_i = F_{i-1} + F_{i-2}$ в противном случае. Доказать, что F_i чётно тогда и только тогда, когда i делится на 3.

33. Доказать для чисел Фибоначчи равенство $F_{i-1}F_{i+1} = F_i^2 + (-1)^i$ для $i > 0$.

34. Число e является суммой следующего бесконечного ряда: $e = \sum\limits_{i=0}^{\infty} {}^1\!/_{i!}$. Пусть e_n — сумма первых n членов этого ряда: $e_n = \sum\limits_{i<n} {}^1\!/_{i!}$. Доказать, что A_n — общее количество размещений без повторений из n предметов по любому количеству мест m, равняется $e_n n!$. Сделать вывод, что при больши́х n оно приближённо равно $en!$. ▼

35. Доказать тождества:

(а) $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$; (б) $nC_{n-1}^{k-1} = kC_n^k$; (в) $C_n^k C_{n-k}^{m-k} = C_m^k C_n^m$. ▼

36. Доказать с помощью математической индукции формулу бинома Ньютона. ▼

37. Доказать, что $C_n^0 < C_n^1 < \ldots < C_n^{[n/2]} = C_n^{]n/2[} > C_n^{]n/2[+1} > \ldots > C_n^n$, где $]x[$ означает округление вверх. ▼

38. Доказать, что число разбиений n-элементного множества на k подмножеств, первое из которых содержит n_1 элементов, второе — n_2 элементов, …, k-е — n_k элементов, равно $\frac{n!}{n_1!n_2!\ldots n_k!}$. ▼

39. Преподаватель рассчитывает читать один и тот же курс в течение 20 лет. Чтобы не наскучить студентам, он решил рассказывать им каждый год три анекдота, причём этот набор из трёх анекдотов не должен повторяться. Каково минимальное число анекдотов, которые он должен приготовить? ▼

40. На острове живёт племя туземцев, у которых набор зубов во рту состоит максимально из 30 зубов. При этом на острове нет двух жителей с одинаковыми наборами зубов (наборы одинаковы, если каждый зуб у обоих либо одновременно присутствует, либо одновременно отсутствует). Может ли на этом острове быть больше жителей чем в

(а) Торжке? (в) Москве? (д) всём мире? ▼

(б) Твери? (г) России?

41. Доказать тождество Коши: ▼

$$C_{n+m}^k = \sum_{i=0}^{i=k} C_n^i C_m^{k-i}.$$

42. Доказать, что число C_n^k нечётно тогда и только тогда, когда $\rho^{-1}(k) \subseteq \rho^{-1}(n)$, где функция ρ определена в примере 2 на стр. 33. Иначе говоря, если в двоичной записи k на каком-то разряде стоит единица, то и в двоичной записи n на этом разряде должна стоять единица. Указание: использовать индукцию по n, представить $n = 2^u + m$ для $1 \leqslant m \leqslant 2^u$ и применить тождество Коши. ▼

43. Мультимножеством называется неупорядоченный набор элементов, каждый из которых может встречаться в нём сколько угодно раз. Два мультимножества равны, если каждый элемент встречается в них одно и тоже количество раз. Например, мультимножество $\{1, 2, 1, 3\}$ равно мультимножеству $\{3, 1, 1, 2\}$ и не равно мультимножеству $\{1, 2, 2, 3\}$.

Доказать, что число способов, которыми можно породить k-элементное мультимножество, имея n разных элементов, равно C_{n+k-1}^k. ▼

44. В кондитерском магазине продаются четыре сорта пирожных: заварные, песочные, «картошка» и бисквитные. Сколькими способами можно купить 6 пирожных?

45. Назовём два исхода первенства России по футболу совпадающими в главном, если в этих исходах совпадают обладатели золотых, серебряных и бронзовых медалей, а также две команды, покидающие премьер-лигу (то есть занявшие два последних места). Найти число различных в главном исходов (напомним, что в первенстве участвуют 16 команд).

46. Сколько существует способов расположить n шаров, из них — m чёрных, остальные — белые, в один ряд, чтобы никакие два чёрных шара не оказались рядом?

47. За круглым столом короля Артура сидят 12 рыцарей. Каждый из них враждует со своими соседями. Нужно выбрать 5 рыцарей, чтобы освободить принцессу. Сколькими способами это можно сделать так, чтобы среди выбранных рыцарей не оказалось врагов?

Решить эту задачу в случае, когда из n рыцарей за столом нужно выбрать k рыцарей.

48. Доказать принцип включения и исключения в теоретико-множественной форме: если $A_1, \ldots, A_n$ — это конечные множества, то

$$\left|\bigcup_{i=1}^{n} A_i\right| = -\sum_{\substack{I\subseteq\{1,\ldots,n\}\\ I\neq\varnothing}} (-1)^{|I|}\left|\bigcap_{i\in I} A_i\right|.$$

49. Сколько в первой сотне положительных натуральных чисел не делится ни на одно из чисел 2, 3, 5? А в первой тысяче чисел?

50. Определить, сколько целочисленных решений имеет следующая система:

$$\begin{cases} x+y+z=11\\ 0\leqslant x\leqslant 4\\ 0\leqslant y\leqslant 3\\ 0\leqslant z\leqslant 7 \end{cases}$$

51. Найти число перестановок из n-элементов, при которых ни один элемент не остаётся в первоначальном положении.

Глава 3

Булевы функции и их представления

Краткое содержание: класс $\mathcal{P}_n$ булевых функций от n переменных, геометрическое и табличное представления булевых функций, булевы функции от одной и двух переменных, булевы формулы и формулы логики высказываний, решение логических задач с помощью формул.

Ключевые слова: булева функция, n-мерный куб $\mathbb{B}^n$, геометрическое представление, таблица булевой функции, отрицание, конъюнкция, дизъюнкция, импликация, сложение по модулю два, эквивалентность, штрих Шеффера, стрелка Пирса, булева формула, глубина формулы, подформула, функция, задаваемая формулой, формула логики высказываний, интерпретация, тождественно истинная, тождественно ложная и выполнимая формула.

§ 3.1. Булевы функции и их представления

Булевы функции[1] названы в честь английского математика XIX века Дж. Буля, который впервые применил алгебраические методы для решения логических задач. Они образуют самый простой нетривиальный класс дискретных функций — их аргументы и значения могут принимать всего два значения (если мощность множества значений функции равна 1, то это тривиальная функция — константа). С другой стороны, этот класс достаточно богат и его функции имеют много интересных свойств. Булевы функции находят применение в логике, электротехнике, многих разделах информатики.

Определение 24 (Булева функция)**.** *Булевой функцией от n переменных (или аргументов) называется n-местная функция на множестве $\mathbb{B} = \{0, 1\}$.*

Из определения n-местной функции 17 на стр. 31 получаем, что аргументами n-местной булевой функции являются двоичные последовательности (упорядоченные n-ки, вектора) длины n, то есть элементы множества

$$\mathbb{B}^n = \underbrace{\mathbb{B} \times \mathbb{B} \times \cdots \times \mathbb{B}}_{n \text{ раз}}.$$

Отдельные аргументы x_i, $1 \leqslant i \leqslant n$, могут принимать одно из двух значений 0 или 1, и значением функции на любом наборе из $\mathbb{B}^n$ также может быть 0 или 1. Обозначим через $\mathcal{P}_n$ множество всех булевых функций от n переменных. Нетрудно подсчитать их число.

Теорема 16. $|\mathcal{P}_n| = 2^{2^n}$.

Доказательство. Действительно, по теореме 8 на стр. 49 число функций из k-элементного множества A в m-элементное множество $\mathbb{B}$ равно m^k. В нашем случае $|\mathbb{B}| = 2$, а $A = \mathbb{B}^n$. Тогда $m = 2$ и $k = |\mathbb{B}^n| = 2^n$. Отсюда следует утверждение теоремы. □

[1] В отечественной литературе их также часто называют функциями алгебры логики.

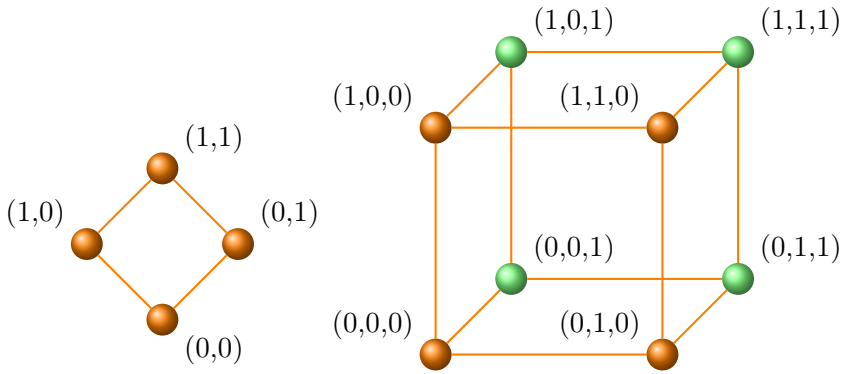


Рис. 1: Единичные кубы $\mathbb{B}^2$ и $\mathbb{B}^3$.

Имеется несколько различных способов представления и интерпретации булевых функций. В этом разделе мы рассмотрим геометрическое и табличное представления, а также представление с помощью логических формул. В главе 4 будет показано, как булевы функции можно представлять с помощью формул специального вида — дизъюнктивных и конъюнктивных нормальных форм и многочленов Жегалкина. Кроме того, в главах 13 и 14 будет рассмотрено ещё два способа представления булевых функций: логические схемы и упорядоченные бинарные диаграммы решений

Множество $\mathbb{B}^n$ можно рассматривать как множество вершин *единичного n-мерного куба*. В самом деле, единичный n-мерный куб состоит из всевозможных векторов вида $(x_1, \ldots, x_n)$, у которых каждая компонента x_i принадлежит отрезку $[0; 1]$. Следовательно, вершинами этого куба являются точки n-мерного пространства $(x_1, \ldots, x_n)$, у которых все компоненты равны граничным значениям, то есть 0 или 1. На рис. 1 представлены единичные кубы $\mathbb{B}^2$ и $\mathbb{B}^3$.

При этом существует естественное взаимно однозначное соответствие между подмножествами вершин n-мерных единичных кубов и булевыми функциями от n переменных: подмножеству $A \subseteq \mathbb{B}^n$ соответствует его характеристическая функция I_A в $\mathbb{B}^n$. Например, «задней» грани куба $\mathbb{B}^3$ (её вершины на рисунке изображены зелёным

x_1	$\cdots$	x_{n-1}	x_n	$f(x_1,\ldots,x_n)$
0	$\cdots$	0	0	$f(0,\ldots,0,0)$
0	$\cdots$	0	1	$f(0,\ldots,0,1)$
0	$\cdots$	1	0	$f(0,\ldots,1,0)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
1	$\cdots$	1	0	$f(1,\ldots,1,0)$
1	$\cdots$	1	1	$f(1,\ldots,1,1)$

Рис. 2: Табличное представление функции $f(x_1,\ldots,x_n)$.

цветом) соответствует функция f:

$$f(0,0,1) = f(0,1,1) = f(1,0,1) = f(1,1,1) = 1,$$
$$f(0,0,0) = f(0,1,0) = f(1,0,0) = f(1,1,0) = 0.$$

Очевидно, что указанное соответствие действительно взаимно однозначное: каждая булева функция f от n переменных является характеристической функцией подмножества

$$A_f = \{(x_1,\ldots,x_n) : f(x_1,\ldots,x_n) = 1\}$$

вершин $\mathbb{B}^n$. Например, функция, тождественно равная нулю, задаёт пустое множество $\varnothing \subset \mathbb{B}^n$, а функция, тождественно равная единице, задаёт множество всех вершин $\mathbb{B}^n$.

Булевы функции от небольшого числа аргументов удобно представлять с помощью таблиц. Таблица для функции $f(x_1,\ldots,x_n)$ имеет $n+1$ столбец. В первых n столбцах указываются значения аргументов $x_1,\ldots,x_n$, а в $(n+1)$-м столбце — значение функции на этих аргументах — $f(x_1,\ldots,x_n)$ (рис. 2).

Наборы аргументов в строках обычно располагаются в *лексикографическом* порядке: $(\alpha_1,\ldots,\alpha_n)$ предшествует $(\beta_1,\ldots,\beta_n)$, если существует такое $i \in [1,n]$, что $\alpha_j = \beta_j$ при $1 \leqslant j < i$, но $\alpha_i < \beta_i$. Если эти наборы рассматривать как записи чисел в двоичной системе счисления, то первая строка представляет число 0, вторая — 1, третья — 2, ..., а последняя — $2^n - 1$.

x_1	f_1	f_2	f_3	f_4
0	0	1	0	1
1	0	1	1	0

Рис. 3: Булевы функции от одной переменной.

При больши́х n табличное представление становится громоздким, например, для функции от 10 переменных потребуется таблица с 1024 строками. Но для малых n оно достаточно наглядно.

§ 3.2. Булевы функции от одной и двух переменных

Представим вначале в табличном виде все булевы функции от одной переменной. Как мы знаем, их всего четыре (рис. 3). В этой таблице представлены следующие функции:

1) $f_1(x_1) = 0$ — константа 0;
2) $f_2(x_1) = 1$ — константа 1;
3) $f_3(x_1) = x_1$ — тождественная функция;
4) $f_4(x_1) = \neg x_1$ — отрицание x_1 (в литературе используется также обозначение $\overline{x}_1$).

На рис. 4 на следующей странице представлены все 16 функций от двух переменных. Многие из этих функций часто используются в качестве «элементарных» и имеют собственные обозначения.

1) $f_1(x_1, x_2) = 0$ — константа 0;
2) $f_2(x_1, x_2) = 1$ — константа 1;
3) $f_3(x_1, x_2) = x_1$ — функция, равная первому аргументу;
4) $f_4(x_1, x_2) = \neg x_1$ — отрицание x_1;
5) $f_5(x_1, x_2) = x_2$ — функция, равная второму аргументу;

x_1	x_2	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
0	0	0	1	0	1	0	1	0	0	1	0	1	1	1	0	0	1
0	1	0	1	0	1	1	0	0	1	1	1	0	1	0	1	0	0
1	0	0	1	1	0	0	1	0	1	0	1	0	1	0	0	1	1
1	1	0	1	1	0	1	0	1	1	1	0	1	0	0	0	0	1

Рис. 4: Булевы функции от двух переменных.

6) $f_6(x_1, x_2) = \neg x_2$ — отрицание x_2;

7) $f_7(x_1, x_2) = (x_1 \wedge x_2)$ — конъюнкция, читается «x_1 и x_2» (используются также обозначения $(x_1 \,\&\, x_2)$, $(x_1 x_2)$, $\min(x_1, x_2)$);

8) $f_8(x_1, x_2) = (x_1 \vee x_2)$ — дизъюнкция, читается «x_1 или x_2» (используется также обозначение $\max(x_1, x_2)$);

9) $f_9(x_1, x_2) = (x_1 \to x_2)$ — импликация, читается «x_1 влечёт x_2» или «из x_1 следует x_2» (используются также обозначения $(x_1 \supset x_2)$ и (if x_1 then x_2)). Импликация некоммутативна: поменяв местами её аргументы мы получим другую функцию — $f_{16}(x_1, x_2) = (x_2 \to x_1)$;

10) $f_{10}(x_1, x_2) = (x_1 \oplus x_2)$ — сложение по модулю 2, читается «x_1 плюс x_2»;

11) $f_{11}(x_1, x_2) = (x_1 \leftrightarrow x_2)$ — эквивалентность, читается «x_1 эквивалентно (равносильно) x_2» (используется также обозначение $(x_1 \equiv x_2)$);

12) $f_{12}(x_1, x_2) = (x_1 \uparrow x_2)$ — штрих Шеффера (антиконъюнкция), иногда читается как «не x_1 и x_2»;

13) $f_{13}(x_1, x_2) = (x_1 \downarrow x_2)$ — стрелка Пирса (антидизъюнкция), иногда читается как «не x_1 или x_2».

В качестве элементарных функций будем также рассматривать нульместные функции-константы 0 и 1.

Почти все языки программирования дают возможность применять булевы функции: отрицание, конъюнкцию, дизъюнкцию, иногда

и другие. Способ обозначения булевых функций в разных языках отличается. Например, в языке C и производных от него отрицание, конъюнкция и дизъюнкция обозначаются с помощью знаков `!`, `&&` и `||` соответственно. В других языках (скажем, в языке SQL, параграф 8.3) для этих же целей применяются английские слова `not`, `and` и `or`.

Отметим, что значения функций $f_1(x_1, x_2)$ и $f_2(x_1, x_2)$ фактически не зависят от значений обоих аргументов, значения функций $f_3(x_1, x_2)$ и $f_4(x_1, x_2)$ не зависят от значений аргумента x_2, а значения функций $f_5(x_1, x_2)$ и $f_6(x_1, x_2)$ не зависят от значений аргумента x_1. Дадим более точное определение такой ситуации.

Определение 25 (Фиктивные и существенные аргументы)**.** *Мы будем говорить, что булева функция $f(x_1, \ldots, x_i, \ldots, x_n)$ не зависит от аргумента x_i, если для каждого набора значений $\sigma_1, \ldots, \sigma_{i-1}, \sigma_{i+1}, \ldots, \sigma_n$ остальных аргументов f выполнено следующее равенство*

$$f(\sigma_1, \ldots, \sigma_{i-1}, 0, \sigma_{i+1}, \ldots, \sigma_n) = f(\sigma_1, \ldots, \sigma_{i-1}, 1, \sigma_{i+1}, \ldots, \sigma_n).$$

Такой аргумент x_i называется фиктивным. Нефиктивные аргументы функции f называются существенными.

Функции $f_1(x_1, \ldots, x_n)$ и $f_2(x_1, \ldots, x_m)$ мы будем считать равными, если функцию f_2 можно получить из функции f_1 путём добавления и удаления фиктивных аргументов, не изменяя порядка остальных.

Например, равными являются одноместная функция $f_3(x_1)$ и двухместная функция $f_3(x_1, x_2)$, так как вторая получается из первой добавлением фиктивного аргумента x_2. Мы не будем различать равные функции и, как правило, будем использовать для обозначения равных функций одно и то же имя функции. В частности, это позволяет считать, что во всяком конечном множестве функций все функции зависят от одного и того же множества переменных: если одна функция имеет аргументов меньше чем другая, к ней можно добавить фиктивные аргументы, сравняв их количество.

§ 3.3. Формулы

Как мы видели, геометрическое и табличное представления булевых функций подходят лишь для функций с небольшим числом аргументов. Формулы позволяют удобно представлять многие функции от большего числа аргументов и оперировать различными представлениями одной и той же функции.

Пусть $\mathcal{B}$ — некоторое (конечное или бесконечное) множество булевых функций. Зафиксируем некоторое счётно бесконечное множество переменных $\boldsymbol{V} = \{x_1, x_2, \ldots\}$. Определим множество ф о р м у л над $\mathcal{B}$ с переменными из $\boldsymbol{V}$. Одновременно будем определять числовую характеристику $\mathrm{depth}(\Phi)$ формулы Φ, называемую её г л у б и н о й, и множество её п о д ф о р м у л.

Определение 26 (Префиксная булева формула)**.** *П р е ф и к с н а я ф о р м у л а — это слово в алфавите* $V \cup \mathcal{B} \cup \{«(», «)», «,»\}$, *определяемое по индукции:*

1) *Б а з и с и н д у к ц и и. Каждая переменная* $x_i \in \boldsymbol{V}$ *и каждая константа* $c \in \mathcal{B}$ *является формулой глубины 0, то есть* $\mathrm{depth}(x_i) = \mathrm{depth}(c) = 0$. *Множество её подформул состоит из неё самой.*

2) *Ш а г и н д у к ц и и. Пусть* $f \in \mathcal{B}$ *— n-местная функция,* $\Phi_1, \ldots, \Phi_m$ *— формулы. Тогда слово* Φ *вида* $f(\Phi_1, \ldots, \Phi_m)$ *тоже является формулой, её глубина равна*

$$\mathrm{depth}(\Phi) = \max\{\mathrm{depth}(\Phi_i) : i = 1, \ldots, m\} + 1,$$

а множество подформул Φ *включает саму формулу* Φ *и все подформулы формул* $\Phi_1, \ldots, \Phi_m$.

Каждой формуле Φ с переменными $x_1, \ldots, x_n$ сопоставим булеву функцию f_Φ, которую эта формула задаёт, используя индукцию по глубине формулы.

1) Базис индукции. Пусть $\operatorname{depth}(\Phi) = 0$. Тогда $\Phi = x_i \in \boldsymbol{V}$ или $\Phi = c \in \mathcal{B}$. В первом случае Φ задаёт функцию $f_\Phi(x_i) = x_i$, во втором — функцию, тождественно равную константе c.

2) Шаг индукции. Пусть Φ — произвольная формула глубины $\operatorname{depth}(\Phi) = k+1$. Тогда $\Phi(x_1, \ldots, x_n) = f(\Phi_1, \ldots, \Phi_m)$, где $f \in \mathcal{B}$ и $\Phi_1, \ldots, \Phi_m$ — формулы, для которых

$$\max\{\operatorname{depth}(\Phi_i) : 1 \leqslant i \leqslant m\} = k.$$

По индукционному предположению этим формулам уже сопоставлены функции $g_1, \ldots, g_m$ соответственно. Тогда формула Φ задаёт функцию

$$f_\Phi(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$

Далее мы в основном будем рассматривать формулы над множеством элементарных функций $\mathcal{B}_e = \{0, 1, \neg, \wedge, \vee, \to, \oplus\}$. Изредка к ним будут добавляться $\leftrightarrow$, $\uparrow$ и $\downarrow$. Все эти функции, кроме констант, называются логическими или булевыми связками (говорят также — логическими или булевыми операциями).

Для классических бинарных булевых связок $\wedge, \vee, \to, \oplus, \leftrightarrow, \uparrow, \downarrow$ кроме префиксного формата используется инфиксная запись, когда знак функции пишется не перед аргументами, а между ними: например, вместо $\oplus(\Phi, \Psi)$ пишут $(\Phi \oplus \Psi)$. После отрицания скобки обычно не ставят: пишут $\neg x$ вместо $\neg(x)$. Точное определение будет дано в следующем параграфе для формул логики высказываний.

Глубина при инфиксной записи определяется так:

$$\operatorname{depth}(\Phi_1 \circ \Phi_2) = 1 + \max\{\operatorname{depth}(\Phi_1), \operatorname{depth}(\Phi_2)\}.$$

По определению инфиксной записи слова́ $\Phi_1 = \neg(x_1 \wedge \neg x_2)$ и $\Phi_2 = ((x_1 \vee \neg\neg x_2) \to (x_3 \leftrightarrow (x_1 \downarrow \neg x_2)))$ являются формулами. Глубина Φ_1 равна 3, а глубина Φ_2 равна 4. Слова $\neg x_1 \oplus (\neg x_2 \vee x_3)$, $(x_1 \neg \wedge x_2)$ и $(x_1 \oplus x_2 \oplus x_3)$ формулами не являются (почему?).

x_1	x_2	x_3	$((x_1$	$\vee$	$\neg$	$\neg$	$x_2)$	$\to$	$(x_3$	$\leftrightarrow$	$(x_1$	$\downarrow$	$\neg$	$x_2)))$
0	0	0	0	0	0	1	0	1	0	1	0	0	1	0
0	0	1	0	0	0	1	0	1	1	0	0	0	1	0
0	1	0	0	1	1	0	1	0	0	0	0	1	0	1
0	1	1	0	1	1	0	1	1	1	1	0	1	0	1
1	0	0	1	1	0	1	0	1	0	1	1	0	1	0
1	0	1	1	1	0	1	0	0	1	0	1	0	1	0
1	1	0	1	1	1	0	1	1	0	1	1	0	0	1
1	1	1	1	1	1	0	1	0	1	0	1	0	0	1

Рис. 5: Функция f_{Φ_2}.

Для определения функции, задаваемой формулой, удобно использовать таблицу, строки которой соответствуют наборам значений переменных, а в столбце под знаком каждой логической связки стоят значения функции, задаваемой соответствующей подформулой. Например, для формулы Φ_2 функция f_{Φ_2} задаётся выделенным столбцом $\to$ таблицы на рис. 5.

§ 3.4. Булевы функции и логика высказываний

Как мы уже отметили, Дж. Буль ввёл булевы функции для решения логических задач.

Определение 27 (Высказывание)**.** *Высказыванием называют повествовательное предложение (фразу), о котором осмысленно спросить (и получить ответ), истинно оно или ложно.*

Заметим, что признание предложения высказыванием не зависит от того, можем ли мы дать этот ответ, требуется лишь потенциальная возможность его получения. Например, сказать будет ли истинным предложение «существует элементарная частица массой 10^{-9} кг» никто не может (пока, во всяком случае. А возможно, и никогда не сможет). Тем не менее оно либо является истинным, либо нет, спрашивать об этом имеет смысл.

Однако не каждое предложение является высказыванием. Например, фраза «яблоко сладкое» им не является, так как не понятно, о каком именно яблоке идёт речь, поэтому в такой формулировке говорить об истинности или ложности этой фразы бессмысленно. По той же причине не будут высказываниями и предложения «светило солнце» (когда, где?), «в комнате три окна» (в какой именно?) и т. д. Но фразы «существуют сладкие яблоки» или «в каждой комнате три окна» уже высказываниями являются (первое, очевидно, истинно, второе — ложно).

Фразы могут не быть высказываниями и в силу внутренних противоречий. Примером является так называемый парадокс лжеца.

Пример 10. *Фраза «настоящее предложение ложно» не является высказыванием.*

Если допустить, что она истинна, то сразу из её содержания получаем, что она должна быть ложной, противоречие.

Если наоборот, будем считать, что она ложна, то её содержание оказывается истинным, снова противоречие.

Логика высказываний занимается выяснением истинности тех или иных высказываний, связью между истинностью различных высказываний и т. д.

Высказывания делятся на элементарные и составные. Элементарными считаются высказывания, которые внутри себя уже не содержат других высказываний. Остальные считаются составными. Например, высказывание «сегодня утром было тепло и не было дождя» является составным: оно включает в себя элементарное высказывание «сегодня утром было тепло» и высказывание «сегодня утром не было дождя», которое снова является составным и содержит элементарное высказывание «сегодня утром был дождь».

Аппарат булевых функций может служить полезным инструментом при решении многих логических задач. Связующим инструментом являются формулы логики высказываний и их интерпретации.

Для формализации высказываний введём счётное множество пропозициональных переменных $\boldsymbol{V}$, которые будут служить для обозначения элементарных высказываний, а также множество

отличных от них логических символов

$$\boldsymbol{L} = \{«(», «)», «\neg», «\wedge», «\vee», «\to»\}.$$

Определение 28 (Формула логики высказываний)**.** *Формулой логики высказываний называется слово в алфавите* $\boldsymbol{V} \cup \boldsymbol{L}$*, определяемое по индукции:*

1) *База индукции. Каждая пропозициональная переменная* $x \in \boldsymbol{V}$ *является формулой.*
2) *Шаг индукции. Если* Φ_1 *и* Φ_2 *— формулы, то слова* $\neg\Phi_1$, $(\Phi_1 \wedge \Phi_2)$, $(\Phi_1 \vee \Phi_2)$, $(\Phi_1 \to \Phi_2)$ *тоже являются формулами.*

Как видим, формулы логики высказываний являются частным случаем инфиксных булевых формул.

Для придания формулам значения используют интерпретации, говорящие, какие из пропозициональных переменных истинны, а какие ложны. Истинность высказывания соответствует значению 1 пропозициональной переменной, а ложность — значению 0.

Определение 29 (Интерпретация и значение формулы)**.** *Интерпретация* I *в логике высказываний — это отображение множества пропозициональных переменных* $\boldsymbol{V}$ *в множество* $\{0, 1\}$*, то есть* $I : \boldsymbol{V} \to \{0, 1\}$*. Значение формулы* Φ *в интерпретации* I *определяется по индукции.*

1) *Если* $\Phi = x \in \boldsymbol{V}$*, то её значение* $I(x)$ *уже задано по определению интерпретации.*
2) *В остальных случаях значение формулы* Φ *определяется по следующей таблице:*

$I(\Phi_1)$	$I(\Phi_2)$	$I(\neg\Phi_1)$	$I(\Phi_1 \wedge \Phi_2)$	$I(\Phi_1 \vee \Phi_2)$	$I(\Phi_1 \to \Phi_2)$
0	0	1	0	0	1
0	1	1	0	1	1
1	0	0	0	1	0
1	1	0	1	1	1

Если $I(\Phi) = 1$, то говорят, что Φ *истинна в интерпретации* I*, а интерпретацию* I *называют моделью формулы* Φ*. Если* $I(\Phi) = 0$*, то* Φ *ложна в* I*.*

Если формула логики высказываний Φ содержит n пропозициональных переменных $x_1, \ldots, x_n$, то ей можно сопоставить булеву функцию f_Φ от n переменных, следующим образом:

$$f_\Phi(I(x_1), \ldots, I(x_n)) = I(\Phi)$$

для всех интерпретаций I.

Рассмотрим некоторые соответствия между логическими связками и конструкциями естественного языка.

Отрицание $\neg$ обычно передаётся в русском языке частицей «не» или оборотом «неверно, что» и подобным ему.

Конъюнкция $\wedge$ соответствует союзу «и» и его вариантам «а», «но», оборотам типа «также», «кроме того» и им подобным. Очень часто конъюнкция в естественной речи передаётся «немым» способом, то есть без использования каких-либо связующих слов: идёт перечень высказываний через запятую или в виде отдельных предложений. Одно исключение из этого правила описано в следующем абзаце.

Дизъюнкция $\vee$ передаётся союзами «или» и «либо». Также дизъюнкция подразумевается в случае «немого» перечисления высказываний, когда перед последним элементом этого перечня стоит союз «или» или «либо». Следует отличать использование одиночных союзов «или» и «либо» от повторяющихся. Повторяющиеся союзы имеют смысл «истинно в точности одно из высказываний», поэтому при помощи одной только дизъюнкции их смысл передать нельзя.

Импликация $A \to B$ может быть выражена многими способами. Наиболее распространёнными являются такие: «если A, то B», «из A следует (вытекает) B», «A влечёт B», «A, тогда B» и подобные им. Следует быть внимательным, так как некоторые обороты предполагают обратный порядок записи, например, «B, когда A». Со словом «только» части импликации меняются местами: «A, только если B», «B, только тогда A», «A, только когда B». Фразы типа «A тогда и

x_1	x_2	x_3	$(x_1$	$\to$	$(\neg$	x_2	$\vee$	$\neg$	$x_3))$
0	0	0	0	1	1	0	1	1	0
0	0	1	0	1	1	0	1	0	1
0	1	0	0	1	0	1	1	1	0
0	1	1	0	1	0	1	0	0	1
1	0	0	1	1	1	0	1	1	0
1	0	1	1	1	1	0	1	0	1
1	1	0	1	1	0	1	1	1	0
1	1	1	1	0	0	1	0	0	1

Рис. 6: Функция f_{Φ_1}.

только тогда, когда B», «A, если и только если B» означают наличие импликаций в обе стороны: $(A \to B) \wedge (B \to A)$.

Рассмотрим следующую задачу.

Пример 11. *Пусть известно, что в дорожном происшествии участвовали три автомобиля с водителями A, B и C. Свидетели происшествия дали следующие показания:*

1) *первый свидетель: если A виновен, то B или C не виновен;*
2) *второй свидетель: если C не виновен, то виновен или A, или B;*
3) *третий свидетель: в столкновении виновны не менее двух водителей.*

Можно ли на основании этих показаний сделать вывод, что C является виновником происшествия? Можно ли определить, есть ли ещё виновники и кто они?

Для ответа на эти вопросы введём три пропозициональные переменные, соответствующие следующим высказываниям: x_1: «виновен A», x_2: «виновен B» и x_3: «виновен C». Тогда показания первого свидетеля описываются формулой

$$\Phi_1 = (x_1 \to (\neg x_2 \vee \neg x_3)).$$

Задаваемая этой формулой функция f_{Φ_1} представлена на рис. 6.

Показаниям второго свидетеля соответствует формула

$$\Phi_2 = (\neg x_3 \to ((x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2))),$$

функция которой f_{Φ_2} задана таблицей на рис. 7 на следующей странице. Следует обратить внимание на имеющуюся здесь конструкцию «или ...,

x_1	x_2	x_3	$(\neg$	x_3	$\to$	$((x_1$	$\vee$	$x_2)$	$\wedge$	$\neg$	$(x_1$	$\wedge$	$x_2)))$
0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	1	0	1	1	0	0	0	0	1	0	0	0
0	1	0	1	0	1	0	1	1	1	1	0	0	1
0	1	1	0	1	1	0	1	1	1	1	0	0	1
1	0	0	1	0	1	1	1	0	1	1	1	0	0
1	0	1	0	1	1	1	1	0	1	1	1	0	0
1	1	0	1	0	0	1	1	1	0	0	1	1	1
1	1	1	0	1	1	1	1	1	0	0	1	1	1

Рис. 7: Функция f_{Φ_2}.

x_1	x_2	x_3	$((x_1$	$\wedge$	$x_2)$	$\vee$	$((x_1$	$\wedge$	$x_3)$	$\vee$	$(x_2$	$\wedge$	$x_3)))$
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1	0	0	0	1
0	1	0	0	0	1	0	0	0	0	0	1	0	0
0	1	1	0	0	1	1	0	0	1	1	1	1	1
1	0	0	1	0	0	0	1	0	0	0	0	0	0
1	0	1	1	0	0	1	1	1	1	1	0	0	1
1	1	0	1	1	1	1	1	0	0	0	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Рис. 8: Функция f_{Φ_3}.

или . . . »: в формуле записано, что одна из переменных x_1 или x_2 истинна, но не обе сразу.

Показаниям третьего свидетеля соответствует формула

$$\Phi_3 = ((x_1 \wedge x_2) \vee ((x_1 \wedge x_3) \vee (x_2 \wedge x_3)))$$

с функцией f_{Φ_3}, представленной в таблице на рис. 8.

Показаниям всех трёх свидетелей соответствует конъюнкция указанных формул $\Psi = (\Phi_1 \wedge (\Phi_2 \wedge \Phi_3))$. Составим таблицу значений для заданной этой формулой функции f_Ψ (рис. 9 на следующей странице).

Из этой таблицы следует, что $f_\Psi(x_1, x_2, x_3) = 1$ на двух наборах: $(0, 1, 1)$ и $(1, 0, 1)$ (строки с этими наборами выделены зелёным цветом). Поскольку в обоих случаях $x_3 = 1$, можно сделать вывод, что C является одним из виновников происшествия. В любом случае второй виновник есть, но одно-

x_1	x_2	x_3	$(\Phi_1$	$\wedge$	$(\Phi_2$	$\wedge$	$\Phi_3))$
0	0	0	1	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	1	0	1	0	0
1	0	1	1	1	1	1	1
1	1	0	1	0	0	0	1
1	1	1	0	0	1	1	1

Рис. 9: Функция f_Ψ.

значно определить его полученная от свидетелей информация не позволяет, так как в одном случае им является A, а в другом — B.

Важную роль в логике играют понятия тождественно истинной, тождественно ложной и выполнимой формулы.

Определение 30 (Тождественная истинность и ложность, выполнимость)**.** *Формула* Φ *называется тождественно истинной, если она истинна при всех значениях входящих в неё переменных, то есть функция* f_Φ *тождественно равна* 1.

Формула Φ *называется тождественно ложной, если она ложна при всех значениях входящих в неё переменных, то есть функция* f_Φ *тождественно равна* 0.

Формула Φ *называется выполнимой, если существует такой набор значений переменных, на котором она истинна, то есть функция* f_Φ *равна* 1 *хоть на одном наборе аргументов.*

Как проверить тождественную истинность (тождественную ложность, выполнимость) формулы Ф? На первый взгляд кажется, что ответ прост — построим по Ф таблицу для функции f_Φ, и, если в столбце значений стоят только единицы, то заключаем, что Ф тождественно истинна, если все нули — тождественно ложна, если есть хоть одна единица, то Ф выполнима. К сожалению, этот способ пригоден только для формул с небольшим числом переменных. Уже для нескольких десятков и сотен переменных он не годится из-за большо-

го размера получающейся таблицы, например, для 100 переменных таблица будет содержать $2^{100} \approx 10^{30}$ строк, даже перебрать их все — задача непосильная. Для самых мощных современных суперкомпьютеров потребуется порядка 10^{12} с ≈ 30000 лет непрерывной работы, и это без учёта самого времени вычисления значения формулы для каждой строки.

В математической логике построены аксиоматические системы, позволяющие формализовать человеческие рассуждения о выводимости одних тождественно истинных формул из других (см., например, [17]). В некоторых случаях они позволяют доказать тождественную истинность достаточно длинных формул, имеющих регулярную структуру. Но в общем случае и они практически не применимы для произвольных формул с большим числом переменных.

В теории сложности алгоритмов имеется ряд результатов (они выходят за рамки нашего курса), которые свидетельствуют о том, что эффективных алгоритмов для проверки выполнимости или тождественной истинности произвольной формулы, скорее всего, не существует. Вместе с тем для некоторых подклассов формул эти задачи решаются достаточно эффективно. Один такой подкласс — хорновские формулы — будет рассмотрен далее в главе 6.

Задачи

52. Какие подмножества вершин $\mathbb{B}^4$ соответствуют следующим булевым функциям:

(а) $f_1(x_1, x_2, x_3, x_4) = 1 \Leftrightarrow x_1 = 0$;

(б) $f_2(x_1, x_2, x_3, x_4) = 1 \Leftrightarrow x_4 = 1$;

(в) $f_3(x_1, x_2, x_3, x_4) = 1 \Leftrightarrow x_1 + x_2 \geqslant x_3 + x_4$;

(г) $f_4(x_1, x_2, x_3, x_4) = 1 \Leftrightarrow x_1 x_2 = 0$ или $x_3 x_4 = 1$. ▼

53. Построить таблицы значений для следующих булевых функций:

(а) $f_1(x_1, x_2, x_3) = 1 \Leftrightarrow x_1 + x_3 \geqslant x_2$;

(б) $f_2(x_1, x_2, x_3) = 1 \Leftrightarrow$ сумма $(x_1 + x_2 + x_3)$ чётна;

(в) $f_3(x_1, x_2, x_3) = 0 \Leftrightarrow (x_1 = x_2$ или $x_1 = x_3)$;

(г) $f_4(x_1, x_2, x_3) = \begin{cases} x_2, & \text{если } x_1 = 1; \\ x_3, & \text{иначе.} \end{cases}$ ▼

54. Построить таблицы для функций, задаваемых следующими формулами:

(а) $\Psi_1 = ((x_1 \to \neg x_3) \vee (x_2 \oplus x_3))$;

(б) $\Psi_2 = (\neg(x_1 \uparrow x_2) \leftrightarrow (\neg x_1 \wedge x_2))$;

(в) $\Psi_3 = ((x_2 \oplus \neg x_3) \wedge ((x_1 \vee x_2) \to (x_1 \leftrightarrow \neg x_3)))$.

55. Назовём два набора $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{B}^n$ и $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{B}^n$ *соседними*, если они находятся в соседних строках таблицы для функции от n переменных, то есть представляют двоичные записи чисел i_α и i_β, для которых $|i_\alpha - i_\beta| = 1$.

Найти число функций в $\mathcal{P}_n$, которые на каждой паре соседних наборов принимают

(а) одинаковые значения;

(б) разные значения.

56. Назовём два набора $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{B}^n$ и $\beta = (\beta_1, \ldots, \beta_n) \in \mathbb{B}^n$ *противоположными*, если $\alpha_i + \beta_i = 1$ для всякого i (иначе говоря, $\alpha_i = 1$ в том и только том случае, когда $\beta_i = 0$).

Найти число функций в $\mathcal{P}_n$, которые на каждой паре противоположных наборов принимают разные значения.

57. Пусть $n = 2k$. Назовём набор $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{B}^n$ *парным*, если $\alpha_i = \alpha_{k+i}$ для всех $i = 1, \ldots, k$, то есть $\alpha = \alpha'\alpha'$ для некоторого набора α' размера k. Найти число функций в $\mathcal{P}_n$, которые на всех парных наборах принимают одинаковое значение.

58. Администратор базы данных обнаружил, что одна или несколько из трёх записей его базы A, B и C ошибочна. Он установил, что

(а) если запись B корректна, то A ошибочна;

(б) хотя бы одна запись из пары B, C корректна и хотя бы одна запись из пары A, C корректна;

(в) если A ошибочна, то хотя одна из записей B или C корректна (но не обе вместе).

Описать знания администратора в виде формулы логики высказываний. Может ли он сделать вывод, что запись B ошибочна? Можно ли достоверно утверждать, что ошибочная запись единственна?

59. Программист Пётр использовал в своей программе на языке C три целочисленные переменные x, y, z. В определённом месте программы он поместил условный оператор:

```
if(x * y >= 0 || x * z >= 0) x = 1; else x = 2;
```

Проанализировав свою программу, Пётр установил, что перед выполнением этого оператора выполнены следующие условия:

(а) если $z < 0$, то $x < 0$ или $y \geqslant 0$;

(б) $x \geqslant 0$ или $y < 0$;

(в) если $y < 0$, то хотя бы одна из переменных x или z отрицательна, но не обе вместе.

Описать знания Петра в виде формулы логики высказываний. Может ли он оптимизировать программу, заменив указанный условный оператор на присваивание `x = 1;` или на присваивание `x = 2;`? Если «да», то на какое? ▼

60. Комитет состоит из пяти членов. Решения принимаются большинством голосов, однако, если председатель голосует «против», то решение не принимается. Построить формулу логики высказываний, зависящую от пяти пропозициональных переменных x_1, x_2, x_3, x_4, y (x_i означает, что i-й член комитета голосует «за», y означает, что председатель голосует «за»), значение которой равно 1 тогда и только тогда, когда в результате голосования решение принимается. ▼

61. В каждом из следующих текстов выделить элементарные высказывания, обозначить их пропозициональными переменными и записать формулу логики высказываний, отражающую содержание текста.

(а) Миша решил сделать небольшой ремонт в квартире и поклеить новые обои или покрасить пол. Вместе с обоями нужно будет обновить ламинат, сменить двери и стеклопакет в окне. После покраски пола нужно установить натяжной потолок, новую люстру и мебель. Если обновить ламинат, то тоже придётся покупать новую мебель. Старые обои нельзя оставлять со старой мебелью, а старую мебель — со старой люстрой.

(б) Бабушка сделала следующий прогноз. Если будет мороз, то будет солнечно или при пасмурной погоде пойдёт снег, а если мороза не будет, то будет солнечно или при пасмурной погоде пойдёт дождь. В случае дождя или снега будет сыро, а в случае дождя — ещё и грязно. Если будет солнечно, то не будет ни сырости, ни грязи.

(в) Когда Вова дома, то его кошка ест или спит, а если Вовы нет, то она ещё и играет. Если кошка ест, то нужно купить корм. Если кошка играет, то придётся покупать новую мебель или новые обои. За кормом нужно идти в магазин «Всё для кошки», а за мебелью или обоями — в магазин «Всё для дома». В любой из магазинов придётся ехать на такси. Вова не поедет на такси в магазин «Всё для дома». ▼

Глава 4

Эквивалентность формул и нормальные формы

Краткое содержание: логическое следование и эквивалентность формул, основные эквивалентности, эквивалентные преобразования формул, принцип замены, дизъюнктивные и конъюнктивные нормальные формы (ДНФ и КНФ), совершенные ДНФ и КНФ, сокращённые ДНФ и их построение методом Блейка, многочлены Жегалкина и их построение с помощью эквивалентных преобразований и методом неопределённых коэффициентов.

Ключевые слова: эквивалентные формулы, законы ассоциативности, коммутативности и дистрибутивности, закон двойного отрицания, законы де Моргана, законы идемпотентности, законы упрощения, законы поглощения, элементарная конъюнкция, элементарная дизъюнкция, дизъюнктивная нормальная форма (ДНФ), конъюнктивная нормальная форма (КНФ), совершенные ДНФ и КНФ, импликант, минимальный импликант, сокращённая ДНФ, многочлен Жегалкина.

§ 4.1. Основные эквивалентности

Одними из важнейших соотношений между формулами являются их эквивалентность и следование одной формулы из другой.

Определение 31 (Следование и эквивалентность)**.** *Формула* Φ *с л е д у е т из формулы* Ψ *(обозначается* $\Psi \Rightarrow \Phi$*), если* Φ *истинна в каждой интерпретации, в которой истинна* Ψ*.*

Формулы Φ *и* Ψ *э к в и в а л е н т н ы, если каждая из них следует из другой. Эквивалентность формул обозначается с помощью* $\Phi \equiv \Psi$*.*

Легко заметить, что формулы эквивалентны тогда и только тогда, когда задаваемые ими булевы функции равны. Другой критерий, который сразу получается из определений следования и импликации такой.

Следствие 17. *Из* Φ *следует* Ψ *тогда и только тогда, когда формула* $\Phi \to \Psi$ *тождественно истинна.*

Таким образом, эквивалентные формулы являются различными заданиями одной и той же булевой функции. Ниже мы приводим ряд пар эквивалентных формул (тождеств), отражающих существенные свойства логических операций и важные соотношения между различными операциями. Они часто позволяют находить для булевых функций по одним задающим их формулам более простые формулы. Большинство из приводимых тождеств имеют собственные имена. Часто их называют з а к о н а м и л о г и к и.

Далее с помощью Φ, Ψ и Θ обозначены произвольные формулы.

Пусть $\circ$ — это одна из связок $\wedge, \vee, \oplus$. Тогда выполнены следующие две эквивалентности (законы ассоциативности и коммутативности).

1) А с с о ц и а т и в н о с т ь:

$$((\Phi \circ \Psi) \circ \Theta) \equiv (\Phi \circ (\Psi \circ \Theta)).$$

2) К о м м у т а т и в н о с т ь:

$$(\Phi \circ \Psi) \equiv (\Psi \circ \Phi).$$

Следующие две группы эквивалентностей позволяют выносить одни связки из-под других.

3) Дистрибутивность:

$$((\Phi \vee \Psi) \wedge \Theta) \equiv ((\Phi \wedge \Theta) \vee (\Psi \wedge \Theta));$$
$$((\Phi \wedge \Psi) \vee \Theta) \equiv ((\Phi \vee \Theta) \wedge (\Psi \vee \Theta));$$
$$((\Phi \oplus \Psi) \wedge \Theta) \equiv ((\Phi \wedge \Theta) \oplus (\Psi \wedge \Theta)).$$

4) Законы де Моргана (внесение отрицания внутрь скобок):

$$\neg(\Phi \vee \Psi) \equiv (\neg\Phi \wedge \neg\Psi); \qquad \neg(\Phi \wedge \Psi) \equiv (\neg\Phi \vee \neg\Psi).$$

Следующие эквивалентности используется для упрощения формул.

5) Закон двойного отрицания:

$$\neg\neg\Phi \equiv \Phi.$$

6) Идемпотентность:

$$(\Phi \wedge \Phi) \equiv \Phi; \quad (\Phi \vee \Phi) \equiv \Phi.$$

7) Законы для констант:

$$(\Phi \wedge \neg\Phi) \equiv 0; \qquad (\Phi \vee \neg\Phi) \equiv 1;$$
$$(\Phi \wedge 0) \equiv 0; \qquad (\Phi \vee 0) \equiv \Phi;$$
$$(\Phi \wedge 1) \equiv \Phi; \qquad (\Phi \vee 1) \equiv 1.$$

Некоторые из этих законов имеют свои собственные названия: $(\Phi \wedge \neg\Phi) \equiv 0$ — это закон противоречия, $(\Phi \vee \neg\Phi) \equiv 1$ — это закон исключённого третьего. Эквивалентности в двух последних строках иногда называют 0-1 законами.

Следующие две эквивалентности позволяют выразить импликацию и сложение по модулю 2 через дизъюнкцию, конъюкцию и отрицание:

8) $(\Phi \to \Psi) \equiv (\neg\Phi \vee \Psi)$;

9) $(\Phi \oplus \Psi) \equiv ((\Phi \wedge \neg\Psi) \vee (\neg\Phi \wedge \Psi)) \equiv ((\Phi \vee \Psi) \wedge (\neg\Phi \vee \neg\Psi))$.

Проверку правильности этих эквивалентностей оставляем читателям (см. задачу 63 на стр. 102).

§ 4.2. Эквивалентные преобразования формул

Прежде чем продолжать, сформулируем некоторые общепринятые соглашения, которые позволят нам упрощать запись формул.

1) Законы ассоциативности для связок $\wedge$, $\vee$, $\oplus$ говорят нам, что значение формулы типа $x_1 \wedge \cdots \wedge x_n$ не будет зависеть от способа расстановки скобок. Поэтому мы в дальнейшем будем записывать формулы видов

$$x_1 \wedge \cdots \wedge x_n; \quad x_1 \vee \cdots \vee x_n; \quad x_1 \oplus \cdots \oplus x_n$$

опуская скобки, поскольку на результат они не влияют.

2) Опускаются самые внешние скобки вокруг всей формулы.

3) Для различных связок общепринятым является следующий приоритет в порядке убывания: $\neg$, $\wedge$, $\vee$, $\oplus$, $\to$. Поэтому в ряде случаев мы будем опускать скобки, например, в формуле

$$a \wedge b \to \neg a \vee \neg c$$

скобки расставляются так:

$$((a \wedge b) \to (\neg a \vee \neg c)).$$

4) Для операций одинакового приоритета используется ассоциативность: $\wedge$, $\vee$ и $\oplus$ считаются левоассоциативными, то есть для них скобки расставляются слева направо (хотя, как мы уже отметили в пункте 1, это не существенно), а $\neg$ и $\to$ — правоассоциативными, скобки для них расставляются справа налево. Например, в формуле

$$a \vee b \vee c \to \neg\neg a \to a \wedge c \wedge \neg\neg b$$

скобки следует расставить так:

$$(((a \vee b) \vee c) \to (\neg\neg a \to ((a \wedge c) \wedge \neg\neg b))).$$

Из определения эквивалентности формул непосредственно следует принцип замены эквивалентными формулами. Дадим сначала определение.

Определение 32 (Замена пропозициональной переменной)**.** *Если* Φ *и* Ψ *— формулы, а* x *— пропозициональная переменная, то* $(\Phi)^x_\Psi$ *— это формула, полученная з а м е н о й в* Φ *всех вхождений переменной* x *на формулу* Ψ*.*

Например, в результате замены $(x \wedge y \to \neg x)^x_{(x\to z)}$ мы получим $((x \to z) \wedge y \to \neg(x \to z))$.

Теорема 18 (Принцип замены)**.** *Если* $\Phi \equiv \Psi$*, то* $(\Theta)^x_\Phi \equiv (\Theta)^x_\Psi$*, для любых формул* Φ*,* Ψ*,* Θ *и пропозициональной переменной* x*.*

Доказательство. Индукция по построению формулы Θ.

Б а з и с и н д у к ц и и. Если формула Θ является константой 0 или 1, а также если Θ — это переменная, отличная от x, то, очевидно, $(\Theta)^x_\Phi = (\Theta)^x_\Psi = \Theta$.

Если $\Theta = x$, то $(\Theta)^x_\Phi = \Phi \equiv \Psi = (\Theta)^x_\Psi$.

И н д у к ц и о н н ы й ш а г. Нужно рассмотреть все варианты построения формулы Θ. Пусть, например, $\Theta = (\Theta_1 \wedge \Theta_2)$. Тогда

$$(\Theta)^x_\Phi = (\Theta_1 \wedge \Theta_2)^x_\Phi = ((\Theta_1)^x_\Phi \wedge (\Theta_2)^x_\Phi);$$
$$(\Theta)^x_\Psi = (\Theta_1 \wedge \Theta_2)^x_\Psi = ((\Theta_1)^x_\Psi \wedge (\Theta_2)^x_\Psi).$$

По индукционному предположению можно считать $(\Theta_1)^x_\Phi \equiv (\Theta_1)^x_\Psi$ и $(\Theta_1)^x_\Phi \equiv (\Theta_2)^x_\Psi$. Но тогда

$$((\Theta_1)^x_\Phi \wedge (\Theta_2)^x_\Phi) \equiv ((\Theta_1)^x_\Psi \wedge (\Theta_2)^x_\Psi),$$

что и требуется.

Доказательство для остальных связок аналогично. □

Рассмотрим пример применения этой теоремы.

Пример 12. *Доказать эквивалентность* $(x \vee y) \to z \equiv (x \to z) \wedge (y \to z)$.

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

Здесь мы по очереди воспользовались эквивалентностями: 8) для замены импликации на отрицание и дизъюнкцию, законом де Моргана, дистрибутивностью, 8) для обратной замены.

Докажем теперь эквивалентность $(x \wedge y) \to z \equiv (x \to z) \vee (y \to z)$.

$$\begin{aligned}(x \wedge y) \to z \equiv \neg(x \wedge y) \vee z \equiv (\neg x \vee \neg y) \vee z \equiv (\neg x \vee \neg y) \vee (z \vee z) \equiv \\ \equiv \neg x \vee (\neg y \vee (z \vee z)) \equiv \neg x \vee ((\neg y \vee z) \vee z) \equiv \neg x \vee (z \vee (\neg y \vee z)) \equiv \\ \equiv (\neg x \vee z) \vee (\neg y \vee z) \equiv (x \to z) \vee (y \to z)\end{aligned}$$

Здесь по очереди применены: 8) для замены импликации на отрицание и дизъюнкцию, закон де Моргана, идемпотентность, два раза ассоциативность, коммутативность, ещё раз ассоциативность, 8) для обратной замены.

Справедлив и второй принцип замены.

Предложение 19. *Если в двух эквивалентных формулах* Φ *и* Ψ *заменить пропозициональную переменную* x *на формулу* Θ*, то полученные формулы снова будут эквивалентными: если* $\Phi \equiv \Psi$*, то* $(\Phi)^x_\Theta \equiv (\Psi)^x_\Theta$.

Доказательство. См. задачу 68 на стр. 102. □

Выведем ещё несколько важных логических тождеств, позволяющих проводить упрощения сложных формул. Их называют законами поглощения.

Предложение 20 (Законы поглощения)**.** *Справедливы следующие эквивалентности:*

1) $\Phi \vee (\Phi \wedge \Psi) \equiv \Phi$;
2) $(\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Psi) \equiv \Psi$;
3) $(\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Theta) \vee (\Psi \wedge \Theta) \equiv (\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Theta)$.

Доказательство. Первый:

$$\Phi \vee (\Phi \wedge \Psi) \equiv (\Phi \wedge 1) \vee (\Phi \wedge \Psi) \equiv \Phi \wedge (1 \vee \Psi) \equiv \Phi \wedge 1 \equiv \Phi.$$

Второй:

$$(\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Psi) \equiv (\Phi \vee \neg\Phi) \wedge \Psi \equiv 1 \wedge \Psi \equiv \Psi.$$

Третий:

$$(\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Theta) \vee (\Psi \wedge \Theta) \equiv (\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Theta) \vee (\Psi \wedge \Theta) \wedge 1 \equiv$$
$$\equiv (\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Theta) \vee (\Psi \wedge \Theta) \wedge (\Phi \vee \neg\Phi) \equiv$$
$$\equiv (\Phi\wedge\Psi)\vee(\neg\Phi\wedge\Theta)\vee(\Psi\wedge\Theta\wedge\Phi)\vee(\Psi\wedge\Theta\wedge\neg\Phi) \equiv (\Phi\wedge\Psi)\vee(\neg\Phi\wedge\Theta).$$

Последняя эквивалентность — это пункт 1, применённый к первой и третей, а также — второй и четвёртой конъюнкции. □

§ 4.3. Дизъюнктивные и конъюнктивные нормальные формы

В этом разделе мы интересуемся представлением произвольной булевой функции посредством формул специального вида, использующих только операции $\wedge, \vee$ и $\neg$.

Введём следующее обозначение.

Определение 33. *Пусть* Φ *— любая формула,* $\sigma \in \{0,1\}$*. Тогда*

$$\Phi^\sigma = \begin{cases} \Phi, & \text{если } \sigma = 1, \\ \neg\Phi, & \text{если } \sigma = 0. \end{cases}$$

Как следствие легко получить следующие эквивалентности:

$$1^\sigma \equiv \sigma; \quad 0^\sigma \equiv \neg\sigma; \quad \sigma^\sigma \equiv 1; \quad \sigma^\tau \equiv 0, \tag{8}$$

для любых $\sigma \neq \tau$, а для любой интерпретации I и пропозициональной переменной x получаем $I(x^{I(x)}) = 1$.

Определение 34 (Элементарные конъюнкция и дизъюнкция)**.** *Формула* Φ *называется элементарной конъюнкцией, если она имеет вид*

$$\Phi = x_1^{\sigma_1} \wedge \cdots \wedge x_n^{\sigma_n},$$

а элементарной дизъюнкцией — в случае

$$\Phi = x_1^{\sigma_1} \vee \cdots \vee x_n^{\sigma_n},$$

где x_i *— пропозициональные переменные, а* $\sigma_i \in \{0,1\}$ *для всех* $i = 1, \ldots, n$*.*

Учитывая законы идемпотентности, можно считать, что в элементарной конъюнкции (дизъюнкции) все элементы $x_i^{\sigma_i}$ попарно различны.

Определение 35 (Дизъюнктивная нормальная форма)**.** *Формула* Φ *называется дизъюнктивной нормальной формой (ДНФ), если она является дизъюнкцией элементарных конъюнкций, то есть*

$$\Phi = K_1 \vee K_2 \vee \ldots \vee K_r,$$

где формулы K_j, $j = 1, \ldots, r$, *— это элементарные конъюнкциии.*

ДНФ Φ *называется совершенной, если в каждую из её элементарных конъюнкций* K_j *входят все переменные* Φ *в точности по одному разу, а сами эти конъюнкции попарно различны с точностью до перестановки их элементов.*

Например, формула $(x \wedge y) \vee \neg x$ является ДНФ, но не является совершенной ДНФ, так как вторая элементарная конъюнкция не содержит y.

Аналогично определяется двойственное понятие.

Определение 36 (Конъюнктивная нормальная форма)**.** *Формула* Φ *называется конъюнктивной нормальной формой (КНФ), если она является конъюнкцией элементарных дизъюнкций, то есть*

$$\Phi = D_1 \wedge D_2 \wedge \ldots \wedge D_r,$$

где формулы D_j, $j = 1, \ldots, r$ *— это элементарные дизъюнкции.*

КНФ Φ *называется совершенной, если в каждую из её дизъюнкций* D_j *все переменные* Φ *входят в точности по одному разу, а сами эти дизъюнкции попарно различны с точностью до перестановки их элементов.*

Рассмотрим произвольную булеву функцию $f(x_1, \ldots, x_n)$. Определим две формулы:

$$\mathcal{D}_f = \bigvee_{f(\sigma_1,\ldots,\sigma_n)=1} \bigwedge_{i=1}^{n} x_i^{\sigma_i}; \qquad \mathcal{C}_f = \bigwedge_{f(\sigma_1,\ldots,\sigma_n)=0} \bigvee_{i=1}^{n} x_i^{\neg\sigma_i}.$$

Теорема 21. *Пусть f — булева функция.*

1) *Если f не равна тождественно нулю, то формула $\mathcal{D}_f$ — это совершенная ДНФ, задающая функцию f.*

2) *Если f не равна тождественно единице, то формула $\mathcal{C}_f$ — это совершенная КНФ, задающая функцию f.*

Доказательство. Докажем теорему для ДНФ $\mathcal{D}_f$, а доказательство для КНФ $\mathcal{C}_f$ оставим в виде задачи 69 на стр. 102.

Пусть $f(\sigma_1, \ldots, \sigma_n) = 1$, тогда $\mathcal{D}_f$ содержит элементарную конъюнкцию $K = \bigwedge\limits_{i=1}^{n} x_i^{\sigma_i}$. При $x_i = \sigma_i$ для всех $i = 1, \ldots, n$, согласно (8), получаем, что все $x_i^{\sigma_i}$ равны 1, значит, K равно 1 и $\mathcal{D}_f$ равно 1.

Пусть теперь $\mathcal{D}_f$ равно 1 для значений $(\sigma_1, \ldots, \sigma_n)$ переменных $(x_1, \ldots, x_n)$. ДНФ $\mathcal{D}_f$ равна 1, только если некоторая её элементарная конъюнкция $K = \bigwedge\limits_{i=1}^{n} x_i^{\tau_i}$ равна 1. Если $\sigma_i \neq \tau_i$ для некоторого i, то, согласно (8), получаем, что $x_i^{\tau_i} \equiv 0$ при $x_i = \sigma_i$, что невозможно из-за истинности всех элементов K. Значит, $\sigma_i = \tau_i$ для всех i и мы имеем $K = \bigwedge\limits_{i=1}^{n} x_i^{\sigma_i}$. Но такая конъюнкция входит в $\mathcal{D}$, только если $f(\sigma_1, \ldots, \sigma_n) = 1$, что и требуется. □

Следствие 22. *Каждая булева функция может быть задана формулой, содержащей переменные и функции конъюнкции, дизъюнкции и отрицания.*

Приведённые выше формулы для $\mathcal{D}_f$ и $\mathcal{C}_f$ позволяют эффективно строить совершенные ДНФ и КНФ по табличному представлению функции f. Можно ли получить такие специальные представления по произвольной формуле, задающей f, не выписывая её полной таблицы? Приводимая ниже процедура позволяет это сделать, используя основные эквивалентности формул.

Пусть дана формула Φ, включающая функции $\neg, \wedge, \vee, \to$ и $\oplus$ (не обязательно все).

Сначала построим для Φ некоторую ДНФ $\mathcal{D}$ (не обязательно пока совершенную). Будем это делать индукцией по длине формулы Φ.

1) Если $\Phi = x$ — пропозициональная переменная, то Φ уже является ДНФ и $\mathcal{D} = x$.

2) Пусть $\Phi = \Psi_1 \vee \Psi_2$. Тогда следует построить эквивалентные ДНФ $\mathcal{D}_1$ и $\mathcal{D}_2$ для Ψ_1 и Ψ_2 соответственно, после чего получаем $\Phi = \Psi_1 \vee \Psi_2 \equiv \mathcal{D}_1 \vee \mathcal{D}_2 = \mathcal{D}$ — ДНФ.

3) Пусть $\Phi = \Psi_1 \wedge \Psi_2$. Тогда следует построить эквивалентные ДНФ
$$\mathcal{D}_1 = \bigvee_{i=1}^{r} K_{1i}; \quad \mathcal{D}_2 = \bigvee_{j=1}^{s} K_{2j}$$
для Ψ_1 и Ψ_2 соответственно. Затем используем закон дистрибутивности:
$$\Phi = \Psi_1 \wedge \Psi_2 \equiv \bigvee_{i=1}^{r} K_{1i} \wedge \bigvee_{j=1}^{s} K_{2j} \equiv \bigvee_{i=1}^{r} \bigvee_{j=1}^{s} (K_{1i} \wedge K_{2j}) = \mathcal{D},$$
что является ДНФ.

4) Если $\Phi = \Psi_1 \to \Psi_2$, то воспользуемся эквивалентностью
$$\Phi = \Psi_1 \to \Psi_2 \equiv \neg\Psi_1 \vee \Psi_2,$$
построим ДНФ $\mathcal{D}_1 \equiv \neg\Psi_1$ и ДНФ $\mathcal{D}_2 \equiv \Psi_2$ (обе формулы короче Φ, для них ДНФ уже построены). Тогда $\mathcal{D} = \mathcal{D}_1 \vee \mathcal{D}_2$.

5) Если $\Phi = \Psi_1 \oplus \Psi_2$, то воспользуемся эквивалентностью
$$\Phi = \Psi_1 \oplus \Psi_2 \equiv (\Psi_1 \wedge \neg\Psi_2) \vee (\neg\Psi_1 \wedge \Psi_2).$$
Построим ДНФ $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$ и $\mathcal{D}_4$ для Ψ_1, $\neg\Psi_2$, $\neg\Psi_1$ и Ψ_2 соответственно. Далее как в пункте 3) построим ДНФ
$$\mathcal{D}' \equiv \mathcal{D}_1 \wedge \mathcal{D}_2 \equiv \Psi_1 \wedge \neg\Psi_2$$
и ДНФ
$$\mathcal{D}'' \equiv \mathcal{D}_3 \wedge \mathcal{D}_4 \equiv \neg\Psi_1 \wedge \Psi_2.$$
Тогда $\mathcal{D} = \mathcal{D}' \vee \mathcal{D}''$.

Для отрицания следует «заглянуть» на один шаг глубже.

6) Если $\Phi = \neg x$, где x — пропозициональная переменная, то Φ уже является ДНФ и $\mathcal{D} = \neg x$.

7) Если $\Phi = \neg\neg\Psi$, то $\Phi \equiv \Psi$ и достаточно построить ДНФ $\mathcal{D}$ для формулы Ψ.

8) Пусть $\Phi = \neg(\Psi_1 \vee \Psi_2)$. Тогда

$$\Phi = \neg(\Psi_1 \vee \Psi_2) \equiv \neg\Psi_1 \wedge \neg\Psi_2.$$

В этом случае следует построить эквивалентные ДНФ $\mathcal{D}_1$ и $\mathcal{D}_2$ для $\neg\Psi_1$ и $\neg\Psi_2$ соответственно, а дальше действуем как в пункте 3).

9) Пусть $\Phi = \neg(\Psi_1 \wedge \Psi_2)$, то есть

$$\Phi = \neg(\Psi_1 \wedge \Psi_2) \equiv \neg\Psi_1 \vee \neg\Psi_2.$$

Тогда строим эквивалентные ДНФ $\mathcal{D}_1$ и $\mathcal{D}_2$ для $\neg\Psi_1$ и $\neg\Psi_2$ соответственно, а дальше действуем как в пункте 2).

10) При

$$\Phi = \neg(\Psi_1 \to \Psi_2) \equiv \Psi_1 \wedge \neg\Psi_2$$

нужно построить эквивалентные ДНФ $\mathcal{D}_1$ и $\mathcal{D}_2$ для Ψ_1 и $\neg\Psi_2$ соответственно, а дальше — как в пункте 3).

11) В последнем случае $\Phi = \neg(\Psi_1 \oplus \Psi_2)$ получаем

$$\Phi = \neg(\Psi_1 \oplus \Psi_2) \equiv \underbrace{(\Psi_1 \wedge \Psi_2)}_{(*)} \vee \underbrace{(\neg\Psi_1 \wedge \neg\Psi_2)}_{(**)}.$$

В пункте 3) показано, как построить ДНФ $\mathcal{D}'$ для $(*)$, а в пункте 8) — ДНФ $\mathcal{D}''$ для $(**)$. Тогда $\Phi \equiv \mathcal{D}' \vee \mathcal{D}''$.

Итак, с помощью пунктов 1)–11) мы сможем по любой формуле Φ построить некоторую эквивалентную ей ДНФ $\mathcal{D}$:

$$\Phi \equiv \mathcal{D} = K_1 \vee \ldots \vee K_m.$$

12) Чтобы завершить построение и получить эквивалентную совершенную ДНФ, построим для каждой элементарной конъюнкции K_i, $i = 1, \ldots, m$, эквивалентную ей совершенную ДНФ $D_i \equiv K_i$ (см. задачу 70 на стр. 102), заменим ею K_i:

$$\mathcal{D}' = D_1 \vee \ldots \vee D_m \equiv K_1 \vee \ldots \vee K_m \equiv \mathcal{D}$$

и устраним в $\mathcal{D}'$ повторения одинаковых конъюнкций.

Пример 13. *Пусть дана формула* $\Phi = ((\neg x \vee z) \to (y \to (x \oplus z)))$, *которую требуется преобразовать к совершенной ДНФ.*

Согласно пункту 4) получаем

$$\neg(\neg x \vee z) \vee (y \to (x \oplus z)),$$

применяем пункт 8) к первой скобке:

$$(\neg\neg x \wedge \neg z) \vee (y \to (x \oplus z))$$

и, наконец, пункт 7):

$$(x \wedge \neg z) \vee (y \to (x \oplus z)),$$

после чего слева получилась часть ДНФ. Далее используем 4) для второй скобки:

$$(x \wedge \neg z) \vee (\neg y \vee (x \oplus z)),$$

затем 5):

$$(x \wedge \neg z) \vee (\neg y \vee ((x \wedge \neg z) \vee (\neg x \wedge z))).$$

На этом этапе мы получили ДНФ:

$$(x \wedge \neg z) \vee \neg y \vee (x \wedge \neg z) \vee (\neg x \wedge z).$$

Удалим повторное вхождение первой конъюнкции, чтобы в дальнейшем не выполнять лишней работы, и получим ДНФ

$$\mathcal{D}' = (x \wedge \neg z) \vee \neg y \vee (\neg x \wedge z).$$

Эта ДНФ не является совершенной, так как в каждую из её трёх конъюнкций входят не все переменные. На последнем шаге построим для них эквивалентные совершенные ДНФ:

$$(x \wedge \neg z) \equiv (x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge \neg z),$$

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

Подставив эти формулы в Φ_1 и устранив повторения конъюнкций, получим совершенную ДНФ, эквивалентную исходной формуле Φ:

$$\mathcal{D} = (x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge \neg z) \vee (x \wedge \neg y \wedge z) \vee$$
$$\vee (\neg x \wedge \neg y \wedge z) \vee (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge z).$$

Мы видим, что ДНФ $\mathcal{D}'$ выглядит существенно проще, то есть является более короткой, чем совершенная ДНФ $\mathcal{D}$. Однако совершенные ДНФ и КНФ обладают важным свойством единственности, которое следует из их конструкции в теореме 21 на стр. 86.

Следствие 23. *Для каждой булевой функции от n переменных, не равной тождественно нулю, существует единственная с точностью до перестановки конъюнкций и переменных внутри конъюнкций совершенная ДНФ, задающая эту функцию.*

Это следствие позволяет предложить следующую процедуру для проверки эквивалентности формул Φ и Ψ.

1) Построить для Φ и Ψ эквивалентные совершенные ДНФ $\mathcal{D}_1$ и $\mathcal{D}_2$ соответственно, используя описанную выше процедуру.
2) Упорядочить (одним и тем же способом, например, в соответствии с номерами переменных) вхождения переменных в каждую конъюнкцию, а затем лексикографически упорядочить между собой конъюнкции, входящие в $\mathcal{D}_1$ и $\mathcal{D}_2$. Пусть в результате получатся совершенные ДНФ $\mathcal{D}'_1$ и $\mathcal{D}'_2$.
3) Если $\mathcal{D}'_1 = \mathcal{D}'_2$, то выдать ответ «Да», иначе — ответ «Нет».

Замечание 4. Аналогичную процедуру можно построить с использованием совершенных КНФ.

§ 4.4. Сокращённые ДНФ

Сокращённые ДНФ являются ещё одним способом однозначного представления булевых функций, которое во многих случаях может оказаться более простым, чем совершенные ДНФ.

Определение 37 (Импликант, минимальный импликант)**.** *Для формулы* Ф *импликантом называется выполнимая элементарная конъюнкция, из которой* Ф *следует.*

Импликант называется минимальным, если никакая его часть импликантом для Ф *не является.*

В русской литературе ещё встречаются названия допустимая элементарная конъюнкций и максимальная элементарная конъюнкция для импликанта и минимального импликанта соответственно.

Например, для формулы $a \to b \wedge \neg c$ минимальными импликантами будут $\neg a$ и $b \wedge \neg c$. Первый нельзя сократить очевидным образом, для второго нетрудно проверить, что b и $\neg c$ импликантами не являются.

Из определения получаем

Следствие 24. *Если ДНФ* $\mathcal{D}$ *эквивалентна формуле* Ф*, то* $\mathcal{D}$ *состоит из импликантов формулы* Ф*.*

Доказательство. В самом деле, если элементарная конъюнкция K входит в $\mathcal{D}$ и истинна, то истинна вся $\mathcal{D}$ и, следовательно, Ф. Значит, из K следует Ф. □

Определение 38 (Сокращённая ДНФ)**.** *Для формулы* Ф *сокращённой ДНФ называется дизъюнкция всех её минимальных импликантов.*

Также сокращённую ДНФ называют нормальной (или канонической) формой Блейка. Из этого определения сразу следует, что для любой формулы существует в точности одна сокращённая ДНФ (с точностью до перестановок элементарных конъюнкций и их элементов).

Сокращённую ДНФ для формулы Ф можно получить, используя процедуру, называемую методом Блейка.

1) Построить для Ф совершенную ДНФ $\mathcal{D}_1$.

2) Применять для $\mathcal{D}_1$, сколько возможно, закон поглощения

$$(x \wedge K_1) \vee (\neg x \wedge K_2) \equiv (x \wedge K_1) \vee (\neg x \wedge K_2) \vee (K_1 \wedge K_2)^*$$

слева направо при условии, что конъюнкция $(K_1 \wedge K_2)$ непротиворечива, то есть не содержит одновременно некоторую переменную и её отрицание. С помощью $(K_1 \wedge K_2)^*$ обозначен результат удаления повторов в конъюнкции $(K_1 \wedge K_2)$. Заметим, что на этом этапе число элементарных конъюнкций в ДНФ, вообще говоря, увеличивается.

3) К ДНФ $\mathcal{D}_2$, полученной на предыдущем шаге, применять, сколько возможно, закон поглощения слева направо

$$K_1 \vee (K_1 \wedge K_2) \equiv K_1.$$

4) Из ДНФ $\mathcal{D}_3$, полученной на предыдущем шаге, удалить повторные вхождения одинаковых элементарных конъюнкций.

Покажем, что в результате применения этого метода мы получим сокращённую ДНФ для формулы Φ, а также, что эта сокращённая ДНФ эквивалентна Φ (из определения 38 на предыдущей странице это непосредственно не следует).

Теорема 25. *В результате применения метода Блейка к формуле* Φ *через конечное число шагов будет получена эквивалентная ей сокращённая ДНФ* $\mathcal{D}$.

Доказательство. Пусть формула Φ содержит n переменных. Последовательно покажем, что после этапа i мы получаем формулу $\mathcal{D}_i$ эквивалентную Φ, которая, кроме того, будет обладать некоторыми дополнительными свойствами.

Очевидно, $\mathcal{D}_1 \equiv \Phi$ и является совершенной ДНФ для Φ.

Покажем, что в $\mathcal{D}_2$ содержатся все импликанты K для Φ и только они (с точностью до перестановки и повторов в элементарных конъюнкциях). Доказательство проведём обратной индукцией по числу k переменных в импликанте K.

Базис индукции. Пусть элементарная конъюнкция K содержит все n переменных: $k = n$. Так как K выполнима, то она не может содержать пары вида $x \wedge \neg x$ ни для какой переменной x. Значит, каждая переменная встречается в K в точности один раз.

Так как из K следует Ф и, следовательно, $\mathcal{D}_1$, то совершенная ДНФ $\mathcal{D}_1$ должна содержать K. Поскольку при выполнении шага 2) мы ничего не удаляем, то K останется и в $\mathcal{D}_2$.

Ш а г и н д у к ц и и. Пусть для некоторого k утверждение доказано для всех импликантов с $k+1$ переменными, $k+1 \leqslant n$. Докажем, что оно верно и для импликантов K с k переменными.

Так как $k+1 \leqslant n$, то $k < n$, следовательно, импликант K не содержит какую-то переменную x. Тогда формулы $K \wedge x$ и $K \wedge \neg x$ выполнимы. Поскольку из K следует Ф, то Ф следует и из $K \wedge x$ и $K \wedge \neg x$. Итак, обе элементарные конъюнкции являются импликантами для Ф, причём они содержат $k+1$ переменную. По индукционному предположению обе они принадлежат $\mathcal{D}_2$. Но тогда из них с помощью закона поглощения должно быть получено $(K \wedge K)^* = K$, то есть K принадлежит $\mathcal{D}_2$, так как в $\mathcal{D}_2$ всевозможные поглощения уже применены.

Чтобы показать, что $\mathcal{D}_2$ не содержит ничего, кроме импликантов Ф, достаточно заметить, что $\mathcal{D}_2 \equiv \mathcal{D}_1 \equiv \Phi$, поскольку выполнялись лишь эквивалентные преобразования. Тогда $\mathcal{D}_2$ — ДНФ для Ф и, как мы уже отмечали в следствии 24 на стр. 91, должна содержать только импликанты Ф.

Далее докажем, что $\mathcal{D}_3$ содержит все минимальные импликанты и только их. Если K — минимальный импликант для Ф, то не может существовать импликант K' для Ф такой, что $K = K' \wedge K''$ для какого-то K''. Значит, при выполнении поглощений на шаге 3) импликант K не исчезнет и останется в $\mathcal{D}_3$.

С другой стороны, если импликант K не минимальный, то найдётся импликант K' такой, что $K = K' \wedge K''$ для какого-то K''. Так как $\mathcal{D}_2$ содержит все импликанты, то он содержит и K'. Но тогда K пропадёт при выполнении поглощений на шаге 3) и не попадёт в $\mathcal{D}_3$.

Так как на шаге 3) выполняются только эквивалентные преобразования, то $\mathcal{D}_3 \equiv \mathcal{D}_2 \equiv \Phi$.

Итак, $\mathcal{D}_3$ состоит в точности из минимальным импликантов. Но тогда при удалении повторов в ней мы на шаге 4) получим в точности

сокращённую ДНФ $\mathcal{D}_4$ для Ф, а в силу законов идемпотентности $\mathcal{D}_4 \equiv \Phi$. □

Рассмотрим пример применения метода Блейка.

Пример 14. *Возьмём функцию f, значения которой при лексикографическом упорядочении аргументов образуют последовательность* (01110100).

Её совершенная ДНФ имеет вид

$$\mathcal{D}_1 = (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3).$$

После применения преобразований на втором этапе получим

$$\mathcal{D}_2 = (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee \\ \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3).$$

После поглощений на третьем этапе останется сокращённая ДНФ

$$\mathcal{D}_3 = (\neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3).$$

Заметим, что она не является самой короткой ДНФ для f, так как

$$\mathcal{D}_3 \equiv (\neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2).$$

Замечание 5 (Сокращённые КНФ)**.** Поскольку ДНФ и КНФ являются двойственными друг другу формами, то существует и аналог сокращённых ДНФ — сокращённые КНФ. Для этого можно ввести понятие антиимпликанта формулы Ф — не тождественно истинной элементарной дизъюнкции, которая следует из Ф, минимального антиимпликанта и сокращённой КНФ, как конъюнкции всех минимальных антиимпликантов. Для построения сокращённой КНФ можно применить тот же метод, только использовать двойственные законы поглощения.

§ 4.5. Многочлены Жегалкина

Многочлены Жегалкина являются ещё одним интересным подклассом формул, позволяющим однозначно представлять булевы функции. Они являются формулами над множеством булевых функций $\{0, 1, \wedge, \oplus\}$. Заметим, что значение формулы $x \wedge y$ равно произведению значений x и y. На этом основании при работе с многочленами Жегалкина вместо конъюнкции $\wedge$ пишут обычное умножение: xy вместо $x \wedge y$, $x(yz \oplus 1)$ вместо $x \wedge (y \wedge z \oplus 1)$ и т. д.

Определение 39 (Многочлен Жегалкина)**.** *Многочленами Жегалкина называются формулы вида $K_1 \oplus \cdots \oplus K_n$, где каждая K_i, $i = 1, \ldots, n$, является нулём, единицей или произведением какого-то числа переменных.*

Назовём многочлен Жегалкина приведённым, если он равен 0 *либо все его слагаемые не равны* 0*, не содержат повторяющихся множителей и сами попарно различны (с точностью до перестановки сомножителей).*

Например, приведёнными многочленами Жегалкина являются $x \oplus 1$ и $xy \oplus xyz \oplus z$.

Таким образом, каждый многочлен Жегалкина представляет сумму по модулю 2 положительных (монотонных) элементарных конъюнкций (то есть элементарных конъюнкций без отрицаний). Поскольку для $\oplus$ и $\wedge$ справедливы законы ассоциативности, мы будем при записи многочлена Жегалкина опускать скобки.

Легко проверить справедливость следующих эквивалентностей:

$$\neg x \equiv (x \oplus 1), \tag{9}$$

$$(x_1 \wedge x_2) \equiv (x_1 x_2), \tag{10}$$

$$(x_1 \vee x_2) \equiv (x_1 x_2 \oplus x_1 \oplus x_2), \tag{11}$$

$$(x_1 \oplus x_2)(x_3 \oplus x_4) \equiv (x_1 x_3 \oplus x_1 x_4 \oplus x_2 x_3 \oplus x_2 x_4). \tag{12}$$

Из этих эквивалентностей и теоремы 21 на стр. 86 легко получить первую часть следующего утверждения.

Теорема 26. *Для всякой булевой функции существует задающий её приведённый многочлен Жегалкина. Он единственен с точностью до перестановок слагаемых и порядка переменных в конъюнкциях.*

Доказательство. Существование такого многочлена следует из того, что для любой ДНФ или КНФ можно с помощью указанных эквивалентностей найти эквивалентный многочлен Жегалкина: (9)–(11) позволяют заменять все вхождения $\neg$, $\wedge$ и $\vee$ на $\oplus$ и умножение, а (12) — перемножать получившиеся после такой замены многочлены. Для построения приведённого многочлена используем коммутативность и ассоциативность конъюнкции и сложения по модулю два,

идемпотентность конъюнкции: $xx \equiv x$, а также эквивалентности $x \oplus x \equiv 0$ и $x \oplus 0 \equiv x$.

Для доказательства единственности представления подсчитаем число различных приведённых многочленов Жегалкина от n переменных. Каждая положительная элементарная конъюнкция имеет вид $x_{i_1} \cdots x_{i_k}$, где $1 \leqslant i_1 < \ldots < i_k \leqslant n$. Таких конъюнкций столько же, сколько подмножеств множества $\{x_1, \ldots, x_n\}$, то есть 2^n. Конъюнкция, соответствующая пустому подмножеству переменных, тождественно равна единице. Упорядочим их произвольным образом (например, лексикографически): $K_1, K_2, \ldots, K_{2^n}$. Тогда любой приведённый многочлен Жегалкина единственным образом можно представить как сумму

$$\alpha_1 K_1 \oplus \alpha_2 K_2 \oplus \ldots \oplus \alpha_{2^n} K_{2^n},$$

где каждый из коэффициентов α_i равен 0 или 1. Следовательно, число многочленов Жегалкина равно 2^{2^n}, то есть числу всех булевых функций от n переменных. Поэтому каждая функция задаётся в точности одним многочленом Жегалкина. □

Пример 15. *Пусть функция $f(x_1, x_2, x_3)$ задаётся ДНФ*

$$\Phi = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3).$$

Найдём многочлен Жегалкина, который также задаёт эту функцию.

Сначала заменяем $\wedge$ на умножение, а затем, применяя эквивалентность (9), устраняем отрицания и получаем:

$$\Phi \equiv x_1(x_2 \oplus 1) \vee (x_1 \oplus 1)x_2(x_3 \oplus 1).$$

Перемножив по правилам (12), получим:

$$\Phi \equiv (x_1x_2 \oplus x_1) \vee (x_1x_2x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_2).$$

Эквивалентность (11) позволяет устранить $\vee$:

$$\Phi \equiv (x_1x_2 \oplus x_1)(x_1x_2x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_2) \oplus \\ \oplus (x_1x_2 \oplus x_1) \oplus (x_1x_2x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_2).$$

Снова, используя (12), *перемножим первые две скобки:*

$$\Phi \equiv (x_1x_2x_1x_2x_3 \oplus x_1x_2x_1x_2 \oplus x_1x_2x_2x_3 \oplus x_1x_2x_2 \oplus \\ \oplus x_1x_1x_2x_3 \oplus x_1x_1x_2 \oplus x_1x_2x_3 \oplus x_1x_2) \oplus \\ \oplus (x_1x_2 \oplus x_1) \oplus (x_1x_2x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_2).$$

Далее устраним повторения переменных в конъюнкциях:

$$\Phi \equiv (x_1x_2x_3 \oplus x_1x_2 \oplus x_1x_2x_3 \oplus x_1x_2 \oplus \\ \oplus x_1x_2x_3 \oplus x_1x_2 \oplus x_1x_2x_3 \oplus x_1x_2) \oplus \\ \oplus (x_1x_2 \oplus x_1) \oplus (x_1x_2x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_2).$$

Упростим эту сумму, используя эквивалентности: $x \oplus x \equiv 0$ *и* $x \oplus 0 \equiv x$. *В результате получим приведённый многочлен Жегалкина*

$$P(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_2x_3 \oplus x_1x_2x_3,$$

эквивалентный исходной ДНФ Φ.

Если функция $f(x_1, \ldots, x_n)$ задана таблично, то для построения реализующего её многочлена Жегалкина можно применить метод неопределённых коэффициентов. Сопоставим i-му набору значений переменных $\sigma_i = (\sigma_i^1, \ldots, \sigma_i^n)$ в таблице положительную конъюнкцию

$$K_i = \bigwedge_{\sigma_i^j = 1} x_j$$

переменных, равных 1 в этом наборе. В частности, K_1 — пустая конъюнкция, $K_2 = x_n$, $K_3 = x_{n-1}$, $K_4 = (x_nx_{n-1})$ и т. д. Тогда для получения нужного многочлена Жегалкина достаточно определить все коэффициенты α_i, $i = 1, \ldots, 2^n$, в выражении

$$f(x_1, \ldots, x_n) = \alpha_1K_1 \oplus \alpha_2K_2 \oplus \ldots \oplus \alpha_{2^n}K_{2^n}.$$

Подставляя в это равенство значения переменных из набора σ_i, $i = 1, \ldots, 2^n$, мы получим 2^n линейных уравнений относительно 2^n неизвестных коэффициентов α_i. Решив эту систему, получим требуемый многочлен Жегалкина. Эта система треугольная и легко решается «сверху вниз»: каждое α_i определяется по значениям $\alpha_1, \ldots, \alpha_{i-1}$ из уравнения, соответствующего набору σ_i.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Рис. 10: Функция $f(x_1, x_2, x_3)$

Пример 16. *Рассмотрим в качестве примера функцию $f(x_1, x_2, x_3)$, заданную таблицей на рис. 10.*

Приведённый многочлен Жегалкина для неё (как и для любой функции от трёх переменных) представляется в общем виде

$$p(x_1, x_2, x_3) = \alpha_0 \oplus \alpha_1 x_1 \oplus \alpha_2 x_2 \oplus \alpha_3 x_3 \oplus \\ \oplus \alpha_{12} x_1 x_2 \oplus \alpha_{13} x_1 x_3 \oplus \alpha_{23} x_2 x_3 \oplus \alpha_{123} x_1 x_2 x_3.$$

В этом представлении в индексах у коэффициентов α перечислены номера переменных, входящих в соответствующие конъюнкции.

Последовательно подставляя значения переменных и f из таблицы, получаем:

$$p(0,0,0) = \alpha_0 = 1;$$
$$p(0,0,1) = \alpha_0 \oplus \alpha_3 = 0, \quad \alpha_3 = 1;$$
$$p(0,1,0) = \alpha_0 \oplus \alpha_2 = 0, \quad \alpha_2 = 1;$$
$$p(0,1,1) = \alpha_0 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_{23} = 0, \quad \alpha_{23} = 1;$$
$$p(1,0,0) = \alpha_0 \oplus \alpha_1 = 1, \quad \alpha_1 = 0;$$
$$p(1,0,1) = \alpha_0 \oplus \alpha_1 \oplus \alpha_3 \oplus \alpha_{13} = 0, \quad \alpha_{13} = 0;$$
$$p(1,1,0) = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_{12} = 0, \quad \alpha_{12} = 0;$$
$$p(1,1,1) = \alpha_0 \oplus \alpha_1 \oplus \alpha_2 \oplus \alpha_3 \oplus \alpha_{12} \oplus \alpha_{13} \oplus \alpha_{23} \oplus \alpha_{123} = 1, \quad \alpha_{123} = 1.$$

Следовательно, функция $f(x_1, x_2, x_3)$ представляется приведённым многочленом Жегалкина

$$p_f(x_1, x_2, x_3) = 1 \oplus x_2 \oplus x_3 \oplus x_2 x_3 \oplus x_1 x_2 x_3.$$

Чтобы показать, как можно быстро найти значения коэффициентов многочлена Жегалкина, введём следующее обозначение. С помощью $\alpha_{\sigma_1\ldots\sigma_n}$, $\sigma_1,\ldots,\sigma_n \in \mathbb{B}$, записываем коэффициент многочлена Жегалкина при конъюнкции, в которую x_i входит тогда и только тогда, когда $\sigma_i = 1$. Например, многочлен Жегалкина с переменными x_1, x_2 будет выглядеть так:

$$\alpha_{00} \oplus \alpha_{01}x_2 \oplus \alpha_{10}x_1 \oplus \alpha_{11}x_1x_2.$$

Если рассматривать строки из нулей и единиц как двоичные записи чисел, то это же самое можно записать

$$\alpha_0 \oplus \alpha_1x_2 \oplus \alpha_2x_1 \oplus \alpha_3x_1x_2.$$

При помощи f_i обозначим значение булевой функции f на i-м наборе аргументов при лексикографическом упорядочении, нумерацию начнём с нуля. Для функции f определим булеву треугольную матрицу J^f размера $2^n \times 2^n$, нумерацию строк и столбцов тоже будем производить с нуля. В матрице J^f нулевая строка содержит значения f_i: $J_{0,i} = f_i$. Каждая следующая строка матрицы J^f содержит суммы по модулю два чисел сверху от текущего и следующего за ним: $J^f_{i+1,\ell} = J^f_{i,\ell} \oplus J^f_{i,\ell+1}$ для $\ell = 0,\ldots,2^n - i - 1$. Тогда нулевой столбец этой матрицы даст упорядоченный по i набор коэффициентов α_i для многочлена Жегалкина функции f.

Например, для функции f на рис. 10 на предыдущей странице получаем:

$$J^f = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & \\ 1 & 0 & 1 & 0 & 1 & 1 & & \\ 1 & 1 & 1 & 1 & 0 & & & \\ 0 & 0 & 0 & 1 & & & & \\ 0 & 0 & 1 & & & & & \\ 0 & 1 & & & & & & \\ 1 & & & & & & & \end{pmatrix}$$

Как видим нулевой столбец имеет те же значения, что вычисленные коэффициенты в примере 16 на противоположной странице.

Чтобы доказать корректность этого метода введём такое отношение на натуральных числах: $a \subseteq b$, если в двоичной записи a и b на всех разрядах, на которых в a стоят единицы, в b тоже стоят единицы. Иначе говоря $a \subseteq b$ — это сокращение для $\rho^{-1}(a) \subseteq \rho^{-1}(b)$, см. пример 2 на стр. 33. Например, $5 \subseteq 13$, так как 5 в двоичной системе — «101», а 13 — «1101». Как видим, там, где в записи числа 5 стоят единицы, в записи числа 13 тоже стоят единицы. Тогда значение функции f через многочлен Жегалкина можно записать в таком виде:

$$f_i = \bigoplus_{k \subseteq i} \alpha_k. \tag{13}$$

С помощью знака $\otimes$ обозначим кратную сумму по модулю два:

$$N \otimes x = \underbrace{x \oplus \cdots \oplus x}_{N \text{ раз}},$$

где N — натуральное число. Легко проверить следующее равенство для натуральных N, M и пропозициональной переменной x:

$$(N + M) \otimes x = (N \otimes x) \oplus (M \otimes x). \tag{14}$$

Теорема 27. *Нулевой столбец матрицы J^f для булевой функции f содержит значения коэффициентов α_i многочлена Жегалкина для f, упорядоченные по возрастанию i.*

Доказательство. Индукцией по k легко доказать, что

$$J^f_{k,\ell} = \bigoplus_{i=0}^{k} C^i_k \otimes f_{\ell+i}. \tag{15}$$

Для $k = 0$ это выполнено по определению J^f, для $k+1$ получаем

$$J^f_{k+1,\ell} = J^f_{k,\ell} \oplus J^f_{k,\ell+1} = \bigoplus_{i=0}^{k} C^i_k \otimes f_{\ell+i} \oplus \bigoplus_{i=0}^{k} C^i_k \otimes f_{\ell+1+i} =$$
$$= \bigoplus_{i=0}^{k+1} C^i_k \otimes f_{\ell+i} \oplus \bigoplus_{i=0}^{k+1} C^{i-1}_k \otimes f_{\ell+i} = \bigoplus_{i=0}^{k+1} (C^i_k \otimes f_{\ell+i} \oplus C^{i-1}_k \otimes f_{\ell+i}) =$$

$$= \bigoplus_{i=0}^{k+1} (C_k^i + C_k^{i-1}) \otimes f_{\ell+i} = \bigoplus_{i=0}^{k+1} C_{k+1}^i \otimes f_{\ell+i}.$$

Здесь мы воспользовались равенствами (3) и (14).

Из задачи 42 на стр. 57 мы знаем, что C_n^k нечётно тогда и только тогда, когда $k \subseteq n$. Кроме того

$$C_k^i \otimes f_{\ell+i} = \begin{cases} 0, & \text{если } C_k^i \text{ чётно,} \\ f_{\ell+i}, & \text{иначе.} \end{cases}$$

Воспользовавшись этими двумя фактами для (15), мы получим

$$J_{k,\ell}^f = \bigoplus_{i \subseteq k} f_{\ell+i}.$$

В частности, при $\ell = 0$

$$J_{k,0}^f = \bigoplus_{i \subseteq k} f_i.$$

Если теперь для произвольного $m = 0, \ldots, 2^n - 1$ просуммировать это равенство по всем k таким, что $k \subseteq m$, получим

$$\bigoplus_{k \subseteq m} J_{k,0}^f = \bigoplus_{k \subseteq m} \bigoplus_{i \subseteq k} f_i = \bigoplus_{i \subseteq m} f_i \Bigg(\bigoplus_{i \subseteq k \subseteq m} 1 \Bigg).$$

С правой стороны единица суммируется столько раз, сколько существует k таких, что $i \subseteq k \subseteq m$. Количество слагаемых равно нулю, если не выполнено $i \subseteq m$, либо степени двойки в противном случае, причём показатель больше нуля, если i и m отличаются. Следовательно, единственным слагаемым, которое может быть не равно нулю, является f_m и мы получаем

$$\bigoplus_{k \subseteq m} J_{k,0}^f = f_m$$

для любого m. Но слева, согласно (13), стоит значение в m-й строке многочлена Жегалкина с коэффициентами $\alpha_k = J_{k,0}^f$. Так как его значения совпадают со значениями функции f для всех строк, то это — многочлен Жегалкина для функции f. □

Задачи

62. Доказать, что из Φ следует Ψ тогда и только тогда, когда $\Phi \wedge \Psi \equiv \Phi$, тогда и только тогда, когда $\Phi \vee \Psi \equiv \Psi$. ▼

63. Проверить все приведённые в параграфе 4.1 эквивалентности 1)–9), непосредственно построив истинностные таблицы для функций, представляемых их левыми и правыми частями.

64. Индукцией по построению Φ доказать, что $\operatorname{depth}(\Phi)^x_\Psi \leqslant \operatorname{depth}\Phi + \operatorname{depth}\Psi$. Привести пример, когда неравенство будет строгим. ▼

65. Назовём *логическим произведением* формулу вида $\Phi_1 \wedge \Phi_2 \wedge \ldots \wedge \Phi_n$ (в этом выражении использованы соглашения о сокращении записи!). Её подформулы $\Phi_i, 1 \leqslant i \leqslant n$, будем называть *сомножителями*. Аналогично, *логической суммой* назовём формулу вида $\Phi_1 \vee \Phi_2 \vee \ldots \vee \Phi_n$. Её подформулы Φ_i, $1 \leqslant i \leqslant n$, будем называть *слагаемыми*.

Показать, что из основных тождеств можно вывести следующие правила преобразования логических произведений и сумм:

(а) Если в логическом произведении хотя бы один из сомножителей равен 0, то и всё произведение равно 0.

(б) Если в логической сумме хотя бы одно из слагаемых равно 1, то и вся сумма равна 1.

(в) Если в логическом произведении $n \geqslant 2$ и есть сомножитель равный 1, то его можно вычеркнуть.

(г) Если в логической сумме $n \geqslant 2$ и есть слагаемое равное 0, то его можно вычеркнуть. ▼

66. Используя основные тождества, доказать эквивалентность следующих пар формул.

(а) $\neg(x \vee \neg y) \wedge (x \to \neg y)$ и $(\neg x \wedge y)$;

(б) $\neg((x \wedge \neg y) \to (\neg x \vee z))$ и $(x \wedge \neg y \wedge \neg z)$;

(в) $(x \oplus y) \to (x \wedge \neg y)$ и $(\neg x \wedge \neg y) \vee x$. ▼

67. Пусть интерпретация J отличается от I только тем, что $J(x) = I(\Theta)$. Доказать, что $J(\Phi) = I\big((\Phi)^x_\Theta\big)$. ▼

68. Доказать предложение 19 на стр. 83. ▼

69. Доказать вторую часть теоремы 21 на стр. 86: для каждого набора значений аргументов $\sigma_1, \ldots, \sigma_n$ выполнено $f(\sigma_1, \ldots, \sigma_n) = \mathcal{C}_f(\sigma_1, \ldots, \sigma_n)$. ▼

70. Предложить процедуры для решения следующих задач:

(а) По произвольной элементарной конъюнкции построить эквивалентную ей совершенную ДНФ с заданным множеством переменных.

(б) По произвольной элементарной дизъюнкции построить эквивалентную ей совершенную КНФ с заданным множеством переменных. ▼

71. Доказать, что для всех $k \leqslant n$ каждую булеву функцию $f \in \mathcal{P}_n$ можно представить в виде

$$f(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n) = \\ = \bigvee_{(\sigma_1,\ldots,\sigma_k)\in\mathbb{B}^k} x_1^{\sigma_1} \wedge \ldots x_k^{\sigma_k} \wedge f(\sigma_1, \ldots, \sigma_k, x_{k+1}, \ldots, x_n).$$

Такое представление называется разложением f по $x_1, \ldots, x_k$. При $k = n$ из него получается совершенная ДНФ из теоремы 21 на стр. 86. ▼

72. Как изменить процедуру приведения к совершенной ДНФ, чтобы в результате получить процедуру приведения к совершенной КНФ? ▼

73. Разработать алгоритм одновременного построения эквивалентных ДНФ и КНФ для произвольной формулы Φ, который позволял бы избежать раздельного рассмотрения случаев 6)–11), используя индукцию по построению Φ. ▼

74. Найти эквивалентные сокращённые ДНФ и доказать эквивалентность следующих пар формул:

(а) $\Phi = (((\neg x \wedge \neg y) \to \neg z) \wedge (x \to y))$, $\Psi = ((1 \oplus y) \to (\neg x \wedge (1 \oplus z)))$;

(б) $\Phi = (\neg((x_1 \to x_2) \vee \neg(x_2 \to x_1)) \wedge x_3)$, $\Psi = \neg((x_1 \wedge x_3) \to x_2)$;

(в) $\Phi = \neg(\neg x \wedge y \wedge \neg z) \to ((y \oplus 1) \wedge ((x \oplus 1) \to \neg(\neg u \vee \neg z)))$,
$\Psi = (\neg x \vee y) \to ((\neg u \vee y \vee z) \to (\neg(x \vee \neg y) \wedge \neg z))$. ▼

75. Доказать эквивалентности (9)–(12). ▼

76. Доказать равенства (13) и (14). ▼

77. Используя основные эквивалентности, найти эквивалентные многочлены Жегалкина и доказать эквивалентность следующих формул: ▼

$$\Phi = ((z \wedge (x \to y)) \vee \neg(\neg x \to z)); \quad \Psi = (x \to (y \wedge z)).$$

78. Найти сокращённую дизъюнктивную нормальную форму и многочлен Жегалкина (методом неопределённых коэффициентов и с помощью матрицы J) для следующих функций от трёх аргументов. Считаем, что наборы их аргументов упорядочены лексикографически и значения на них задаются последовательностью 8 нулей и единиц:

(а) $f = (0010\ 1100)$;

(б) $f = (1110\ 1100)$;

(в) $f = (1100\ 0011)$;

(г) $f = (0110\ 1011)$. ▼

79. Найти булеву функцию от n переменных, у которой приведённый многочлен Жегалкина имеет наибольшую длину. ▼

Глава 5

Полные множества функций и теорема Поста

Краткое содержание: замкнутые множества функций, полные множества булевых функций, множества функций, сохраняющих ноль, сохраняющих единицу, самодвойственных, монотонных и линейных функций, критерий полноты множества булевых функций (теорема Поста), базисы.

Ключевые слова: замыкание множества функций, полное множество функций, замкнутое множество функций, сохраняющая ноль функция, сохраняющая единицу функция, самодвойственная функция, монотонная функция, линейная функция, базис.

§ 5.1. Замкнутые множества функций

Пусть $\mathcal{P}$ — это множество всех булевых функций:

$$\mathcal{P} = \bigcup_{n=0}^{\infty} \mathcal{P}_n.$$

В предыдущем разделе мы установили, что любую функцию из $\mathcal{P}$ можно задать булевой формулой над множеством $F_B = \{\neg, \wedge, \vee\}$ (в качестве этих формул можно, например, взять соответствующие ДНФ и КНФ). Такие множества функций называются полными.

Определение 40 (Полное множество)**.** *Множество булевых функций F называется* п о л н ы м*, если формулами над этим множеством можно задать всякую булеву функцию из $\mathcal{P}$.*

Другим уже известным нам примером полного множества функций является $F_J = \{0, 1, \wedge, \oplus\}$, позволяющее задать произвольную булеву функцию с помощью многочлена Жегалкина. Разумеется, не всякое множество является полным. Например, формулами над множеством $\{\vee\}$ невозможно выразить функцию тождественно равную 0 (почему?). Наша цель в этом разделе — найти критерий, позволяющий по множеству функций определять его полноту.

Для исследования полноты полезно следующее понятие.

Определение 41 (Замыкание)**.** *З а м ы к а н и е $[F]$ множества функций F — это множество всех функций, которые можно задать с помощью формул над F.*

Тогда определение полного множества можно переформулировать так: множество F называется п о л н ы м, если $[F] = \mathcal{P}$.

Предложение 28. *Замыкание обладает следующими свойствами:*

1) $F \subseteq [F]$;
2) $[[F]] = [F]$;
3) *если* $F \subseteq G$, *то* $[F] \subseteq [G]$;
4) *если множество F является полным и $F \subseteq [G]$, то и множество G является полным.*

Доказательство. Все эти утверждения достаточно просто получаются из определения замыкания. Пункты 1) и 3) очевидны.

Справедливость пункта 2) следует из того, что всякая функция из $[F]$ задаётся некоторой формулой над F, а тогда всякая функция из $[[F]]$, которая задаётся суперпозицией функций из $[F]$, задаётся также некоторой формулой над F.

Пункт 4) следует из 2) и 3):

$$F \subseteq [G] \Rightarrow [F] \subseteq [[G]] \Rightarrow [F] \subseteq [G]$$

и так как $[F] = \mathcal{P}$, то и $[G] = \mathcal{P}$. □

Пункт 4) предыдущего предложения позволяет устанавливать полноту некоторого множества, выражая с его помощью все функции другого множества, полнота которого уже установлена.

Пример 17. *Законы де Моргана позволяют выразить* $\vee$ *через пару* $\neg$ *и* $\wedge$*:*

$$x \vee y \equiv \neg(\neg x \wedge \neg y),$$

а $\wedge$ *— через пару* $\neg$ *и* $\vee$*:*

$$x \wedge y \equiv \neg(\neg x \vee \neg y).$$

Поэтому каждое из множеств $F_\wedge = \{\neg, \wedge\}$ *и* $F_\vee = \{\neg, \vee\}$ *также является полным. Эквивалентности 7) и 8) позволяют выразить* $\vee$ *через пару* $\neg$ *и* $\to$*:*

$$x \vee y \equiv \neg x \to y.$$

Следовательно, полным будет и множество $F_\to = \{\neg, \to\}$.

Имеются ли полные множества из одной двуместной функции? Да. Рассмотрим множество, $\{\uparrow\}$, включающее лишь штрих Шеффера. Напомним, что

$$(x \uparrow y) \equiv \neg(x \wedge y).$$

Тогда нетрудно проверить, что

$$\neg x \equiv (x \uparrow x) \quad \text{и} \quad (x \wedge y) \equiv ((x \uparrow y) \uparrow (x \uparrow y)).$$

Следовательно, множество $\{\uparrow\}$ — полное.

Определение 42 (Замкнутое множество)**.** *Множество функций* F *называется* *замкнутым*, *если* $F = [F]$.

Очевидно, что если в замкнутом множестве F отсутствует хотя бы одна функция из $\mathcal{P}$, то F не является полным.

Далее в этом разделе мы будем использовать верхний индекс в круглых скобках для указания числа аргументов функции, то есть $f^{(n)}$ означает, что $f \in \mathcal{P}_n$.

Определим пять важных замкнутых множеств.

Определение 43 (Функции, сохраняющие ноль)**.** *Булева функция $f^{(n)}$ сохраняет ноль, если $f(0,0,\ldots,0)=0$. Множество всех функций, сохраняющих ноль, обозначим через $\mathcal{S}_0$.*

Определение 44 (Функции, сохраняющие единицу)**.** *Если для булевой функции $f^{(n)}$ выполнено $f(1,1,\ldots,1)=1$, то f сохраняет единицу. Множество всех таких функций обозначим $\mathcal{S}_1$.*

Определение 45 (Самодвойственные функции)**.** *Булева функция $f^{(n)}$ называется самодвойственной, если для каждого набора аргументов $(\sigma_1,\sigma_2,\ldots,\sigma_n)\in\mathbb{B}^n$ имеет место равенство:*

$$f(\sigma_1,\sigma_2,\ldots,\sigma_n)=\neg f(\neg\sigma_1,\neg\sigma_2,\ldots,\neg\sigma_n).$$

Таким образом, самодвойственные функции принимают на противоположных наборах противоположные значения. Множество всех самодвойственных функций обозначим через $\mathcal{S}$.

Определение 46 (Линейные функции)**.** *Булева функция $f^{(n)}$ называется линейной, если она может быть задана линейным многочленом Жегалкина вида*

$$\alpha_0\oplus\alpha_1x_1\oplus\alpha_2x_2\oplus\cdots\oplus\alpha_nx_n,$$

где $\alpha_i\in\{0,1\}$ при $i=0,1,2,\ldots,n$. Множество всех линейных функций обозначим через $\mathcal{L}$.

Определение 47 (Монотонные функции)**.** *Булева функция $f^{(n)}$ называется монотонной, если для всяких двух наборов аргументов $(\sigma_1,\sigma_2,\ldots,\sigma_n)\in\mathbb{B}^n$ и $(\rho_1,\rho_2,\ldots,\rho_n)\in\mathbb{B}^n$ таких, что $\sigma_j\geqslant\rho_j$ для всех $j=1,\ldots,n$ имеет место неравенство*

$$f(\sigma_1,\sigma_2,\ldots,\sigma_n)\geqslant f(\rho_1,\rho_2,\ldots,\rho_n).$$

Множество всех монотонных функций обозначим через $\mathcal{M}$.

x_1	x_2	x_3	f_1	f_2	f_3	f_4	f_5
0	0	0	1	1	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	1	1	1
0	1	1	1	0	0	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	0	0	0
1	1	0	0	1	1	0	1
1	1	1	0	1	1	1	1

Рис. 11: Функции f_1, f_2, f_3, f_4 и f_5.

Пример 18. *Рассмотрим для примера пять функций от трёх переменных, которые представлены на рис. 11.*

Из определений непосредственно следует, что f_3, f_4 и f_5 сохраняют ноль, то есть входят в $\mathcal{S}_0$, а функции f_2, f_3, f_4 и f_5 сохраняют единицу, то есть входят в $\mathcal{S}_1$. Функция f_3 является самодвойственной, а f_2 — нет, так как $f_2(0,0,0) = f_2(1,1,1)$. Функция f_2 является линейной — она задаётся многочленом $x_1 \oplus x_2 \oplus 1$. Функция f_5 является монотонной, а f_3 — нет, так как $f_3(0,1,1) = 0 < 1 = f_3(0,1,0)$.

Теорема 29. *Множества $\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}, \mathcal{L}$ и $\mathcal{M}$ являются замкнутыми.*

Доказательство. Замкнутость всех указанных множеств устанавливается одним и тем же способом. Пусть $\mathcal{F}$ — это одно из множеств $\mathcal{S}_0$, $\mathcal{S}_1$, $\mathcal{S}$, $\mathcal{L}$, $\mathcal{M}$, и функция $f \in [\mathcal{F}]$ задаётся некоторой формулой над $\mathcal{F}$. Нужно показать, что тогда $f \in \mathcal{F}$. Доказываем это индукцией по построению формулы для функции f.

Базис индукции, когда эта формула есть переменная x, очевиден: функция $f(x) = x$ принадлежит всем пяти множествам.

Индукционный шаг: пусть

$$f(x_1, \ldots, x_k) = g(f_1(x_1, \ldots, x_k), \ldots, f_n(x_1, \ldots, x_k)),$$

и все функции $g^{(n)}, f_1^{(k)}, \ldots, f_n^{(k)}$ входят в F. Требуется показать, что тогда и f входит в $\mathcal{F}$.

Для $\mathcal{F} = \mathcal{S}_0$ и $\mathcal{F} = \mathcal{S}_1$ это просто:

$$f(0,0,\ldots,0) = g(f_1(0,\ldots,0),\ldots,f_n(0,\ldots,0)) = g(0,0,\ldots,0) = 0;$$
$$f(1,1,\ldots,1) = g(f_1(1,\ldots,1),\ldots,f_n(1,\ldots,1)) = g(1,1,\ldots,1) = 1.$$

Если $\mathcal{F} = \mathcal{S}$ и $(\sigma_1,\sigma_2,\ldots,\sigma_k)$ — произвольный набор аргументов, то получаем

$$\begin{aligned} f(\sigma_1,\sigma_2,\ldots,\sigma_k) &= \neg g(\neg f_1(\sigma_1,\ldots,\sigma_k),\ldots,\neg f_n(\sigma_1,\ldots,\sigma_k)) = \\ &= \neg g(\neg\neg f_1(\neg\sigma_1,\ldots,\neg\sigma_k),\ldots,\neg\neg f_n(\neg\sigma_1,\ldots,\neg\sigma_k)) = \\ &= \neg g(f_1(\neg\sigma_1,\ldots,\neg\sigma_k),\ldots,f_n(\neg\sigma_1,\ldots,\neg\sigma_k)) = \\ &= \neg f(\neg\sigma_1,\neg\sigma_2,\ldots,\neg\sigma_k). \end{aligned}$$

Следовательно, $f \in \mathcal{S}$.

Пусть $\mathcal{F} = \mathcal{L}$. Так как функция $g^{(n)}$ и все функции $f_i^{(k)}$ линейны, то существуют коэффициенты $\alpha_0,\alpha_1,\ldots,\alpha_n$ и $\beta_{0i},\beta_{1i},\ldots,\beta_{ki}$, $i = 1,\ldots,n$, такие, что

$$g(x_1,\ldots,x_n) = \alpha_0 \oplus \bigoplus_{i=1}^{n} \alpha_i x_i; \quad f_i(x_1,\ldots,x_k) = \beta_{0i} \oplus \bigoplus_{j=1}^{k} \beta_{ji} x_j$$

для $i = 1,\ldots,n$. Подставив эти выражения в формулу для $f^{(k)}$, получим

$$\begin{aligned} f(x_1,\ldots,x_k) &= \alpha_0 \oplus \bigoplus_{i=1}^{n} \alpha_i \left(\beta_{0i} \oplus \bigoplus_{j=1}^{k} \beta_{ji} x_j \right) = \\ &= \alpha_0 \oplus \bigoplus_{i=1}^{n} \alpha_i\beta_{0i} \oplus \bigoplus_{i=1}^{n}\bigoplus_{j=1}^{k} \alpha_i\beta_{ji} x_j = \\ &= \underbrace{\left(\alpha_0 \oplus \bigoplus_{i=1}^{n} \alpha_i\beta_{0i} \right)}_{\gamma_0} \oplus \bigoplus_{j=1}^{k} \underbrace{\left(\bigoplus_{i=1}^{n} \alpha_i\beta_{ji} \right)}_{\gamma_j} x_j = \gamma_0 \oplus \bigoplus_{j=1}^{k} \gamma_j x_j, \end{aligned}$$

где $\gamma_0, \gamma_1, \ldots, \gamma_k$ — значения сумм констант в соответствующих скобках. Следовательно, $f \in \mathcal{L}$.

Наконец рассмотрим множество монотонных функций $\mathcal{F} = \mathcal{M}$. Если $g^{(n)}$ и все $f_i^{(k)} \in \mathcal{M}$ и $(\sigma_1, \sigma_2, \ldots, \sigma_k)$ и $(\rho_1, \rho_2, \ldots, \rho_k)$ — два набора аргументов такие, что для всех $j = 1, \ldots, k$ выполнено $\sigma_j \geqslant \rho_j$, то

$$f_i(\sigma_1, \sigma_2, \ldots, \sigma_k) \geqslant f_i(\rho_1, \rho_2, \ldots, \rho_k)$$

и поэтому

$$f(\sigma_1, \sigma_2, \ldots, \sigma_k) = g(f_1(\sigma_1, \sigma_2, \ldots, \sigma_k), \ldots, f_n(\sigma_1, \sigma_2, \ldots, \sigma_k)) \geqslant$$
$$\geqslant g(f_1(\rho_1, \rho_2, \ldots, \rho_k), \ldots, f_n(\rho_1, \rho_2, \ldots, \rho_k)) = f(\rho_1, \rho_2, \ldots, \rho_k).$$

Таким образом, $f \in \mathcal{M}$. □

Следствие 30. *Множества функций $\mathcal{S}_0$, $\mathcal{S}_1$, $\mathcal{S}$, $\mathcal{L}$ и $\mathcal{M}$ не являются полными.*

Доказательство. Штрих Шеффера $\uparrow$ не принадлежит ни одному из этих множеств. □

§ 5.2. Критерий полноты (теорема Поста)

Оказывается, что всякое множество, не включаемое ни в одно из указанных пяти замкнутых множеств, полно.

Теорема 31 (Теорема Поста)**.** *Множество булевых функций F является полным тогда и только тогда, когда оно не включено ни в одно из множеств $\mathcal{S}_0$, $\mathcal{S}_1$, $\mathcal{S}$, $\mathcal{L}$ и $\mathcal{M}$.*

Иными словами, в F имеется хотя бы одна функция, не сохраняющая ноль, хотя бы одна функция, не сохраняющая единицу, хотя бы одна несамодвойственная функция, хотя бы одна нелинейная функция и хотя бы одна немонотонная функция.

Прежде чем доказывать саму теорему установим несколько вспомогательных лемм.

Лемма 32. *С помощью констант* 0, 1 *и произвольной немонотонной функции f_m можно выразить отрицание.*

Доказательство. Так как функция f_m немонотонна, то найдутся два набора $(\sigma_1, \ldots, \sigma_n)$ и $(\rho_1, \ldots, \rho_n)$, для которых $\sigma_i \geqslant \rho_i$ для $i = 1, \ldots, n$, но при этом $f_m(\sigma_1, \ldots, \sigma_n) < f_m(\rho_1, \ldots, \rho_n)$. Последнее неравенство возможно только в случае $f_m(\sigma_1, \ldots, \sigma_n) = 0$ и $f_m(\rho_1, \ldots, \rho_n) = 1$. Разобьём все $i = 1, \ldots, n$ на три части, выберем переменную x и определим для каждого i соответствующее x_i^*:

1) $\sigma_i = \rho_i = 0$, тогда $x_i^* = 0$;
2) $\sigma_i = \rho_i = 1$, тогда $x_i^* = 1$;
3) $\sigma_i = 1$ и $\rho_i = 0$ (противоположный вариант невозможен), тогда $x_i^* = x$.

Пусть теперь $f'_m(x) = f_m(x_1^*, \ldots, x_n^*)$.

Заметим, что набор $(x_1^*, \ldots, x_n^*)$ всегда совпадает с наборами $(\sigma_1, \ldots, \sigma_n)$ и $(\rho_1, \ldots, \rho_n)$ в позициях i, принадлежащих частям 1) и 2). В части 3) набор $(x_1^*, \ldots, x_n^*)$ совпадает с $(\sigma_1, \ldots, \sigma_n)$ при $x = 1$ и совпадает с $(\rho_1, \ldots, \rho_n)$ при $x = 0$. Следовательно, при $x = 1$ получаем $(x_1^*, \ldots, x_n^*) = (\sigma_1, \ldots, \sigma_n)$, а при $x = 0$ — $(x_1^*, \ldots, x_n^*) = (\rho_1, \ldots, \rho_n)$.

Таким образом, имеем:

$$f'_m(1) = f_m(\sigma_1, \ldots, \sigma_n) = 0; \quad f'_m(0) = f_m(\rho_1, \ldots, \rho_n) = 1.$$

Это означает, что $f'_m(x)$ — отрицание. □

Лемма 33. *С помощью отрицания* $\neg$ *и произвольной несамодвойственной функции* f_s *можно выразить константы* 0 *и* 1.

Доказательство. Так как функция f_s несамодвойственна, то для некоторого набора $(\sigma_1, \ldots, \sigma_n)$ выполнено

$$f_s(\sigma_1, \ldots, \sigma_n) \neq \neg f_s(\neg\sigma_1, \ldots, \neg\sigma_n),$$

а поскольку функция может принимать всего два значения, то

$$f_s(\sigma_1, \ldots, \sigma_n) = f_s(\neg\sigma_1, \ldots, \neg\sigma_n).$$

Рассмотрим функцию $f'_s(x) = f_s(x^{\sigma_1}, \ldots, x^{\sigma_n})$. Для неё получим

$$f'_s(0) = f_s(0^{\sigma_1}, \ldots, 0^{\sigma_n}) = f_s(\neg\sigma_1, \ldots, \neg\sigma_n) = \\ = f_s(\sigma_1, \ldots, \sigma_n) = f_s(1^{\sigma_1}, \ldots, 1^{\sigma_n}) = f'_s(1).$$

Таким образом, одноместная функция $f'_s(x)$ должна быть константой 0 или 1, тогда $\neg f'_s(x)$ даст вторую константу. □

Лемма 34. *С помощью констант* 0, 1, *отрицания* $\neg$ *и произвольной нелинейной функции* f_ℓ *можно выразить конъюнкцию* $\wedge$.

Доказательство. Так как функция f_ℓ нелинейна, то её приведённый многочлен Жегалкина содержит хотя бы одну конъюнкцию с двумя или более элементами: $x_{i_1} \cdots x_{i_k}$, $k \geqslant 2$. Выберем из таких конъюнкций самую короткую K, а в ней выберем две переменные, например, x и y. Пусть $u_1, \ldots, u_t$ — это остальные переменные (кроме x и y), входящие в K, а $v_1, \ldots, v_s$ — переменные, которые в K не входят.

Рассмотрим теперь функцию $f'_\ell(x, y)$, полученную подстановкой в f_ℓ константы 1 вместо всех u_i и константы 0 вместо всех v_j. При такой замене от конъюнкции K останется только xy, поскольку остальные её переменные превратятся в 1. Любые другие конъюнкции длины 2 или больше обязательно должны содержать хотя бы одну переменную v_j, поэтому их значения станут равны 0 и их можно исключить из суммы. Следовательно, многочлен Жегалкина для функции $f'_\ell(x, y)$ имеет вид

$$xy \oplus \alpha_x x \oplus \alpha_y y \oplus \alpha_0,$$

где $\alpha_x, \alpha_y, \alpha_0 \in \{0, 1\}$.

Если $\alpha_x = 0$, то положим $f''_\ell = f'_\ell$. В противном случае, если $\alpha_x = 1$, полагаем

$$f''_\ell(x, y) = f'_\ell(x, \neg y) = x(y \oplus 1) \oplus x \oplus \alpha_y(y \oplus 1) \oplus \alpha_0 = \\ = xy \oplus x \oplus x \oplus \alpha_y y \oplus \alpha_y \oplus \alpha_0 = xy \oplus \alpha_y y \oplus \alpha'_0.$$

Если $\alpha_y = 0$, то $f'''_\ell = f''_\ell$. В противном случае, если $\alpha_y = 1$:

$$f'''_\ell(x, y) = f''_\ell(\neg x, y) = (x \oplus 1)y \oplus y \oplus \alpha'_0 =$$

$$= xy \oplus y \oplus y \oplus \alpha'_0 = xy \oplus \alpha'_0.$$

Если теперь $\alpha'_0 = 0$, то сразу получаем $x \wedge y = f'''_\ell(x, y)$, а если $\alpha'_0 = 1$, то $x \wedge y = \neg f'''_\ell(x, y)$. □

Теперь можно перейти к доказательству теоремы Поста.

Доказательство. Необходимость условия теоремы непосредственно следует из установленного выше следствия 30 на стр. 110.

Для доказательства достаточности предположим, что F содержит не сохраняющую ноль функцию f_0, не сохраняющую единицу функцию f_1, несамодвойственную функцию f_s, немонотонную функцию f_m и нелинейную функцию f_ℓ. Покажем, что с помощью этих функций всегда можно выразить функции одного из уже известных нам полных множеств: $\{\neg, \wedge\}$.

Рассмотрим функции f_1 и f_0 и построим одноместные функции $f'_1(x)$ и $f'_0(x)$, отождествив все аргументы:

$$f'_1(x) = f_1(x, \ldots, x); \quad f'_0(x) = f_0(x, \ldots, x).$$

Тогда получим

$$f'_1(1) = f_1(1, \ldots, 1) = 0; \quad f'_0(0) = f_0(0, \ldots, 0) = 1.$$

Взглянув на все возможные одноместные булевы функции (рис. 3 на стр. 63), легко убедиться, что f'_1 может быть константой 0 или отрицанием, а f'_0 — константой 1 или отрицанием.

Таким образом, имеет место один из четырёх случаев:

1) $f'_1 = 0$, $f'_0 = 1$. В этом случае по лемме 32 на стр. 110 из f'_1, f'_0 и f_m можно получить отрицание $\neg$;
2) $f'_1 = 0$, $f'_0 = \neg$. Тогда $1 = f'_0(f'_1)$;
3) $f'_1 = \neg$, $f'_0 = 1$. Тогда $0 = f'_1(f'_0)$;
4) $f'_1 = f'_0 = \neg$. В этом случае по лемме 33 на стр. 111 из f'_1 и f_s можно получить константы 0 и 1.

x_1	x_2	x_3	f	g	h
0	0	0	1	1	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	1	0

Рис. 12: Функции f, g, h.

Итак, во всех четырёх случаях мы можем получить отрицание $\neg$ и константы 0 и 1. Теперь по лемме 34 на стр. 112 из $\neg$, 0, 1 и функции f_ℓ можно построить конъюнкцию.

В результате, мы смогли из функций множества F получить отрицание и конъюнкцию, которые образуют полное множество. Поэтому и наше множество F полно. □

Пример 19. *Рассмотрим набор функций f и g, представленный на рис. 12.*

Функция f, очевидно, немонотонна, не сохраняет ноль и единицу, но является самодвойственной и линейной. Функция

$$g(x_1, x_2, x_3) = 1 \oplus x_1 \oplus x_2 \oplus x_1 \wedge x_3 \oplus x_1 \wedge x_2 \wedge x_3 \tag{16}$$

является несамодвойственной, немонотонной, нелинейной не сохраняет ноль, но сохраняет единицу. Следовательно, по теореме Поста можно утверждать, что множество $\{f, g\}$ является полным. Покажем, как именно выразить необходимые функции через f и g.

Построим функции $f'(x) = f(x, x, x)$ и $g'_0(x) = g(x, x, x)$. Легко видеть, что $f'(x) = \neg x$ и $g'_0(x) = 1$. Тогда получаем $f'(g'_0(x)) = 0$. Таким образом, получены отрицание и обе константы.

Самой короткой конъюнкцией в многочлене Жегалкина (16) является $x_1 \wedge x_3$, поэтому вместо x_2 нужно подставить 0 и мы получим

$$g'_\ell(x_1, x_3) = g(x_1, 0, x_3) = 1 \oplus x_1 \oplus x_1 \wedge x_3.$$

Теперь

$$g''_\ell(x_1, x_3) = g'_\ell(x_1, \neg x_3) = 1 \oplus x_1 \oplus x_1 \wedge (x_3 \oplus 1) = 1 + x_1 \wedge x_3,$$

и, наконец, $x_1 \wedge x_3 = \neg g''_\ell(x_1, x_3)$.

Можно пойти и другим путём, если заметить, что при $x_1 = 1$ *функция* g *представляет собой дизъюнкцию:* $x_2 \vee x_3 = g(1, x_2, x_3)$.

Определение 48 (Базис)**.** *Полное множество функций называется* б а з и с о м *(или* м и н и м а л ь н ы м п о л н ы м м н о ж е с т в о м*), если после удаления из него любой функции оно перестаёт быть полным.*

Например, множество $\{\neg, \wedge, \vee\}$ не является базисом, а каждое из множеств $\{\neg, \wedge\}, \{\neg, \vee\}$ является. Мы уже знаем, что имеются полные множества, состоящие из одной функции (например, множество $\{\uparrow\}$). Они, конечно, являются базисами. Имеются и полные множества, включающие 3 функции, например, $\{1, \wedge, \oplus\}$, на котором основаны многочлены Жегалкина (отметим, что константу 0 можно выразить как $1 \oplus 1$). Из теоремы Поста следует, что никакое минимальное множество не может содержать более пяти функций. Оказывается, что для полноты всегда достаточно четырёх функций.

Теорема 35. *Во всяком базисе* F *содержится не более четырёх функций.*

Доказательство. Действительно, по теореме Поста в базисе F имеется функция $f_0 \notin \mathcal{S}_0$. Поэтому $f_0(0, 0, \ldots, 0) = 1$. Если она хотя бы на одном наборе значений аргументов принимает значение 0, то она немонотонна, то есть $f_0 \notin \mathcal{M}$. В противном случае, f_0 на всех наборах значений аргументов равна 1. Но тогда она несамодвойственна, так как $f_0(0, 0, \ldots, 0) = 1 = f_0(1, 1, \ldots, 1)$. Во всех случаях $f_0 \notin \mathcal{M}$ или $f_0 \notin \mathcal{S}$ и, следовательно, в множестве F содержится не более четырёх функций. □

Может быть, базис всегда содержит не более трёх функций? Это не так.

Пример 20. *Рассмотрим множество* $F^{(4)} = \{0, 1, x \wedge y, x \oplus y \oplus z\}$. *В этом множестве первые три функции несамодвойственны,* 0 *является единственной функцией, не сохраняющей единицу,* 1 *— единственной функцией, не сохраняющей ноль,* $x \wedge y$ *— единственной нелинейной функцией, а* $x \oplus y \oplus z$ *— единственной немонотонной функцией. Таким образом, множество* $F^{(4)}$

полно, но при выкидывании любой из функций перестанет быть полным, следовательно, оно является базисом.

Как видим, предыдущая теорема даёт наилучшую верхнюю оценку числа функций в базисе.

Замечание 6. Э. Пост в работах, опубликованных в 1921 и 1941 гг. не только установил критерий полноты, но и описал структуру всех замкнутых множеств функций в $\mathcal{P}$. Он показал, что число таких множеств счётно, а в каждом из них имеется собственный конечный базис.

Задачи

80. Доказать полноту множества $\{\downarrow\}$, включающего только стрелку Пирса, непосредственно выразив через неё отрицание, дизъюнкцию и конъюнкцию. ▼

81. Определить принадлежность каждой из функций f_1, f_2, f_3, f_4 и f_5, представленных в таблице на рис. 11 на стр. 108, каждому из множеств $\mathcal{S}_0$, $\mathcal{S}_1$, $\mathcal{S}$, $\mathcal{L}$ и $\mathcal{M}$. ▼

82. Используя результаты задачи 81, определить, какие из троек функций, представленных на рис. 11 на стр. 108, являются полными множествами. Имеются ли среди них полные множества из двух функций? Из одной функции? ▼

83. Проверить полноту множества функций $\{g, h\}$, представленных в таблице на рис. 12 на стр. 114. Если она полна, выразить с помощью этих функций обе константы, отрицание $\neg$ и импликацию $\to$. ▼

84. Доказать, что множество $\{\vee, \wedge, \to\}$ не является полным. Можно ли его сделать полным, добавив некоторую константу? ▼

85. Выразить функции $\{0, 1, \vee, \wedge, \leftrightarrow\}$ с помощью формул, построенных из функций полного множества $\{\neg, \to\}$. ▼

86. Определить количество функций из $\mathcal{P}_n$, принадлежащих каждому из множеств $\mathcal{S}_0$, $\mathcal{S}_1$, $\mathcal{S}$ и $\mathcal{L}$. ▼

87. Доказать, что для монотонных функций $f^{(n)} \in \mathcal{M}$ справедливо представление

$$f(x_1, \ldots, x_n) = (x_i \wedge f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)) \vee \\ \vee f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n),$$

для всех $i = 1, \ldots, n$. Вывести отсюда (индукцией по n), что для всякой монотонной функции, отличной от константы, существуют задающие её ДНФ и КНФ, не содержащие отрицаний переменных. ▼

88. Доказать, что число монотонных функций в $\mathcal{P}_n$ не меньше $2^{C_n^{[n/2]}}$. ▼

89. Назовём функцию $f(x_1, \ldots, x_n)$ линейной по переменной x_i, если f можно представить в виде $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n) \oplus \alpha_i x_i$, где $g(x_1, \ldots, x_n)$ — некоторая функция, не зависящая от x_i, $\alpha_i \in \{0, 1\}$. Доказать, что булева функция

$f(x_1, \ldots, x_n)$ является линейной тогда и только тогда, когда она линейна по всем переменным. ▼

90. Доказать, что функция $f(x_1, \ldots, x_n)$ линейна по переменной x_i тогда и только тогда, когда при изменении значения переменной x_i значение функции либо всегда меняется (то есть для всех значений остальных переменных), либо никогда не меняется (то есть f от x_i не зависит). ▼

91. Доказать, что булева функция $f^{(n)}$ является монотонной тогда и только тогда, когда она монотонна по каждому аргументу: для всех $i = 1, \ldots, n$ и всех $\sigma_1, \ldots, \sigma_n \in \mathbb{B}$ выполнено неравенство ▼

$$f(\sigma_1, \ldots, \sigma_{i-1}, 0, \sigma_{i+1}, \ldots, \sigma_n) \leqslant f(\sigma_1, \ldots, \sigma_{i-1}, 1, \sigma_{i+1}, \ldots, \sigma_n).$$

92. Доказать, что булева функция $f(x_1, \ldots, x_n)$ является монотонной тогда и только тогда, когда её сокращённая ДНФ не содержит отрицаний. ▼

93. Найти все базисы, которые можно получить, удаляя функции из множества $\{0, 1, \wedge, \vee, \to\}$. ▼

94. Определить, что будет замыканием множества $\{\wedge, \vee\}$. ▼

95. Определить, как по заданной ДНФ (КНФ) определить, будет ли она сохранять ноль или единицу, не вычисляя её значений. ▼

96. Функция $f^*(x_1, \ldots, x_n)$ называется двойственной к функции $f(x_1, \ldots, x_n)$, если $f^*(x_1, \ldots, x_n) = \neg f(\neg x_1, \ldots, \neg x_n)$ для всех значений переменных. Доказать, что двойственной к линейной (монотонной) функции снова будет линейная (соответственно, монотонная) функция. Определить, как ведёт себя двойственная функция f^*, если f сохраняет ноль или единицу. ▼

97. Найти число булевых функций от n переменных, являющихся одновременно самодвойственными и линейными. ▼

98. Найти число булевых функций от n переменных, являющихся одновременно монотонными и линейными. ▼

99. Доказать, что если $f(x_1, \ldots, x_n)$ — линейная функция, отличная от константы, то количество единиц в таблице истинности равно 2^{n-1}. Верно ли обратное? ▼

100. Найти все трёхместные булевы функции f, которые в одиночку образуют полное множество $\{f\}$. Найти их количество. ▼

Глава 6

Хорновские формулы и задача получения продукции

Краткое содержание: хорновские формулы, задача получения продукции, связь между задачей о следствии для хорновских формул и разрешимостью задачи о продукции, эффективные алгоритмы прямого поиска (поиска от данных) для решения задачи о продукции.

Ключевые слова: хорновская формула, задача о следствии для хорновских формул, задача о продукции, технологический процесс, замыкание множества продуктов посредством технологических процессов, алгоритм прямого поиска (поиска от данных).

§ 6.1. Хорновские формулы

Определим вначале один интересный класс булевых формул — хорновские формулы.

Определение 49 (Хорновская формула)**.** *Пусть* $\boldsymbol{V}$ *— это множество пропозициональных переменных. Хорновская формула — это формула вида*

$$(a_1 \wedge a_2 \wedge \cdots \wedge a_n) \to b,$$

где $a_1, \ldots, a_n, b \in \boldsymbol{V}$*.*

Содержательно, такая хорновская формула утверждает, что из истинности всех условий набора $\{a_1, a_2, \ldots, a_r\}$ следует истинность заключения b. Утверждения такого вида находят широкое применение в различных разделах информатики. В частности, в теории баз данных такой вид имеют «функциональные зависимости», в логическом программировании — правила логических программ, в автоматическом синтезе программ — «аксиомы вычислимости». В таком же виде формулируются правила вывода во многих экспертных системах.

Иногда бывает удобно считать, что в левой части импликации может стоять пустая конъюнкция, то есть хорновская формула может приобрести вид $\to b$. Так как пустая конъюнкция тождественно равна единице, то формула $\to b$ просто эквивалентна b.

В этих и других областях представляют интерес связанные между собой задачи о минимальности набора хорновских формул и о выводимости некоторой хорновской формулы из заданного набора хорновских формул. Первая задача состоит в выяснении того, входит ли в набор хорновских формул F некоторая формула φ, которая может быть удалена из F без потери информации, то есть которая следует из $F \setminus \{\varphi\}$. Уточним эту задачу.

Мы уже формулировали понятие следования одной формулы из другой. Теперь распространим это определение на множество формул.

Определение 50 (Следствие)**.** *Формула* φ *является следствием множества формул* F*, если во всякой интерпретации, в которой истинны все формулы из* F*, истинна и* φ *(будем это обозначать как* $F \Rightarrow \varphi$*).*

Следующее простое утверждение показывает, что понятие следования можно переформулировать в терминах тождественной истинности (см. определение 30 на стр. 74).

Предложение 36. *Хорновская формула φ является следствием конечного множества хорновских формул F тогда и только тогда, когда формула*

$$\Big(\bigwedge_{\psi \in F} \psi \Big) \to \varphi \tag{17}$$

является тождественно истинной.

Доказательство. Непосредственно следует из определения значений конъюнкции и импликации. □

Как уже отмечалось в параграфе 3.4 на стр. 68, проблема проверки по произвольной формуле её тождественной истинности является весьма сложной. Известный нам метод такой проверки с помощью построения таблицы значений на всех наборах переменных практически не работает уже для формул с несколькими десятками переменных. В то же время во многих практических задачах число логических параметров исчисляется сотнями. Оказывается, что для установления тождественной истинности формул вида (17) или, что то же самое, для задачи проверки условия $F \Rightarrow \varphi$ для хорновских формул имеются простые и очень эффективные алгоритмы, позволяющие её решать для формул с сотнями и тысячами переменных.

Мы изложим этот алгоритм на примере одного из интересных «экономико-хозяйственных» приложений хорновских формул — задачи о возможности производства заданной продукции (набора товаров) из некоторого множества исходных продуктов (товаров, сырья).

§ 6.2. Задача получения продукции

Пусть задано некоторое множество $A = \{a_1, a_2, \ldots, a_N\}$ имён товаров (продуктов, сырья и т. д.) и имеется некоторое множество F технологических процессов (производств), описывающих возможности получения одних продуктов из других. Каждый технологический процесс $t \in F$ задаётся множеством $L_t \subseteq A$ исходных

продуктов (входов) этого процесса и результирующим продуктом (выходом) $b_t \in A$, то есть процесс t позволяет из исходных продуктов L_t получить продукт b_t — его выход. Будем записывать такой технологический процесс в виде $t : L_t \to b_t$. Продукт, полученный в одном процессе, может далее использоваться в других процессах.

Определение 51 (Задача получения продукции)**.** *Задача получения продукции состоит в том, чтобы выяснить по заданному набору исходных продуктов $X \subseteq A$ и результирующему продукту $y \in A$, можно ли с помощью технологических процессов из F получить выход y по входным продуктам из X.*

Можно обобщить эту задачу и рассматривать возможность получения по X некоторого множества результирующих продуктов $Y \subseteq A$.

Пример 21. *Пусть*

$$A = \{\textit{дерево, клей, гвозди, кирпич, стекло,}\\ \textit{окна, полы, стены, крыша, столы, дача}\}.$$

Множество технологических процессов $F = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ задаётся соответствующими множествами входов и выходов:

$$t_1 : \{\textit{дерево, клей, гвозди}\} \to \textit{столы}$$
$$t_2 : \{\textit{дерево, гвозди}\} \to \textit{полы}$$
$$t_3 : \{\textit{дерево, клей, стекло}\} \to \textit{окна}$$
$$t_4 : \{\textit{стены, полы, крыша}\} \to \textit{дача}$$
$$t_5 : \{\textit{кирпич, окна, дерево}\} \to \textit{стены}$$
$$t_6 : \{\textit{дерево, гвозди}\} \to \textit{столы}$$

Рассмотрим для этой системы технологических процессов задачу получения продукта «дача» по исходному множеству продуктов:

$$\{\textit{дерево, клей, гвозди, стекло, кирпич, крыша}\}.$$

Нетрудно понять, что эта задача решается положительно с помощью следующей цепочки процессов: $(t_3; t_5; t_2; t_4)$. Действительно, в t_3 получаются «окна», которые используются в t_5 для получения «стен», в t_2 производятся «полы», а затем произведённые ранее «стены», «полы», «крыша» используются в t_4 для получения результата «дача».

Подчеркнём, что мы абстрагируемся от количественных оценок исходных и производимых продуктов и считаем, что они всегда даются на входе и производятся в количестве, достаточном для обеспечения «сырьём» всех запускаемых процессов.

Построим формальную модель задачи о производстве с помощью булевых формул.

Будем рассматривать A как множество булевых переменных. Каждому процессу t с параметрами $L_t = \{a_1, \ldots, a_r\}$ и b_t сопоставим следующую хорновскую формулу $\Phi(t)$:

$$(a_1 \wedge a_2 \wedge \cdots \wedge a_r) \to b_t.$$

Например, процессу t_5 из нашего примера соответствует формула $\Phi(t_5)$:

$$(\text{кирпич} \wedge \text{окна} \wedge \text{дерево}) \to \text{стены}.$$

Сохраним для множества хорновских формул, соответствующих процессам, обозначение F.

Справедлива следующая теорема, которая показывает, что задача о возможности получения продукции и задача о следствии из множества хорновских формул эквивалентны.

Теорема 37. *Для любых множества продуктов A, множества технологических процессов F, множества исходных продуктов $X \subseteq A$ и результирующего продукта $y \in A$ задача получения продукта y по входным продуктам из X с помощью процессов из F разрешима тогда и только тогда, когда $F \Rightarrow \varphi_X \to y$, где $\varphi_X = \bigwedge\limits_{a \in X} a$.*

Доказательство. Сначала докажем теорему в прямую сторону. Предположим, что с помощью набора процессов из множества $F = \{t_1, \ldots, t_h\}$ из множества исходных продуктов X можно получить y. Пусть $\tau = (t_{i_1}, \ldots, t_{i_m})$ — это последовательность процессов из F, которая приводит к получению y. Докажем, что тогда $F \Rightarrow \varphi_X \to y$. Рассмотрим произвольную интерпретацию I переменных $a_1, \ldots, a_n$, в которой истинны все формулы из F. Если хотя бы для одной переменной $a_j \in X$ будет $I(a_j) = 0$, то формула $\varphi_X \to y$ истинна, поскольку

её левая часть ложна. Предположим теперь, что $I(a_j) = 1$ для любой переменной $a_j \in X$.

Тогда индукцией по номеру r процесса t_{i_r} в τ покажем, что для каждого $r = 1, \ldots, m$ значение соответствующей результирующей переменной $\sigma(b_{i_r}) = 1$.

Базис индукции. При $r = 1$ из применимости процесса t_{i_1} следует, что $L_{i_1} \subseteq X$, но тогда $I(a) = 1$ для любой переменной $a \in L_{i_1}$ и левая часть импликации $\Phi(t_{i_1})$ истинна в I. Но так как и вся формула $\Phi(t_{i_1})$ истинна в I, то и её заключение b_{i_1} тоже истинно в I, то есть $I(b_{i_1}) = 1$.

Индукционный шаг. Пусть теперь для некоторого $k > 1$ при всех $r < k$ выполнено $I(b_{i_r}) = 1$. Докажем, что $I(b_{i_k}) = 1$. Поскольку процесс t_{i_k} применим после процессов $t_{i_1}, \ldots, t_{i_{k-1}}$, то $L_{i_k} \subseteq X \cup \{b_{i_r} : r = 1, \ldots, k-1\}$. Тогда все переменные из L_{i_k} истинны в I и, следовательно, $I(b_{i_k}) = 1$.

Из доказанного утверждения следует, что $I(b_{i_m}) = 1$. Но так как последовательность τ приводит к выпуску y, то $b_{i_m} = y$ и, следовательно, $I(y) = 1$. Таким образом, формула $\varphi_X \to y$ истинна в I и условие $F \Rightarrow \varphi_X \to y$ выполнено.

Теперь докажем теорему в обратную сторону. Предположим, выполнено условие $F \Rightarrow \varphi_X \to y$. Опишем построение последовательности процессов τ, которая приведёт к производству y. Эта последовательность будет строиться по шагам. На шаге i вместе с последовательностью τ_i будем определять множество продуктов X_i, которые можно произвести, исходя из X с помощью τ_i. Процедура построения последовательности τ_i завершается, как только в неё включается некоторый процесс с результатом y, либо когда на очередном этапе в X_i не добавляются новые элементы.

Шаг 0: положим $\tau_0 = \varnothing$, $X_0 = X$.

Шаг 1: положим

$$\tau_1 = (t \in F : L_t \subseteq X_0); \qquad X_1 = X_0 \cup \{b_t : t \in \tau_1\}.$$

Шаг $i + 1$: пусть уже определены τ_i и X_i. Положим

$$\tau'_{i+1} = (t \in (F \setminus \tau_i) : L_t \subseteq X_i); \qquad X_{i+1} = X_i \cup \{b_t : t \in \tau'_{i+1}\}$$

и $\tau_{i+1} = \tau_i \tau'_{i+1}$, процессы внутри τ'_{i+1} упорядочиваются в произвольном порядке. Если $y \in X_{i+1}$ или $X_i = X_{i+1}$, то положим $\tau = \tau_{i+1}$ и закончим процедуру.

Заметим вначале, что эта процедура построения τ обязательно завершится через конечное число шагов, так как размер X_i не может превысить размер множества всех продуктов A. Покажем, что процесс построения τ завершится на таком шаге $i+1$, для которого впервые $y \in X_{i+1}$, то есть что последовательность процессов τ приводит к производству y. Действительно, предположим, что процедура завершилась после этапа $i+1$ из-за выполнения равенства $X_i = X_{i+1}$, при этом $y \notin X_{i+1}$. Покажем, что тогда существует интерпретация I, в которой все формулы из F истинны, а формула $\varphi_X \to y$ ложна. Положим $I(a) = 1$ при $a \in X_{i+1}$ и $I(a) = 0$ при $a \notin X_{i+1}$. Так как $X \subseteq X_0 \subseteq X_{i+1}$, то для каждого $a \in X$ значение $I(a) = 1$, а так как $y \notin X_{i+1}$, то $I(y) = 0$, то есть формула $\varphi_X \to y$ в интерпретации I ложна. Каждая формула $\Phi(t)$ для $t \in \tau_{i+1}$ истинна, поскольку $b_t \in X_{i+1}$ и, следовательно, $\sigma(b_t) = 1$. Ложной могла бы оказаться лишь формула $\Phi(t)$ для такого процесса $t \in F$, что $b_t \notin X_{i+1}$. Но для такого процесса t обязательно имеется продукт $a \in L_t$, который не входит в X_i (иначе бы b_t попало в X_{i+1} и процедура не остановилась бы на шаге $i+1$). Значение этого a равно нулю в I. Но тогда условие импликации $\Phi(t)$ ложно в I, а вся формула $\Phi(t)$ на нём истинна. Таким образом, мы пришли к противоречию, которое показывает, что $y \in X_{i+1}$ и процесс τ приводит к производству y. □

§ 6.3. Решение задачи о продукции

Можно ли построить эффективную процедуру, проверяющую разрешимость задачи о продукции или, что то же самое, задачи о следствии для хорновских формул? Процедура, описанная во второй части доказательства теоремы 37 на стр. 122, является основой для следующего алгоритма, который строит множество всех продуктов, которые можно получить из исходных продуктов с помощью заданной системы технологических процессов.

Определение 52 (Замыкание)**.** *Пусть F — множество технологических процессов, X — исходное множество продуктов. Замыкание X с помощью F — это множество продуктов*

$$\mathrm{cl}(X, F) = \{y : y \in A \text{ и } F \Rightarrow \varphi_X \to y\}.$$

Приведённый на рис. 13 на следующей странице алгоритм «Прямая волна» решает задачу о продукции в два этапа: на первом строится замыкание $\mathrm{cl}(X, F)$, а на втором — проверяется, входит ли y в это замыкание. Основную роль играет алгоритм построения замыкания Замыкание(X, F). В нём переменные S и N — это множества «старых» и «новых» продуктов (истинных булевых переменных). Идея построения замыкания проста: вначале исходные данные из X заносятся в N. Затем работа идёт поэтапно. Перед началом каждого этапа полученные ранее N передаются в S. После этого результаты всех технологических процессов, входы которых уже получены, то есть содержатся в множестве N, добавляются к этому множеству. Алгоритм завершает работу, когда на очередном этапе ничего не добавилось, то есть нет технологических процессов, применение которых может расширить множество полученных продуктов.

Алгоритмы такого вида часто называются алгоритмами прямого поиска или поиска от данных.

Докажем корректность приведённого алгоритма.

Теорема 38. *Алгоритм* Замыкание(X, F) *возвращает множество* $\mathrm{cl}(X, F)$*, а алгоритм* ПрямаяВолна(X, y, F) *выдаёт ответ «да» тогда и только тогда, когда* $F \Rightarrow \varphi_X \to y$*.*

Доказательство. Доказательство этого утверждения фактически содержится в комментариях перед алгоритмом и его уточнение предоставляется читателю (см. задачу 106 на стр. 131). □

Пример 22. *Рассмотрим работу алгоритма* Замыкание(X, F) *на задаче получения $y =$ «дача» по исходному множеству*

$$X = \{\textit{дерево, клей, гвозди, стекло, кирпич, крыша}\}$$

из примера 21 на стр. 121.

```
1: Алгоритм Замыкание(X, F)    # Замыкание X с помощью F
2:     Вход: X — множество исходных продуктов
3:     Вход: F — множество технологических процессов
4:     Выход: N — множество продуктов, N = cl(F, X)
5:     S ← ∅
6:     N ← X
7:     Пока N ≠ S выполнять
8:         S ← N
9:         Для всех t ∈ F выполнять
10:            Если L_t ⊆ N то
11:                N ← N ∪ {b_t}
12:            Конец Если
13:        Конец Для
14:    Конец Пока
15: Конец Алгоритм

16: Алгоритм ПрямаяВолна(X, y, F)# Возможность получения y
17:    Вход: X — множество исходных продуктов
18:    Вход: y — требуемый продукт
19:    Вход: F — множество процессов
20:    Z ← Замыкание (X, F)
21:    Если y ∈ Z то
22:        Вернуть «да»
23:    Иначе
24:        Вернуть «нет»
25:    Конец Если
26: Конец Алгоритм
```

Рис. 13: Алгоритм «Прямая волна»

Стр.	S	N
6	$\varnothing$	дерево, клей, гвозди, кирпич, крыша, стекло
8	дерево, клей, гвозди, кирпич, крыша, стекло	дерево, клей, гвозди, кирпич, крыша, стекло
11(t_1)	*(не меняется)*	дерево, клей, гвозди, кирпич, крыша, стекло, столы
11(t_2)	*(не меняется)*	дерево, клей, гвозди, кирпич, крыша, стекло, столы, полы
11(t_3)	*(не меняется)*	дерево, клей, гвозди, кирпич, крыша, стекло, столы, полы, окна
11(t_5)	*(не меняется)*	дерево, клей, гвозди, кирпич, крыша, стекло, столы, полы, окна, стены
8	дерево, клей, гвозди, кирпич, крыша, стекло, столы, полы, окна, стены	дерево, клей, гвозди, кирпич, крыша, стекло, столы, полы, окна, стены
11(t_4)	*(не меняется)*	дерево, клей, гвозди, кирпич, крыша, стекло, столы, полы,окна, стены, дача
8	дерево, клей, гвозди, кирпич, крыша, стекло, столы, полы, окна,стены, дача	дерево, клей, гвозди, кирпич, крыша, стекло, столы, полы, окна, стены, дача

Рис. 14: Пошаговая работа алгоритма из примера 22.

В таблице на рис. 14 показаны шаги алгоритма, на которых изменяются значения переменных S и N. В первом столбце этой таблицы указан номер соответствующей строки алгоритма, после строки 11 в скобках указан тот процесс, который приводит к увеличению множества N.

Таким образом, в данном примере построение результирующего множества N, равного $\mathrm{cl}(X, F)$, *завершается за два этапа, а на третьем выясняется, что ничего нового в него не может быть добавлено. После этого алгоритм* ПрямаяВолна (X, y, F) *проверит, что «дача» входит в* $\mathrm{cl}(X, F)$, *и выдаст ответ «да».*

С точки зрения сложности вычислений недостатком алгоритма Замыкание является то, что на каждой итерации основного цик-

ла (строки 7–14) в строке 9 перебираются все процессы, даже уже отработавшие, а в строке 10 на вхождение в N проверяются все элементы L_t, даже те, вхождение которых в N уже было установлено на предыдущих итерациях. На рис. 15 на следующей странице приведён более эффективный алгоритм для построения замыкания.[1]

На этапе инициализации в нём для каждого процесса t подсчитывается число элементов в L_t и помещается в ячейку массива $C[t]$, кроме того, для каждого $a \in \boldsymbol{V}$ создаётся список $L[a]$ процессов, в левые части которых входит продукт a. В основной части алгоритма все полученные продукты, как и в алгоритме Замыкание, собираются во множестве N. При этом перед началом очередного этапа множество A содержит новые продукты, полученные на предыдущем этапе. В начале $N = A = X$. Выполнение этапа состоит в том, что для каждого продукта a из множества A и каждого процесса t, в условие которого входит a, из $C[t]$ вычитается 1. Для поиска таких t используется $L[a]$. Если $C[t]$ становится равным нулю, это означает, что все продукты из L_t уже получены и тогда его результат b_t добавляется в N, если его там ранее не было, а также временно запоминается во множестве D. По окончании этапа продукты из этого множества переносятся во множество A. Множества продуктов N, A и D реализуются двоичными массивами длины $|\boldsymbol{V}|$, единицы которых объединены в списки.

Теорема 39. *Алгоритм* ОптЗам (X, F) *строит замыкание* $\mathrm{cl}(X, F)$. Доказательство. Доказательство этой теоремы проведите самостоятельно (см. задачу 112 на стр. 132). □

Отметим, что фактически общее количество шагов алгоритма ОптЗам (X, F) пропорционально размеру его входа, то есть числу продуктов в F и X или числу букв в записи формулы (17). Такие алгоритмы называются работающими в линейное от размера входа

[1] Этот алгоритм был независимо открыт несколькими авторами, рассматривающими различные эквивалентные формулировки в различных областях: в синтезе программ — А. Диковским (1979), в базах данных — К. Бирном и П. Бернстайном (1979), в логическом программировании — В. Доулингом и Дж. Гальером (1984). Мы, в основном, следуем изложению этого алгоритма в [13].

```
1: Алгоритм ОптЗам(X, F)          # Замыкание X с помощью F
2:     Вход: X — множество исходных продуктов
3:     Вход: F — множество технологических процессов
4:     Выход: N — множество продуктов, N = cl(F, X)
5:     Создать массивы C[F] и L[V]
6:     Для всех a ∈ V выполнять
7:         Создать список L[a]
8:     Конец Для
9:     Для всех t ∈ F выполнять
10:        C[t] ← |L_t|
11:        Для всех a ∈ L_t выполнять
12:            L[a] ← L[a] ∪ {t}
13:        Конец Для
14:    Конец Для
15:    N ← X
16:    A ← X
17:    Пока A ≠ ∅ выполнять
18:        D ← ∅
19:        Для всех a ∈ A выполнять
20:            Для всех t ∈ L[a] выполнять
21:                C[t] ← C[t] − 1
22:                Если C[t] = 0 и b_t ∉ N то
23:                    N ← N ∪ {b_t}
24:                    D ← D ∪ {b_t}
25:                Конец Если
26:            Конец Для
27:        Конец Для
28:        A ← D
29:    Конец Пока
30: Конец Алгоритм
```

Рис. 15: Оптимизированный алгоритм замыкания

время или, просто, линейными. Действительно, при инициализации каждый элемент F рассматривается два раза, а в основном цикле общее число рассматриваемых элементов и операций уменьшения на единицу значений $C[t]$ в строке 21 алгоритма не больше суммы размеров всех L_t, то есть также не превосходит размера входа.

Пример 23. *Рассмотрим работу алгоритма* ОптЗам *на следующем примере. Пусть* $A = \{a, b, c, d, e, f, g, h\}$, $X = \{b, f\}$, *а множество* F *состоит из следующих шести процессов:*

$$\begin{array}{lll} (1)\quad a, b, c, h \to d; & (3)\quad g, b \to e; & (5)\quad f, e \to d; \\ (2)\quad b, c, d \to a; & (4)\quad e, f \to c; & (6)\quad b, f \to g. \end{array}$$

Тогда при инициализации будут построен массив $C = [4, 3, 2, 2, 2, 2]$ *и следующие списки:*

$$\begin{array}{lll} L[a] = (1); & L[d] = (2); & L[g] = (3); \\ L[b] = (1, 2, 3, 6); & L[e] = (4, 5); & L[h] = (1). \\ L[c] = (1, 2); & L[f] = (4, 5, 6); & \end{array}$$

Множества A *и* N *будут инициализированы в строках 15–16 массивами* 01000100 *с единицами на местах, соответствующих продуктам* b *и* f. *Дальнейшие изменения этих структур на этапах алгоритма представлены в следующей таблице:*

C						A	N
(1)	(2)	(3)	(4)	(5)	(6)	$abcdefgh$	$abcdefgh$
4	3	2	2	2	2	01000100	01000100
3	2	1	1	1	0	00000010	01000110
3	2	0	1	1	0	00001000	01001110
3	2	0	0	0	0	00110000	01111110
2	0	0	0	0	0	10000000	11111110
1	0	0	0	0	0	00000000	11111110

Алгоритм завершает работу, когда множество A *становится пустым. В этот момент результат* $\mathrm{cl}(X, F)$ *представлен множеством* N. *В нашем примере оно равно* $\{a, b, c, d, e, f, g\}$.

Замечание 7. Как мы специально подчёркивали, при рассмотрении задачи о продукции не принимались во внимание количественные характеристики, то есть какое именно количество исходных продуктов нужно для

получения единицы производимого. Мы говорили только о возможности получения, если исходные продукты гарантированно имеются в достаточном количестве.

К сожалению, если ввести это ограничение и учитывать количества исходных и получаемых продуктов, то задача опять превращается в вычислительно очень сложную. Никаких существенно лучших методов её решения, кроме перебора всех возможных вариантов цепочек процессов неизвестно, и предполагается, что их нет.

Задачи

101. Доказать, что каждое множество хорновских формул имеет модель, то есть интерпретацию, в которой все они истинны. ▼

102. Доказать, что если КНФ в каждой элементарной конъюнкции содержит в точностью одну переменную без отрицания, то она эквивалентна конъюнкции хорновских формул. ▼

103. Пусть I и J — интерпретации. Назовём их пересечением интерпретацию $K = IJ$ такую, что $K(x) = 1$ тогда и только тогда, когда $I(x) = J(x) = 1$ для всех переменных x. Доказать, что если хорновская формула имеет модели I и J, то их пересечение $K = IJ$ тоже будет моделью. ▼

104. Доказать, что формула Ф эквивалентна конъюнкции хорновских формул тогда и только тогда, когда её сокращённая КНФ содержит в каждой элементарной дизъюнкции в точности одну переменную без отрицания. ▼

105. Доказать, что последовательность процессов τ_i в доказательстве теоремы 37 на стр. 122 определена корректно, то есть все исходные продукты каждого процесса в этой последовательности имеются перед его запуском. ▼

106. Доказать теорему 38 на стр. 125. (Указание. Пусть N_k — это значение N после k итераций основного цикла алгоритма Замыкание в строках 7–14. Показать, что для каждого продукта $z \in \mathrm{cl}(X, F)$, который может быть получен из X цепочкой процессов длины k и менее, z входит в N_k.) ▼

107. Алгоритм ПрямаяВолна (X, y, F) позволяет ответить на вопрос о возможности производства y из исходных продуктов X с помощью процессов F, но не строит цепочку процессов, приводящую к y. Изменить алгоритм Замыкание (X, F) так, чтобы по его результату для любого продукта $a \in \mathrm{cl}(X, F)$ можно было построить цепочку процессов, приводящую к a. ▼

108. Назовём *сложным технологическим процессом* такой процесс t, который по набору исходных продуктов L_t одновременно производит некоторое множество продуктов B_t (а не один продукт b_t). Доказать, что сложному технологическому процессу соответствует конъюнкция хорновских формул. ▼

109. Обобщить алгоритм Замыкание (X, F) так, чтобы он строил замыкание X относительно системы сложных технологических процессов F. ▼

110. Определить, какая цепочка процессов в примере 23 на стр. 130 приводит к получению a. ▼

111. Используя алгоритм Замыкание, вычислить замыкание для набора исходных продуктов $X = \{c, d\}$ и системы технологических процессов F:

$$\begin{array}{rll} a, b, d \to h & e, f \to c & h, d, c \to g \\ a, c, d, g \to f & b, k \to a & d, g, a \to e \\ d, g \to b & d, c \to k & c, d, k \to h \end{array}$$

Определить, какая цепочка процессов приводит к получению e. ▼

112. Доказать теорему 39 на стр. 128. (Указание. Пусть N_k — это значение N, A_k — значение A, а C_k — значение C после k итераций основного цикла алгоритма ОптЗам в строках 17–29. Показать, что

(а) $C_k[t] = |L_t \setminus (N_k \setminus A_k)|$;

(б) для каждого продукта $z \in \mathrm{cl}(X, F)$, который может быть получен из X цепочкой процессов длины не более k, $z \in N_k$;

(в) условие $A = \varnothing$ выхода из основного цикла выполнено после $(k+1)$-й итерации тогда и только тогда, когда $N_k = N_{k+1}$.) ▼

113. Изменить алгоритм ОптЗам таким образом, чтобы по его результату для любого продукта $a \in \mathrm{cl}(X, F)$ можно было построить цепочку процессов, приводящую к a. ▼

114. Используя алгоритм ОптЗам, вычислить замыкание для набора исходных атрибутов $X = \{a, f\}$ и следующей системы зависимостей F:

$$\begin{array}{rll} a, b, c \to h & e, f \to c & g, d \to e \\ a, c, d, g \to h & f, a \to d & d, f, a \to g \end{array}$$

Определить, какая цепочка процессов приводит к получению h. ▼

Глава 7

Язык логики предикатов

Краткое содержание: объекты, их свойства, отношения между объектами, утверждения о свойствах объектов и отношениях между ними, предикаты, синтаксис и семантика логики предикатов, интерпретации, значения формул в интерпретациях, эквивалентности логики предикатов, предварённые формулы.

Ключевые слова: логика предикатов, предикат, квантор существования, квантор всеобщности, атомная формула, формула логики предикатов, свободные и связанные переменные, интерпретация, значение формулы, тождественно истинная формула, предварённая формула.

§ 7.1. Утверждения о свойствах объектов и отношениях между ними

Утверждения, которые можно сформулировать с помощью средств логики высказываний всегда относятся к конкретным свойствам конкретных предметов, объектов, ситуаций. Мы уже сталкивались с примерами таких утверждений: «Сегодня идёт дождь», «Вася любит Олю», «Если A — преступник, то B невиновен», «Если цена на нефть

растёт и страна продаёт нефть, то растут и доходы её бюджета» и т. д. Эти утверждения строятся из элементарных высказываний (пропозициональных переменных) с помощью логических связок (операций). Например, последнее из приведённых утверждений может быть формально записано как $(x \wedge y) \to z$, где x, y и z — это переменные, соответственно обозначающие высказывания «цена на нефть растёт», «страна продаёт нефть» и «растут доходы бюджета».

Но можно заметить, что эти высказывания хотя и являются элементарными, то есть не содержат внутри себя других высказываний, но всё же имеют внутреннюю структуру. Например, высказывание «Вася любит Олю» означает, что два объекта «Вася» и «Оля» связаны между собой чем-то, что именуется «любит». Поскольку «Вася», «Оля» и «любит» высказываниями уже не являются, то разобрать строение такого предложения с помощи логики высказываний нельзя.

Для того, чтобы точнее описывать взаимоотношения между объектами нужен более богатый формальный язык. Им является язык *логики предикатов* (или *логики первого порядка*).

Прежде, чем переходить к формальным определениям, приведём некоторые содержательные примеры используемых понятий: объектов, отношений, свойств и операций (функций).

Объектами могут быть люди, предприятия, числа, цвета, футбольные матчи, экзамены, дома, столы, компьютеры, фигуры...

Отношениями являются: x является братом y, x занимает должность y с зарплатой z, x больше чем y, x находится внутри y, x любит y, x имеет цвет y, x случится после y, x владеет y и т. д. С помощью x, y, z обозначены места, куда можно подставить конкретные объекты, чтобы получить высказывание. Например, с помощью первого отношения можно построить высказывания «Коля является братом Оли» и «Оля является братом Коли». Подобные языковые конструкции, которые можно заполнять конкретными названиями, получая высказывания, и называют *предикатами*. С математической точки зрения термины «предикат» и «отношение» — синонимы.

Частным случаем отношений являются с в о й с т в а, когда имеется в точности одно «вакантное место»: x зелёный, x весит более 5 кг, x девятиэтажный, x — мужчина и т. д.

О п е р а ц и и являются способом указать по одним объектам другие: отец x, брат x, дом, в котором живёт x...

Даже этих небольших перечислений достаточно, чтобы понять, что язык логики предикатов позволяет описать почти любой факт и сформулировать нужное утверждение о той или иной рассматриваемой предметной области. Вот несколько простых примеров:

1) «Два умножить на три равно шесть»; здесь объекты: два, три и шесть; функция: умножить, отношение: равно.

2) «Отцы всех бармаглотов живут в зелёных домах»; объекты: бармаглоты, дома; свойство: зелёный; отношение: «жить в»; функция «отец».

3) «Некоторые предприятия в Твери являются банкротами»; объекты: Тверь, предприятия; свойство: банкрот; отношение: «быть» («являться»).

С формализацией этих понятий мы уже знакомы: объекты — это элементы каких-то множеств (параграф 1.1); отношения — подмножества декартовых произведений (определения 8 на стр. 27 и 17 на стр. 31); а функции являются частным случаем отношений (определения 14 на стр. 30 и 17).

В определениях формальных языков, таких, как логические языки или языки программирования, выделяют два основных аспекта: синтаксис и семантику. С и н т а к с и с определяет алфавит, из символов которого строятся выражения языка, и правила, выделяющие из множества всех слов в данном алфавите п р а в и л ь н о п о с т р о е н н ы е в ы р а ж е н и я. Для логических языков такие выражения обычно называются формулами, а для языков программирования — программами. С е м а н т и к а занимается определением смысла, значения этих выражений. Для формул большинства логических языков значением является одно из истинностных значений: 1 (истина) или 0

(ложь), которое приписывается формуле в зависимости от интерпретации входящих в неё символов языка. Например, значения формул логики высказываний зависят от значений входящих в них булевых переменных. Значения формул определяемой в этом параграфе логики предикатов будут зависеть от интерпретации предикатов и функций языка, а также от значений входящих в эти формулы переменных.

§ 7.2. Синтаксис логики предикатов

Прежде чем начать строить формулы логики предикатов, нам потребуется определить алфавит, из которого будем строить формулы. Он будет состоять из двух частей: фиксированной — логических символов, и переменной — сигнатуры.

Логическими символами называют элементы следующего фиксированного множества (кавычки поставлены для удобства):

$$\boldsymbol{L} = \{\text{«}\approx\text{»},\ \text{«,»},\ \text{«(»},\ \text{«)»},\ \text{«}\neg\text{»},\ \text{«}\wedge\text{»},\ \text{«}\vee\text{»},\ \text{«}\to\text{»},\ \text{«}\exists\text{»},\ \text{«}\forall\text{»}\}.$$

Определение 53 (Сигнатура)**.** *Сигнатурой называется пара вида* $(\boldsymbol{P}, \boldsymbol{V})$*. Здесь* $\boldsymbol{P}$ *— это множество имён отношений (или, говорят, предикатов), для каждого из которых указана некоторая местность — натуральное число, а* $\boldsymbol{V}$ *— счётно бесконечное множество имён для обозначения объектов.*

Имена отношений называют предикатными символами, а имена объектов — предметными переменными. Местность предикатных символов мы будем обозначать с помощью верхнего индекса в скобках: если $A^{(3)} \in \boldsymbol{P}$*, то* A *— трёхместный предикатный символ.*

Мы уже неоднократно отмечали, что функции являются частным случаем отношений, поэтому специальных символов для обозначений функций мы вводить не будем, чтобы не усложнять изложение. Например, вместо функции сложения чисел, мы будем использовать трёхместное отношение $A(x, y, z)$, означающее, что $x + y = z$.

Определение 54 (Формула логики предикатов)**.** *Пусть зафиксирована некоторая сигнатура $\Sigma = (\boldsymbol{P}, \boldsymbol{V})$. Формулой сигнатуры Σ называют слово в алфавите $\boldsymbol{L} \cup \boldsymbol{P} \cup \boldsymbol{V}$, определяемое по индукции.*

Формулами являются

1) $x \approx y$, *где* $x, y \in \boldsymbol{V}$;

2) $R(x_1, \ldots, x_n)$, *где* $x_1, \ldots, x_n \in \boldsymbol{V}$, $R^{(n)} \in \boldsymbol{P}$.

Формулы двух перечисленных выше видов называются а т о м н ы м и. *Они образуют базис индукционного определения.*

Пусть теперь уже построены формулы Φ и Ψ, тогда следующие слова тоже являются формулами:

3) $\neg\Phi$;

4) $(\Phi \wedge \Psi)$;

5) $(\Phi \vee \Psi)$;

6) $(\Phi \to \Psi)$;

7) $(\exists x)\Phi$, *где* $x \in \boldsymbol{V}$;

8) $(\forall x)\Phi$, *где* $x \in \boldsymbol{V}$.

Для «традиционных» отношений в формулах часто применяется инфиксная запись, например, вместо $<(x, y)$ пишут $x < y$, а вместо $\in (x, y)$ — $x \in y$ и т. д.

Строка $(\exists x)$ называется к в а н т о р о м с у щ е с т в о в а н и я по переменной x, а строка $(\forall x)$ — к в а н т о р о м в с е о б щ н о с т и по x.

Каждое вхождение предметной переменной в формулу является либо свободным, либо связанным.

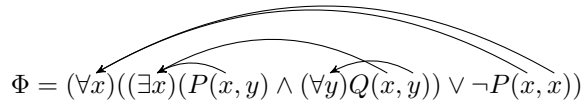
Определение 55 (Свободные и связанные переменные)**.** *Каждое вхождение любой переменной в атомную формулу является* с в о б о д н ы м.

В пунктах 3)–6) все вхождения переменных остаются свободными или связанными, в соответствии с тем, какими они были в исходных формулах Φ и Ψ.

В пунктах 7) и 8) все свободные вхождения переменной x в формулу Φ становятся с в я з а н н ы м и н а ч а л ь н ы м к в а н т о р о м $(\exists x)$ *или* $(\forall x)$ *соответственно. Все остальные вхождения переменной x и все вхождения остальных переменных остаются свободными или связанными, в соответствии с тем, какими они были в исход-*

ной формуле Φ. *Формула* Φ *в этом случае называется* о б л а с т ь ю д е й с т в и я к в а н т о р а.

Пример 24. *Пусть в некоторой сигнатуре имеются два двухместных предикатных символа* $P^{(2)}, Q^{(2)}$. *Тогда в формуле*



оба вхождения x *в* $(P(x,y) \wedge (\forall y)Q(x,y))$ *связаны квантором* $(\exists x)$, *первое вхождение* y *является свободным, а второе — связано квантором* $(\forall y)$. *Оба вхождения* x *в* $\neg P(x,x)$ *связаны квантором* $(\forall x)$.

В приведённом примере продемонстрировано, что разные вхождения одной и той же переменной могут быть связаны разными кванторами, а могут быть свободными. Если, забегая вперёд, говорить неформально, то такие вхождения могут обозначать разные объекты, несмотря на использование одной переменной.

С помощью формул логики предикатов можно записывать фразы «естественных» языков, подобно тому, как мы это делали в логике высказываний (параграф 3.4 на стр. 68). Рассмотрим основные принципы, с помощью которых те или иные предложения можно формализовать с помощью формул.

Как правило, каждое предложение содержит сказуемое, которое и является предикатом. Подлежащее и дополнения чаще всего являются его аргументами. Например, в предложении «Вася любит Олю» предикатом будет «любит», а его аргументами — «Вася» и «Оля». Таким образом, для формализации этой фразы мы должны ввести в сигнатуру двухместное отношение $L^{(2)}$ (первый аргумент — кто любит, второй — кого) и две предметные переменные v и o для обозначения вышеупомянутых лиц. Формула, соответствующая приведённой фразе, имеет вид $L(v,o)$.

Обстоятельства тоже могут быть аргументами предиката, а могут быть его частью. Здесь однозначных правил нет. Если эти обстоятельства сами по себе могут состоять в каких-то отношениях, то это — аргументы, если они лишь комментируют (уточняют) смысл, то

являются частью предиката. В последнем случае частью предиката могут быть и дополнения. Например, в предложении «Вася гулял в лесу» обстоятельство «в лесу» может рассматриваться как часть одноместного предиката «гулял в лесу», а может — как элемент двухместного предиката «гулял» и связывающего объекты «Вася» и «лес». В первом случае формула будет иметь вид $G(v)$, во втором — $G(v, \ell)$, где ℓ — предметная переменная, обозначающая «лес».

Определения обычно являются свойствами, то есть одноместными отношениями. Скажем, в предложении «Красный карандаш стоит дороже зелёного» упомянуто два объекта, обозначим их переменными x и y, которые обладают различными свойствами «быть красным» и «быть зелёным» (обозначим эти отношения символами $K^{(1)}$ и $Z^{(1)}$) и связаны двухместным предикатом «стоить дороже» — $D^{(2)}$. Поэтому формула выглядит так $K(x) \wedge Z(y) \wedge D(x, y)$.

В некоторых случаях определения тоже выражают двухместное отношение между основным и зависимым словами. Часто это происходит с притяжательными формами, означающими в том или ином виде принадлежность. Например, в предложении «Отец Петра работает шофёром» подлежащее «отец» является отношением $F^{(2)}$ между двумя людьми, первый из которых явно не назван, а вторым является определение «Пётр» (обозначим его переменной p). Второе отношение — это свойство $R^{(1)}$ «работать шофёром». Поскольку явно отец Петра в предложении не назван, для построения формулы придётся употребить квантор существования: $(\exists x)(F(x, p) \wedge R(x))$.

Разобранный пример, когда тот или иной объект упоминается неявно, через его зависимости, показывает, что это — один из случаев, когда появляются кванторы. Как нетрудно понять, квантор существования имеет смысл «существует», «имеется», «некоторый», «какой-то», «есть» и т. д. Например, формула, которую мы построили, говорит нам, что существует x, который является отцом Петра и работает шофёром.

Заметим, что при составлении формул стремятся сократить количество используемых отношений. Например, при описании родственных связей между людьми может понадобиться огромное число

предикатных символов, чтобы обозначить отца, мать, сына, дочь, внука, дядю и т. д. Однако все их можно выразить, используя всего три отношения: «быть родителем», «состоять в браке с» и «быть мужчиной». Будем считать, что для этого в сигнатуру включены символы $P^{(2)}$, $S^{(2)}$ и $M^{(1)}$. Тогда та же фраза может быть записана в виде формулы $(\exists x)(P(x,p) \wedge M(x) \wedge R(x))$, то вместо «отец» мы использовали «родитель-мужчина».

Квантор всеобщности появляется, когда используются слова «все», «всякий», «каждый» и т. д. Например, чтобы построить формулу для предложения «Все дочери Петра учатся в школе», кроме указанных выше нам нужен одноместный символ $U^{(1)}$ — учиться в школе. Тогда формула примет вид $(\forall x)(P(p,x) \wedge \neg M(x) \to U(x))$, то есть каждый x, если он является ребёнком Петра и не мужчиной, то учится в школе.

Осторожность нужно проявлять со словами «любой», «произвольный» и им подобными. В зависимости от контекста они могут передаваться либо тем, либо другим квантором. Например, в предложении «Каждому родителю полагаются льготы, если любой его ребёнок учится в школе» условие «если любой его ребёнок учится в школе» можно истолковать двояко. Можно так: «хотя бы один из детей», тогда формула будет иметь вид

$$(\forall x)((\exists y)(P(x,y) \wedge U(y)) \to L(x)),$$

с помощью L обозначен одноместный предикат «иметь право на льготы». А можно и по другому: «все дети», тогда формула станет такой

$$(\forall x)((\forall y)(P(x,y) \to U(y)) \to L(x)).$$

Обратим внимание, что с квантором существования мы употребили конъюнкцию, а с квантором всеобщности — импликацию. Это не случайно, многие обороты естественной речи имеют именно такую структуру: фраза «всякий A является B» соответствует формуле типа $(\forall x)(A(x) \to B(x))$, а фраза «существует A такой, что B» —

формуле $(\exists x)(A(x) \wedge B(x))$. Вместо свойств A и B, разумеется, могут стоять и другие формулы.

Равенство как правило соответствует словам «тот же», «этот же», «равный», «совпадающий» и т. д. Неравенство (то есть отрицание равенства, $\neg x \approx y$), может быть передано с помощью местоимений «другой», «иной» и подобными. Следует быть аккуратным, так как эти слова могут относиться к объектам, тогда они означают равенство, а могут — к каким-то его свойствам, тогда используются предикаты. Например, во фразе «кроме Ильи у Петра есть другие сыновья» слово «другой» относится к объекту, поэтому соответствующая формула имеет вид $(\exists x)(P(p, x) \wedge M(x) \wedge \neg x \approx i)$. А в предложении «нет красных карандашей, но есть других цветов» слово «другой» относится к свойству «красный», поэтому формула приобретает вид $\neg(\exists x)K(x) \wedge (\exists x)\neg K(x)$.

Равенство и неравенство в естественной речи часто передаются неявно с помощью тех или иных лексических особенностей. Рассмотрим фразу «У Петра есть только один сын». Фактически здесь сказано гораздо больше, чем написано: у Петра есть хотя бы один сын (он не назван) и у Петра нет других сыновей. Следовательно, для неё мы должны записать такую формулу:

$$(\exists x)(P(p, x) \wedge M(x) \wedge \neg(\exists y)(P(p, y) \wedge M(y) \wedge \neg x \approx y)),$$

то есть мы написали, что x — ребёнок Петра, мужчина, и не существует y, который тоже был бы ребёнком Петра, мужчиной, но отличался бы от x. На этом примере видно, что разные переменные не обязаны обозначать разные объекты, это всегда следует делать явно, при помощи неравенств.

Пример 25. *Рассмотрим сигнатуру, включающую описанные выше символы P, S и M для обозначения родственных связей, а также множество предметных переменных x, y, z, u, v.*

Чтобы записать фразу «x — брат y», мы должны сказать, что x — мужчина, он не совпадает с y (y не является братом самому себе) и у x и y есть общий родитель:

$$M(x) \wedge \neg x \approx y \wedge (\exists z)(P(z, x) \wedge P(z, y)).$$

Попробуем записать фразу «У x все дочери замужем и имеют по одному ребёнку». Пусть переменная z будет означать дочерей x: $P(x,z) \wedge \neg M(z)$. «Замужем» означает «состоит в браке с мужчиной»: $(\exists y)(M(y) \wedge S(z,y))$. Как будет выглядеть утверждение «z имеет одного ребёнка», мы уже рассматривали:

$$(\exists u)(P(z,u) \wedge \neg(\exists v)(P(z,v) \wedge \neg u \approx v)).$$

Тогда вся фраза выражается следующим образом:

$$(\forall z)(P(x,z) \wedge \neg M(z) \rightarrow \\ \rightarrow (\exists y)(M(y) \wedge S(z,y)) \wedge (\exists u)(P(z,u) \wedge \neg(\exists v)(P(z,v) \wedge \neg u \approx v))).$$

Бывает так, что описываемая предметная область состоит из разнотипных объектов. В этом случае нужно ввести одноместные предикатные символы, чтобы отличать объекты разных типов друг от друга.

Пример 26. *Опишем некоторые простейшие понятия математического анализа. Предметная область должны включать действительные числа и одноместные бесконечно дифференцируемые функции. Используем одноместный символ F в качестве свойства «быть функцией», тогда отрицание $\neg F(x)$ будет означать, что x — число, так как типов только два. Также используем трёхместный предикатный символ V и двухместный символ D. $V(f,x,y)$ используется для обозначения отношения «значение функции f на аргументе x равно y», а $D(f,g)$ — отношения «f — производная g».*

Тогда следующая формула будет означать, что функция f является константой:

$$(\exists y)(\forall x)(\neg F(x) \rightarrow V(f,x,y)),$$

если добавить условие, что производная функции совпадает с ней, то мы выразим, что f является константой ноль:

$$D(f,f) \wedge (\exists y)(\forall x)(\neg F(x) \rightarrow V(f,x,y)),$$

а взяв значение этой функции — свойство «x является числом ноль»:

$$(\exists f)(D(f,f) \wedge (\exists y)(\forall x)(\neg F(x) \rightarrow V(f,x,y)) \wedge V(f,x,x)).$$

Заметим, что для переменных f и y указывать $F(f)$ и $\neg F(y)$ излишне, так как отношение $V(f,x,y)$ это само гарантирует.

Уже построенным формулам удобно давать названия, чтобы в дальнейшем их использовать. Поскольку свободные переменные можно называть по разному, то их нужно указывать. Пусть, например, $\mathrm{const}(f)$ *— первая формула, означающая, что* f *— константа, а* $\mathrm{zero}(x)$ *— третья формула, говорящая, что* x *равен нулю.*

Утверждение «x *равен* 1*» можно выразить с помощью значения производной тождественной функции:*

$$\mathrm{id}(f) = (\forall x)(\neg F(x) \to V(f, x, x));$$
$$\mathrm{one}(x) = (\exists f)(\exists e)(D(e, f) \wedge \mathrm{id}(f) \wedge V(e, x, x)).$$

Здесь формула $\mathrm{id}(f)$ *говорит, что* f *— тождественная функция, а* $\mathrm{one}(x)$ *— что* x *является значением её производной.*

Используя построенные формулы, можно определить сложение и умножение действительных чисел. Действительно, чтобы найти $a + b$*, нужно взять функцию вида* $f(x) = x + b$ *и найти её значение в точке* a*. Функцию* $x + b$ *можно отличить от остальных тем, что её производная является константой* 1*, а значение в точке* 0 *равно* b*:*

$$A(x, y, z) = (\exists f)(\exists f')(D(f', f) \wedge$$
$$\wedge (\exists u)(\mathrm{one}(u) \wedge (\forall v)(\neg F(v) \to V(f', v, u))) \wedge$$
$$\wedge (\exists u)(\mathrm{zero}(u) \wedge V(f, u, y)) \wedge V(f, x, z)).$$

Аналогичным образом можно поступить для умножения: возьмём функцию $f(x) = bx$ *и найдём её значение в точке* a*. Функция* bx *«опознаётся» тем, что её производная является константой* b*, а значение в точке* 0 *равно* 0*:*

$$M(x, y, z) = (\exists f)(\exists f')(D(f', f) \wedge (\forall v)(\neg F(v) \to V(f', v, y)) \wedge$$
$$\wedge (\exists u)(\mathrm{zero}(u) \wedge V(f, u, u)) \wedge V(f, x, z)).$$

§ 7.3. Интерпретации и значение формул

Даже приведённых выше небольших примеров достаточно, чтобы понять, что семантика (значение, смысл) формул логики предикатов зависит от той предметной области, о которой идёт речь в формулах, а также от того, какой смысл мы придаём предикатным символам и предметным переменным.

Определение 56 (Интерпретация)**.** *Пусть $\Sigma = (\boldsymbol{P}, \boldsymbol{V})$ — сигнатура. Интерпретацией сигнатуры Σ назовём тройку $I = (A, \nu, \sigma)$, в которой*

1) *A — непустое множество, которое называется предметной областью (или носителем) интерпретации I;*
2) *ν — функция, определённая на множестве предикатных символов $\boldsymbol{P}$, такая, что если R — n-местный предикатный символ, то $\nu(R)$ — n-местное отношение на множестве A;*
3) *σ — функция, отображающая множество переменных $\boldsymbol{V}$ в A.*

С помощью $(I)_a^x$, где $x \in \boldsymbol{V}$, $a \in A$, обозначается интерпретация (A, ν, τ), в которой $\tau(x) = a$ и $\tau(y) = \sigma(y)$ для $y \in \boldsymbol{V} \setminus \{x\}$. Функцию τ тоже будем в этом случае обозначать с помощью $(\sigma)_a^x$.

Таким образом, $(I)_a^x$ отличается от I только тем (если вообще отличается), что значение переменной x равно a.

Определение 57 (Значение формулы)**.** *Пусть зафиксирована сигнатура Σ. Определим теперь значение формулы Φ сигнатуры Σ в интерпретации $I = (A, \nu, \sigma)$ этой же сигнатуры. Обозначаем это значение с помощью $I(\Phi)$. Будем делать это индукцией в соответствии с шагами построения формулы Φ.*

1) $I(x \approx y) = \begin{cases} 1, & \text{если } \sigma(x) = \sigma(y), \\ 0 & \text{в противном случае;} \end{cases}$
2) $I(R(x_1, \ldots, x_n)) = \begin{cases} 1, & \text{если } (\sigma(x_1), \ldots, \sigma(x_n)) \in \nu(R), \\ 0 & \text{в противном случае.} \end{cases}$

Для формул, построенных с помощью булевых связок 3)–6), значение определяется той же таблицей, что и в логике высказываний (определение 29 на стр. 70).

7) *$I((\exists x)\Psi) = 1$, если $(I)_a^x(\Psi) = 1$ для некоторого $a \in A$, в противном случае $I((\exists x)\Psi) = 0$;*

8) $I((\forall x)\Psi) = 1$, *если* $(I)_a^x(\Psi) = 1$ *для каждого* $a \in A$, *в противном случае* $I((\forall x)\Psi) = 0$.

Если значение формулы Φ *в интерпретации* I *равно* 1, *то говорят, что* Φ *истинна в* I, *а* I *является моделью* Φ, *это записывается в виде* $I \models \Phi$. *В противном случае —* Φ *ложна в* I.

Из определения получаем, что истинность атомных формул проверяется так. Равенство $x \approx y$ истинно, если эти переменные имеют одинаковое значение. Для определения истинности предиката R от набора переменных нужно проверить, входит набор значений переменных в отношение, именем которого является R.

Формулы, построенные с помощью булевых связок, получают своё значение таким же образом, как и логике высказываний.

Формула $(\exists x)\Psi$ истинна, если есть возможность подобрать значение x, не меняя ничего больше, чтобы формула Ψ стало истинной. Формула $(\forall x)\Psi$ истинна, если при всяком изменении значения x (в том числе — на то же самое значение), формула Ψ остаётся истинной.

Пример 27. *Элементарной арифметикой натуральных чисел называется следующая интерпретация* $\mathcal{N} = (\omega, \nu, \sigma)$: *носитель — множество натуральных чисел* ω, *трёхместные предикатные символы* $A(x, y, z)$ *и* $M(x, y, z)$ *означают, что* $x + y = z$ *и* $xy = z$ *соответственно, а названия чисел записывают обычным образом в десятичной системе:* 0 *— нуль,* 1 *— один и т. д. Также есть переменные* x, y, z *и другие, значения которых заранее неизвестны.*

Найдём значение формулы $(\exists u)A(x, u, y)$. *Из семантики квантора существования получаем, что эта формула будет истинной, если найдётся число* a *такое, что* $(\mathcal{N})_a^u(A(x, u, y)) = 1$, *то есть если к* $\sigma(x)$ *прибавить* a, *то получим* $\sigma(y)$. *Очевидно, что такое натуральное число* a *можно найти в том и только том случае, когда* $\sigma(x) \leqslant \sigma(y)$. *Следовательно, можно сказать, что формула* $(\exists u)A(x, u, y)$ *выражает отношение нестрогого порядка на натуральных числах.*

Рассмотрим теперь формулу

$$\neg x \approx 0 \wedge \neg x \approx 1 \wedge (\forall u)(\forall v)(M(u, v, x) \to u \approx x \vee v \approx x). \qquad (18)$$

Эта формула будет истинной в том и только том случае, когда $\sigma(x)$ *— простое число. В самом деле, если число* $\sigma(x)$ *простое, то оно не меньше 2,*

поэтому первые два неравенства выполнены. Если произведение двух чисел равно простому, то один из множителей равен 1, а второй — самому числу. Следовательно, какие бы значения для переменных u и v мы не выбрали так, чтобы левая часть импликации стала истинной, правая часть тоже будет истинной.

Пусть теперь формула (18) истинна. Из истинности первых двух неравенств заключаем, что $\sigma(x) \geqslant 2$. Истинность третей части означает, что для любых натуральных чисел a и b будет выполнено

$$(\mathcal{N})_{a\,b}^{u\,v}(M(u,v,x) \to u \approx x \vee v \approx x) = 1. \tag{19}$$

Последнее истинно, если выполнено хотя бы одно из трёх: $\sigma(u) \times \sigma(v) \neq \sigma(x)$, $\sigma(u) = \sigma(x)$ или $\sigma(v) = \sigma(x)$. Но это возможно только если $\sigma(x)$ — простое число, в противном случае мы могли бы разложить $\sigma(x)$ на меньшие множители, взять их в качестве a и b соответственно и получить, что формула (19) ложна.

Рассмотрим ещё один пример:

$$(\exists y)(\neg y \approx 1 \wedge (\exists u)M(y,u,x) \wedge (\forall u)((\exists v)M(u,v,x) \to u \approx 1 \vee (\exists v)M(y,v,u))).$$

Формула $(\exists u)M(y,u,x)$ будет истинной, если $\sigma(y)$ делит $\sigma(x)$. Аналогично, формула $(\exists v)M(u,v,x)$ истинна в случае, когда $\sigma(u)$ делит $\sigma(x)$, а формула $(\exists v)M(y,v,u)$ — когда $\sigma(y)$ делит $\sigma(u)$. Таким образом, импликация утверждает: если u — делитель x, то u равно единице или делится на y. Следовательно, вся формула говорит следующее: существует такой y, делитель x, что любой делитель x равен единице или делится на y.

Такое возможно тогда и только тогда, когда значение x является ненулевой степенью какого-то простого числа. В самом деле, пусть $\sigma(x) = p^n$ для простого p, $n > 0$. Возьмём в качестве значения y само число p. Тогда первые два элемента конъюнкции, очевидно, истинны. Делителями p^n могут быть только степени p. Если значение u равно единице, то истинно $u \approx 1$, если же значением u является p^k, $k > 0$, то истинно $(\exists v)M(y,v,u)$.

Пусть теперь значение x не является ненулевой степенью простого числа. Возможны два случая: $\sigma(x) = 1$ или $\sigma(x) \neq 1$. В первом случае ни для какого значения y не могут быть истинными два первых элемента конъюнкции, так как единственный делитель единицы — сама единица. Покажем, что и во втором случае подходящего значения b для y найти нельзя. Это значение b должно быть неединичным делителем числа $\sigma(x)$. Если существует неединичный делитель c числа $\sigma(x)$, который меньше b,

то взяв это c в качестве значения u мы сделаем ложной импликацию, то есть третий элемент конъюнкции ложен. Если же b будет наименьшим из неединичных делителей, то он должен быть простым. Так как $\sigma(x)$ не является степенью простого числа, в том числе и степенью b, то у $\sigma(x)$ должен быть ещё какой-то простой делитель d. Тогда возьмём d в качестве значения u, и снова третий элемент конъюнкции будет ложен.

Следующие определения почти дословно повторяют аналогичные из логики высказываний.

Определение 58 (Тождественная истинность и ложность, выполнимость)**.** *Формула Φ называется тождественно истинной (или общезначимой), если она истинна во всех интерпретациях.*

Формула Φ называется тождественно ложной, если она ложна во всех интерпретациях.

Формула Φ называется выполнимой, если она истинна хотя бы в одной интерпретации.

Из определения значения формулы нетрудно вывести, что значение формулы Φ в интерпретации не изменится, если поменять значение переменных, которые не входят в Φ свободно.

Предложение 40. *Пусть зафиксирована произвольная сигнатура Σ, Φ — формула сигнатуры Σ, $I = (A, \nu, \sigma)$ и $J = (A, \nu, \tau)$ — интерпретации сигнатуры Σ, причём $\sigma(x) = \tau(x)$ для всех переменных, входящих в Φ свободно. Тогда $I(\Phi) = J(\Phi)$.*

Доказательство. Используем индукцию по построению Φ. Действительно, утверждение, очевидно, справедливо для атомных формул, так как все вхождения переменных в них — свободные, поэтому в I и J они имеют одни и те же значения.

Для булевых связок результат легко следует из индукционного предположения: если $I(\Psi_1) = J(\Psi_1)$, $I(\Psi_2) = J(\Psi_2)$ и $\circ$ — это один из символов $\wedge, \vee, \to$, то $I(\Psi_1 \circ \Psi_2) = J(\Psi_1 \circ \Psi_2)$ и $I(\neg\Psi_1) = J(\neg\Psi_1)$ (проведите это рассуждение самостоятельно).

Рассмотрим формулу вида $\Phi = (\exists x)\Psi$. Пусть, например, $I(\Phi) = 1$. По определению это означает, что $(I)_a^x(\Psi) = 1$ для некоторого $a \in A$. Формула Ψ имеет те же свободные переменные, что и Φ, а также,

возможно, x. Тогда в интерпретациях $(I)_a^x$ и $(J)_a^x$ совпадают значения всех свободных переменных формулы Ψ. По индукционному предположению $(J)_a^x(\Psi) = (I)_a^x(\Psi) = 1$. Из определения получаем $J((\exists x)\Psi) = 1$.

Аналогично рассматривается случай с квантором всеобщности. □

§ 7.4. Замена предметных переменных

Одной из операций, которую часто приходится применять при работе с формулами логики предикатов, является замена предметных переменных.

Определение 59 (Замена предметной переменной)**.** *Пусть* Φ *— формула,* x *и* y *— предметные переменные. Тогда* $(\Phi)_y^x$ *обозначает результат замены в формуле* Φ *всех свободных вхождений переменной* x *на переменную* y*, при условии, что все новые вхождения* y *тоже являются свободными. Если второе условие не выполнено, то* $(\Phi)_y^x$ *неопределено.*

Говоря другими словами, при замене $(\Phi)_y^x$ свободные вхождения x в формуле Φ не должны находиться в области действия кванторов по переменной y.

Пример 28. *В примере 24 на стр. 138 для формулы* Φ *можно построить*

$$(\Phi)_z^y = (\forall x)((\exists x)(P(x,z) \wedge (\forall y)Q(x,y)) \vee \neg P(x,x)).$$

Однако $(\Phi)_x^y$ *неопределено, так как при попытке замены:*

$$(\forall x)((\exists x)(P(x,\boldsymbol{x}) \wedge (\forall y)Q(x,y)) \vee \neg P(x,x))$$

новое вхождение x *(отмечено красным) не будет свободным: это вхождение* y *находилось в формуле* Φ *в области действия квантора* $(\exists x)$.

Аналогичным образом определяется замена нескольких переменных: $(\Phi)_{y_1\ldots y_n}^{x_1\ldots x_n}$ означает замену каждого свободного вхождения переменной x_i на y_i для всех $i = 1, \ldots, n$. При этом все получивши-

еся новые вхождения переменных $y_1, \ldots, y_n$ должны остаться свободными. Следует подчеркнуть, что в последнем случае все замены выполняются одновременно, а не последовательно. Например, $(x \approx y)^{xy}_{yv} = y \approx v$, в то время как при последовательной замене получим $((x \approx y)^x_y)^y_v = v \approx v$.

Докажем ещё одну лемму, которая понадобится нам в будущем.

Лемма 41 (О замене)**.** *Пусть x и y — переменные, $I = (A, \nu, \sigma)$ — интерпретация, $\sigma(y) = a$, Φ — формула. Тогда $I((\Phi)^x_y) = (I)^x_a(\Phi)$.*

Доказательство. Используем индукцию по построению Φ.

Прежде всего заметим, что для любой переменой u выполнено $(\sigma)^x_a(u) = \sigma((u)^x_y)$. В самом деле, при $u = x$ и то, и другое будет равно a, а для других u значение не меняется.

Если $\Phi = u \approx v$, то $(\Phi)^x_y = (u)^x_y \approx (v)^x_y$. Но тогда

$$(I)^x_a(u \approx v) = \left\{ \begin{array}{ll} 1, & \text{если } (\sigma)^x_a(u) = (\sigma)^x_a(v), \\ 0, & \text{иначе} \end{array} \right\} =$$

$$= \left\{ \begin{array}{ll} 1, & \text{если } \sigma((u)^x_y) = \sigma((v)^x_y), \\ 0, & \text{иначе} \end{array} \right\} = I((u)^x_y \approx (v)^x_y)$$

Аналогично при $\Phi = R(u_1, \ldots, u_n)$ получаем

$$(I)^x_a(R(u_1, \ldots, u_n)) =$$

$$= \left\{ \begin{array}{ll} 1, & \text{если } ((\sigma)^x_a(u_1), \ldots, (\sigma)^x_a(u_n)) \in \nu(R), \\ 0, & \text{иначе} \end{array} \right\} =$$

$$= \left\{ \begin{array}{ll} 1, & \text{если } (\sigma((u_1)^x_y), \ldots, \sigma((u_n)^x_y)) \in \nu(R), \\ 0, & \text{иначе} \end{array} \right\} =$$

$$= I(R((u_1)^x_y, \ldots, (u_n)^x_y))$$

Для булевых связок индукционный шаг непосредственно вытекает из индукционного предположения.

Рассмотрим случай с квантором существования: $\Phi = (\exists u)\Psi$. Если $u = x$, то в силу предложения 40 на стр. 147 $I(\Phi) = (I)^x_a(\Phi)$, так как

в формуле $(\exists x)\Psi$ свободных вхождений x нет. С другой стороны, по той же причине $(\Phi)^x_y = \Phi$. Получаем

$$I((\Phi)^x_y) = I(\Phi) = (I)^x_a(\Phi).$$

Если переменная u отличается от x, то $((I)^x_a)^u_b = ((I)^u_b)^x_a$, так как значения различных переменных всё равно в каком порядке изменять, на результат это не повлияет. Тогда получаем следующую цепочку рассуждений. Если $(I)^x_a((\exists u)\Psi) = 1$, то существует такое $b \in A$, что $((I)^x_a)^u_b(\Psi) = 1$, что означает $((I)^u_b)^x_a(\Psi) = 1$. По индукционному предположению $((I)^u_b)^x_a(\Psi) = (I)^u_b((\Psi)^x_y)$. Последнее означает $I((\exists u)(\Psi)^x_y) = 1$. В обратную сторону применяются те же рассуждения, но в противоположном порядке.

Доказательство для квантора всеобщности аналогично. □

§ 7.5. Эквивалентные преобразования и предварённые формулы

С эквивалентными формулами мы уже работали в логике высказываний.

Определение 60 (Следование и эквивалентность)**.** *Формула* Φ *следует из формулы* Ψ*, если* Φ *истинна во всех интерпретациях, в которых истинна* Ψ*, обозначаем это с помощью* $\Psi \Rightarrow \Phi$*.*

Формулы Φ *и* Ψ *эквивалентны, если они следуют друг из друга, обозначаем* $\Psi \equiv \Phi$*.*

Сразу заметим, что все эквивалентности логики высказываний (параграф 4.1 на стр. 79) остаются в силе и в логике предикатов, поскольку семантика булевых связок не изменилась.

Из леммы 41 на предыдущей странице можно вывести следующее.

Предложение 42. *Если* $\Phi \equiv \Psi$*, то* $(\Phi)^x_y \equiv (\Psi)^x_y$*.*

Доказательство. $I((\Phi)^x_y) = (I)^x_a(\Phi) = (I)^x_a(\Psi) = I((\Psi)^x_y)$, если значение y в I равно a. □

Для логики предикатов остаётся справедливым и принцип замены (теорема 18 на стр. 82). Однако, в связи с наличием предметных переменных его нужно уточнить.

Определение 61 (Замена предиката)**.** *Пусть Q — n-местный предикатный символ, Φ — произвольная формула, а Ψ — формула, не имеющая никаких свободных переменных, кроме, может быть, $x_1, \ldots, x_n$, причём порядок переменных заранее неким образом зафиксирован (например, по алфавиту или по возрастанию индексов). Тогда с помощью $(\Phi)^Q_\Psi$ мы будем обозначать результат замены в формуле Φ каждой атомной формулы вида $Q(z_1, \ldots, z_n)$ на $(\Psi)^{x_1 \ldots x_n}_{z_1 \ldots z_n}$.*

Пример 29. *Пусть*

$$\Phi = (\exists x)(Q(x, y) \wedge (\forall v)(Q(y, v) \to x \approx v)),$$
$$\Psi = (\exists u)(\neg u \approx y \wedge R(u, x)).$$

Здесь Q и R — двухместные предикатные символы. В формуле Ψ имеются свободные переменные y и x. Следовательно, при замене $(\Phi)^Q_\Psi$ вместо $Q(x, y)$ мы должны подставить $(\Psi)^{yx}_{xy} = (\exists u)(\neg u \approx x \wedge R(u, y))$, а вместо $Q(y, v)$ — $(\Psi)^{yx}_{yv} = (\exists u)(\neg u \approx y \wedge R(u, v))$. Окончательно получим

$$(\Phi)^Q_\Psi = (\exists x)((\exists u)(\neg u \approx x \wedge R(u, y)) \wedge (\forall v)((\exists u)(\neg u \approx y \wedge R(u, v)) \to x \approx v)).$$

Теорема 43 (Принцип замены)**.** *Если Q — n-местный предикатный символ, формулы Φ и Ψ не имеют никаких свободных переменных, кроме, может быть, $x_1, \ldots, x_n$, и $\Phi \equiv \Psi$, то $(\Theta)^Q_\Phi \equiv (\Theta)^Q_\Psi$, для любой формулы Θ.*

Доказательство. Используется индукция по построению формулы Θ, как в теореме 18 на стр. 82 (см. задачу 128 на стр. 160). □

Новые эквивалентности будут связаны с кванторами.

Теорема 44. *Для любой переменной x и любых формул Φ и Ψ таких, что Ψ не содержит свободных вхождений x, имеют место следующие эквивалентности:*

1) $\neg(\forall x)\Phi \equiv (\exists x)\neg\Phi$;
2) $\neg(\exists x)\Phi \equiv (\forall x)\neg\Phi$;
3) $\Psi \wedge (\forall x)\Phi \equiv (\forall x)(\Psi \wedge \Phi)$;
4) $\Psi \wedge (\exists x)\Phi \equiv (\exists x)(\Psi \wedge \Phi)$;
5) $\Psi \vee (\forall x)\Phi \equiv (\forall x)(\Psi \vee \Phi)$;
6) $\Psi \vee (\exists x)\Phi \equiv (\exists x)(\Psi \vee \Phi)$;
7) $(\exists x)\Psi \equiv (\forall x)\Psi \equiv \Psi$.

Доказательство. Доказательства всех этих эквивалентностей получаются из определения значения формулы логики предикатов в интерпретации $I = (A, \nu, \sigma)$.

Рассмотрим, например, первую из них. Пусть $I(\neg(\forall x)\Phi) = 1$. Это означает, что $I((\forall x)\Phi) = 0$. Из определения значения для квантора всеобщности получаем, что не для всех $a \in A$ имеет место $(I)^x_a(\Phi) = 1$, то есть хотя бы для одного $a \in A$ выполнено $(I)^x_a(\Phi) = 0$. Тогда для этого a будет выполняться $(I)^x_a(\neg\Phi) = 1$, а по определению значения для квантора существования получим $I((\exists x)\neg\Phi) = 1$. Чтобы доказать эквивалентность в другую сторону нужно все эти шаги повторить в обратном порядке.

Рассмотрим ещё доказательство четвёртой эквивалентности. Если $I(\Psi \wedge (\exists x)\Phi) = 1$, то по определению истинности для конъюнкции получаем $I(\Psi) = 1$ и $I((\exists x)\Phi) = 1$. Второе означает, что найдётся $a \in A$, для которого $(I)^x_a(\Phi) = 1$. Так как переменная x не входит свободно в формулу Ψ, а интерпретации I и $(I)^x_a$ отличаются только значением этой переменной, то из предложения 40 на стр. 147 получаем $(I)^x_a(\Psi) = I(\Psi) = 1$. Значит, $(I)^x_a(\Psi \wedge \Phi) = 1$. Тогда $I((\exists x)(\Psi \wedge \Phi)) = 1$. В обратную сторону эквивалентность получается теми же рассуждениями, выполненными в противоположном порядке.

Седьмые эквивалентности непосредственно получаются из предложения 40 на стр. 147. Если $I((\exists x)\Psi) = 1$, то $(I)^x_a(\Psi) = 1$ для некоторого a, поэтому $I(\Psi) = 1$. Если $I(\Psi) = 1$, то $I((\exists x)\Psi) = 1$, так как для истинности Ψ даже значение x менять не нужно.

Остальные эквивалентности доказываются по такому же образцу (задача 129 на стр. 160). □

Останутся ли эквивалентности 3)–6) справедливыми, если мы не будем требовать, чтобы формула Ψ не содержала свободных вхождений переменной x? Нет.

Пример 30. *Рассмотрим, например, две формулы* $\Phi = x \approx x$ *и* $\Psi = x \approx y$. *Тогда формула* $(\Psi \wedge (\forall x)\Phi)$ *в любой интерпретации будет истинной, когда значения переменных* x *и* y *совпадают, так как часть* $(\forall x)x \approx x$ *тождественно истинна. С другой стороны, формула* $(\forall x)(\Psi \wedge \Phi)$ *будет истинной, только когда носитель интерпретации содержит один элемент. В противном случае можно выбрать значение переменной* x *отличное от значения* y.

Чтобы иметь возможность передвигать квантор в описанном случае, нужно иметь возможность переименовывать связанные переменные.

Лемма 45 (Переименование связанной переменной)**.** *Пусть переменная* y *не входит свободно в формулу* Φ *и результат замены* $(\Phi)^x_y$ *определён. Тогда*

$$(\exists x)\Phi \equiv (\exists y)(\Phi)^x_y; \quad (\forall x)\Phi \equiv (\forall y)(\Phi)^x_y.$$

Доказательство. Докажем первую из эквивалентностей.

Если переменные x и y совпадают, то и доказывать нечего, обе формулы просто равны.

Рассмотрим случай, когда переменные x и y различны. Пусть $I((\exists x)\Phi) = 1$ в интерпретации $I = (A, \nu, \sigma)$. Значит, существует $a \in A$, для которого $(I)^x_a(\Phi) = 1$. Как мы уже отмечали при доказательстве леммы 41 на стр. 149 в этом случае $((I)^x_a)^y_a = ((I)^y_a)^x_a$. Так как переменная y не входит в Φ свободно, то, согласно предложению 40 на стр. 147,

$$((I)^y_a)^x_a(\Phi) = ((I)^x_a)^y_a(\Phi) = (I)^x_a(\Phi) = 1.$$

Поскольку $(\sigma)^y_a(y) = a$, то согласно лемме 41 на стр. 149 мы получаем $((I)^y_a)^x_a(\Phi) = (I)^y_a((\Phi)^x_y)$. Последнее означает $I((\exists y)(\Phi)^x_y) = 1$.

Чтобы показать следование в обратную сторону, достаточно заметить, что в условиях леммы $((\Phi)^x_y)^y_x = \Phi$. □

Определение 62 (Предварённая формула)**.** *Формула, которая имеет вид* $(Q_1x_1)\dots(Q_nx_n)\Phi$*, где* Φ *— бескванторная формула,* $x_1,\dots,x_n$ *— переменные, а каждый символ* Q_i *— это один из символов* $\forall$ *или* $\exists$*, называется предварённой.*

Строка $(Q_1x_1)\dots(Q_nx_n)$ *в этом случае называется кванторной приставкой, а формула* Φ *— матрицей этой формулы.*

Любую формулу можно преобразовать к предварённому виду.

Теорема 46 (О предварённой форме)**.** *Для всякой формулы* Φ *существует эквивалентная предварённая формула* Ψ*.*

Доказательство. Используем индукцию по построению формулы Φ. Атомные формулы являются бескванторными, поэтому они уже являются предварёнными с пустой кванторной приставкой.

Если для формулы Φ найдена эквивалентная предварённая формула Ψ, то для формул $(\exists x)\Phi$ и $(\forall x)\Phi$ предварёнными будут формулы $(\exists x)\Psi$ и $(\forall x)\Psi$ соответственно.

Рассмотрим случай с отрицанием: $\Phi=\neg\Phi_1$. По индукционному предположению для Φ_1 уже найдена эквивалентная предварённая формула:

$$\Phi_1\equiv\Psi_1=(Q_1x_1)\dots(Q_nx_n)\Theta.$$

Тогда используем эквивалентности 1) и 2) из теоремы 44 на стр. 151 и получим

$$\Phi=\neg\Phi_1\equiv\neg\Psi_1=\neg(Q_1x_1)\dots(Q_nx_n)\Theta\equiv(Q_1^*x_1)\dots(Q_n^*x_n)\neg\Theta,$$

где с помощью Q^* обозначен кванторный знак противоположный Q. Последняя формула является предварённой.

Наконец, рассмотрим случай бинарных булевых связок. Возьмём, например, конъюнкцию: $\Phi=\Phi_1\wedge\Phi_2$. По индукционному предположению для формул Φ_1 и Φ_2 уже найдены эквивалентные предварённые:

$$\Phi_1\equiv\Psi_1=(Q_1x_1)\dots(Q_nx_n)\Theta_1;$$
$$\Phi_2\equiv\Psi_2=(P_1y_1)\dots(P_my_m)\Theta_2.$$

Выберем новые переменные $u_1, \ldots, u_n$ и $v_1, \ldots, v_m$, которые не встречаются в этих формулах (напомним, что множество предметных переменных бесконечно, поэтому такие переменные всегда можно найти). Согласно лемме 45 на стр. 153,

$$(Q_1x_1)\ldots(Q_nx_n)\Theta_1 \equiv (Q_1u_1)\ldots(Q_nu_n)(\Theta_1)^{x_1\ldots x_n}_{u_1\ldots u_n};$$
$$(P_1y_1)\ldots(P_my_m)\Theta_2 \equiv (P_1v_1)\ldots(P_mv_m)(\Theta_2)^{y_1\ldots y_m}_{v_1\ldots v_m}.$$

Теперь получаем:

$$\begin{aligned}
\Phi = \Phi_1 \wedge \Phi_2 &\equiv (Q_1x_1)\ldots(Q_nx_n)\Theta_1 \wedge (P_1y_1)\ldots(P_my_m)\Theta_2 \equiv \\
&\equiv (Q_1u_1)\ldots(Q_nu_n)(\Theta_1)^{x_1\ldots x_n}_{u_1\ldots u_n} \wedge (P_1v_1)\ldots(P_mv_m)(\Theta_2)^{y_1\ldots y_m}_{v_1\ldots v_m} \equiv \\
&\equiv (Q_1u_1)\ldots(Q_nu_n)((\Theta_1)^{x_1\ldots x_n}_{u_1\ldots u_n} \wedge (P_1v_1)\ldots(P_mv_m)(\Theta_2)^{y_1\ldots y_m}_{v_1\ldots v_m}) \equiv \\
&\equiv (Q_1u_1)\ldots(Q_nu_n)(P_1v_1)\ldots(P_mv_m)((\Theta_1)^{x_1\ldots x_n}_{u_1\ldots u_n} \wedge (\Theta_2)^{y_1\ldots y_m}_{v_1\ldots v_m}),
\end{aligned}$$

последняя формула является предварённой. Здесь мы применили эквивалентности 3) и 4) из теоремы 44 на стр. 151, пользуясь тем, что в формуле $(\Theta_1)^{x_1\ldots x_n}_{u_1\ldots u_n}$ не встречаются свободно переменные $v_1, \ldots, v_m$, а в формуле $(\Theta_2)^{y_1\ldots y_m}_{v_1\ldots v_m}$ — переменные $u_1, \ldots, u_n$.

Для дизъюнкции доказательство аналогично, а случай импликации сводится к дизъюнкции и отрицанию с помощью эквивалентности $\Phi_1 \to \Phi_2 \equiv \neg\Phi_1 \vee \Phi_2$. □

Пример 31. *Преобразуем к предварённому виду формулу*

$$\Phi = ((\exists y)P(x,y) \wedge \neg((\forall x)P(x,y) \to (\exists x)Q(x,z))).$$

Сначала избавимся от импликации:

$$((\exists y)P(x,y) \wedge \neg(\neg(\forall x)P(x,y) \vee (\exists x)Q(x,z))),$$

далее вынесем квантор из-под внутреннего отрицания:

$$((\exists y)P(x,y) \wedge \neg(\underbrace{(\exists x)\neg P(x,y) \vee (\exists x)Q(x,z)}_{(*)})).$$

Чтобы продолжить, нужно в $(*)$ заменить переменные x на новые:

$$((\exists y)P(x,y) \wedge \neg((\exists u)\neg P(u,y) \vee (\exists v)Q(v,z))).$$

Теперь ничто не мешает сделать следующий шаг:

$$((\exists y)P(x,y) \wedge \neg(\exists u)(\exists v)(\neg P(u,y) \vee Q(v,z))).$$

Опять отрицание:

$$((\exists y)P(x,y) \wedge (\forall u)(\forall v)\neg(\neg P(u,y) \vee Q(v,z))).$$

Далее опять нужно заменять переменные, но теперь задача упрощается — в первой формуле переменных u и v нет, поэтому их можно оставить, и ограничится только заменой y в первой формуле:

$$((\exists w)P(x,w) \wedge (\forall u)(\forall v)\neg(\neg P(u,y) \vee Q(v,z))).$$

И окончательно получаем:

$$(\exists w)(\forall u)(\forall v)(P(x,w) \wedge \neg(\neg P(u,y) \vee Q(v,z))).$$

Конечно же, эквивалентности логики предикатов не ограничиваются уже рассмотренными. Приведём ещё несколько, которые бывают полезны в определённых ситуациях.

Теорема 47. *Имеют место следующие эквивалентности:*

1) $(\exists x)(\Phi \vee \Psi) \equiv (\exists x)\Phi \vee (\exists x)\Psi$;
2) $(\forall x)(\Phi \wedge \Psi) \equiv (\forall x)\Phi \wedge (\forall x)\Psi$;
3) $(\exists x)(\exists y)\Phi \equiv (\exists y)(\exists x)\Phi$;
4) $(\forall x)(\forall y)\Phi \equiv (\forall y)(\forall x)\Phi$.

Доказательство. См. задачу 130 на стр. 160. □

Приведённый перечень производит впечатление неполного, например, почему отсутствует эквивалентность для конъюнкции и квантора существования? Оказывается, что в остальных случаях есть следование лишь в одну сторону.

Теорема 48. *Имеют место такие следования:*

1) $(\exists x)(\Phi \wedge \Psi) \Rightarrow (\exists x)\Phi \wedge (\exists x)\Psi$;
2) $(\forall x)\Phi \vee (\forall x)\Psi \Rightarrow (\forall x)(\Phi \vee \Psi)$;
3) $(\exists x)(\forall y)\Phi \Rightarrow (\forall y)(\exists x)\Phi$.

Доказательство. См. задачу 132 на стр. 160. □

Нетрудно показать, что в обратную сторону указанные следования неверны.

Пример 32. *Рассмотрим такую интерпретацию: носитель — ω, а два одноместных предикатных символа O и E означают, что число нечётное или чётное соответственно. Тогда формула $(\exists x)O(x) \wedge (\exists x)E(x)$, конечно, истинна, а формула $(\exists x)(O(x) \wedge E(x))$ ложна.*

Аналогично можно привести примеры и для двух других случаев (задача 133 на стр. 160).

Задачи

115. Для каждой из следующих формул определить, какие вхождения переменных в них являются свободными, а какие — связанными (и каким квантором). Здесь Q — трёхместный предикатный символ.

(а) $(\forall x)((\exists z)Q(x,z,y) \to (\forall x)(Q(z,x,y) \to (\exists y)(Q(z,y,x) \vee x \approx z)))$;

(б) $(\exists y)((\exists x)(\neg x \approx y \wedge Q(x,x,y) \wedge (\forall x)(Q(x,z,y) \to (\exists z)(Q(x,y,z) \vee Q(z,z,y)))))$;

(в) $(\forall x)((\exists z)Q(z,z,x) \wedge (\exists x)Q(y,y,x)) \vee (\exists x)((\exists z)Q(x,x,z) \to (\forall x)Q(x,y,y))$;

(г) $(\exists y)Q(x,y,z) \vee (\exists x)(Q(z,x,y) \wedge (\forall y)Q(y,x,y)) \wedge (\forall z)(\exists x)Q(z,z,x)$. ▼

116. Для каждой формулы Φ из предыдущей задачи определить, какие из следующих замен возможны: $(\Phi)^x_y$, $(\Phi)^x_z$, $(\Phi)^y_x$, $(\Phi)^y_z$, $(\Phi)^z_x$, $(\Phi)^z_y$. Объяснить, почему. В тех случаях, когда это возможно, найти результат замены. ▼

117. Используя сигнатуру из примера 25 на стр. 141, написать формулы, означающие: ▼

(а) x является внуком y;

(б) у x есть не менее двух детей;

(в) x и y имеют одного общего ребёнка;

(г) у x есть незамужняя сестра;

(д) x и y — двоюродные братья;

(е) x женат, а из его сыновей женат только один.

118. Используя сигнатуру из примера 26 на стр. 142, записать формулы, обозначающие следующее. Допускается строить вспомогательные формулы и использовать построенные ранее: ▼

(а) x — неотрицательное число;

(б) число x меньше числа y;

(в) функция f принимает все действительные значения;

(г) функция f является взаимно однозначной;

(д) функция f — это синус;

(е) число x равно π;

(ж) число x является натуральным.

119. Записать формулы со свободными переменными из $\{x, y, z\}$, которые истинны в элементарной арифметике натуральных чисел тогда и только тогда, когда

(а) x является чётным числом;

(б) x и y взаимно просты;

(в) z лежит в интервале между x и y;

(г) x является наибольшим общим делителем y и z;

(д) x является наименьшим общим кратным всех чисел промежутка $[y; z)$. ▼

120. Пусть сигнатура содержит два предикатных символа $L^{(3)}$ и $E^{(4)}$. Геометрией Тарского называется интерпретация, носителем которой является множество точек на евклидовой плоскости, $L(x, y, z)$ означает, что точки x, y и z лежат на одной прямой и именно в такой последовательности (y между x и z, не обязательно они различны), а $E(x, y, u, v)$ означает, что длины отрезков xy и uv равны.

Записать формулы со свободными переменными из $\{x, y, z, u\}$, которые истинны в геометрии Тарского тогда и только тогда, когда

(а) x, y и z образуют треугольник;

(б) x является серединой отрезка yz;

(в) луч xy является биссектрисой угла $\angle zxu$;

(г) угол $\angle xyz$ является прямым;

(д) точки x, y, z, u лежат на одной окружности;

(е) треугольник $\triangle xyz$ является остроугольным

(ж) отрезок xy короче чем zu;

(з) точка x лежит внутри окружности с центром y и точкой z на ней. ▼

121. Представить каждое из следующих предложений формулой логики предикатов, определив в каждом случае подходящую сигнатуру.

(а) Не все студенты изучают и анализ, и историю.

(б) Только один студент не сдавал экзамен по дискретной математике.

(в) Только один студент сдал все экзамены на 100 баллов.

(г) Максимальные баллы, полученные по дискретной математике, превышают максимальные баллы, полученные по информатике.

(д) Имеется брадобрей, бреющий только тех жителей города, которые не бреются сами.

(е) Есть политики, которые могут обманывать всех людей некоторое время, есть политики, которые могут обманывать некоторых людей всё время, но никто не может обманывать всех людей всё время. ▼

122. Написать формулу логики предикатов, истинную только в интерпретациях, носитель которых содержит один элемент (два, три, ..., k элементов). ▼

123. Написать формулу логики предикатов, истинную только в интерпретациях, в которых $(n + 1)$-местное отношение F означает функцию. ▼

124. Определить, какие из следующих формул истинны в элементарной арифметике, пояснить, что они означают:

(а) $(\forall x)(\exists y)(\exists z)(A(x,y,z) \wedge M(x,y,z))$;

(б) $(\forall x)(\exists y)(A(y,y,x) \vee (\exists z)(A(y,y,z) \wedge A(z,1,x)))$;

(в) $(\forall x)(\forall y)(\forall z)((\exists u)M(x,u,y) \wedge (\exists u)M(y,u,z) \to (\exists u)M(x,u,z))$;

(г) $(\forall x)(\exists y)(M(y,y,x) \vee (\exists z)(M(y,y,z) \wedge M(z,y,x)))$;

(д) $(\forall x)(M(x,x,x) \vee (\exists y)(M(x,x,y) \wedge$
$\wedge (\exists u)(\exists v)(\exists w)(A(x,u,v) \wedge A(v,w,y) \wedge (\exists z)(M(u,w,z) \wedge \neg z \approx 0))))$. ▼

125. Определить, какие из следующих формул истинны в геометрии Тарского, пояснить, что они означают:

(а) $(\forall x)(\forall y)(\exists z)(L(x,y,z) \wedge E(x,y,y,z))$;

(б) $(\forall x)(\forall y)(\forall z)(\exists u)(E(u,x,u,y) \wedge E(u,y,u,z))$;

(в) $(\forall x)(\forall y)(\exists u)(\exists v)(\forall z)(E(z,x,z,y) \to L(z,u,v) \vee L(z,v,u) \vee L(u,z,v))$;

(г) $(\forall x)(\exists y)(\exists z)(\exists u)(L(y,u,z) \wedge (\forall v)(L(y,v,z) \wedge E(x,u,x,v) \to u \approx v))$;

(д) $(\forall x)(\forall y)(\forall z)(\neg x \approx y \wedge \neg E(x,y,x,z) \to (\forall u)(E(x,z,x,u) \wedge L(x,u,y) \to$
$\to (\forall v)(E(x,u,x,v) \wedge E(y,u,y,v) \to u \approx v)))$. ▼

126. Сигнатура Σ взята из примера 26 на стр. 142, интерпретация состоит из действительных чисел и бесконечно дифференцируемых функций на них. Отношение F выделяет последние, $V(f,x,y)$ означает, что функция f в точке x имеет значение y, а $D(f,g)$ означает, что f — производная функции g. Определить, какие из следующих формул истинны в этой интерпретации, пояснить, что они означают:

(а) $(\forall x)(\forall y)((\exists z)(D(z,x) \wedge D(z,y)) \to x \approx y)$;

(б) $(\forall x)(\forall y)(\exists z)(\forall u)(\forall v)(\forall w)(V(x,u,v) \wedge V(y,v,w) \to V(z,u,w))$;

(в) $(\forall x)(\exists y)(\forall u)(\forall v)(V(x,u,v) \to V(y,v,u))$;

(г) $(\forall x)(\forall y)((\exists z)(D(z,x) \wedge D(z,y)) \wedge (\exists u)(\exists v)(V(x,u,v) \wedge V(y,u,v)) \to x \approx y)$;

(д) $(\exists x)(D(x,x) \wedge (\forall y)\neg V(x,y,y))$;

(е) $(\exists x)(\neg D(x,x) \wedge (\exists y)(D(y,x) \wedge D(x,y)))$. ▼

127. Сигнатура Σ состоит из двухместного предикатного символа $E^{(2)}$, предметных переменных $a, \ldots, g$ и других. Интерпретация $\mathfrak{G}$ изображена на рис. 16 на следующей странице, стрелка из x в y означает, что $(x,y) \in E$ (такие интерпретации называются графами, они будут подробно изучаться в главе 9 и далее). Определить, какие из следующих формул истинны в этой интерпретации, пояснить, что они означают:

(а) $(\forall x)(\exists y)(\exists z)(E(x,y) \wedge E(z,x))$;

(б) $(\forall x)(\exists y)(\exists z)(E(x,y) \wedge E(y,z) \wedge E(z,x))$;

(в) $(\forall x)(\exists z)(E(z,z) \wedge (E(x,z) \vee (\exists y)(E(x,y) \wedge E(y,z))))$;

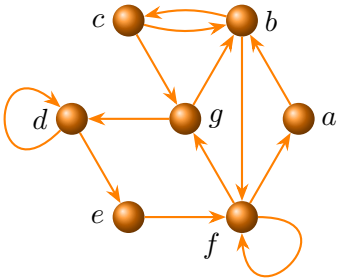


Рис. 16: Граф $\mathfrak{G}$ из задачи 127.

(г) $(\forall x)(\forall y)(E(x,y) \to \neg E(x,x) \vee \neg E(y,y))$;

(д) $(\exists x)(\forall y)(\forall z)(E(x,y) \wedge E(y,z) \to \neg x \approx y \wedge \neg x \approx z)$;

(е) $(\exists x)(\exists y)(E(x,y) \wedge (\forall z)(E(x,z) \to y \approx z))$;

(ж) $(\exists x)(\forall y)(\forall z)(\forall u)(E(a,y) \wedge E(y,z) \wedge E(z,u) \to \neg y \approx x \wedge \neg z \approx x \wedge \neg u \approx x)$. ▼

128. Доказать теорему 43 на стр. 151. ▼

129. Доказать оставшиеся эквивалентности из теоремы 44 на стр. 151.

130. Доказать эквивалентности из теоремы 47 на стр. 156. ▼

131. Доказать следующие эквивалентности: ▼

(а) $(\exists x)(x \approx y \wedge \Phi) \equiv (\Phi)^x_y$;

(б) $(\forall x)(x \approx y \to \Phi) \equiv (\Phi)^x_y$;

(в) $\Phi \vee (\exists x)\Phi \equiv (\exists x)\Phi$;

(г) $\Phi \wedge (\forall x)\Phi \equiv (\forall x)\Phi$.

132. Доказать следования из теоремы 48 на стр. 156. ▼

133. Привести оставшиеся примеры, показывающие, что следования из теоремы 48 на стр. 156 в обратную сторону неверны. ▼

134. Привести к предварённому виду следующие формулы, Q — двухместный предикатный символ:

(а) $\neg(\forall z)\neg(\forall x)(\exists y)(\forall u)(Q(x,y) \wedge Q(z,u))$;

(б) $(\exists x)(\forall y)Q(x,y) \to (\exists x)(\forall y)Q(y,x)$;

(в) $\neg((\forall x)((\forall y)Q(x,y) \to Q(x,z)) \vee ((\forall z)Q(z,x) \wedge \neg(\exists y)Q(y,x)))$. ▼

135. С помощью преобразований доказать следующие эквивалентности, формула Θ не содержит переменной x свободно:

(а) $(\exists x)(\Phi \to \Psi) \equiv (\forall x)\Phi \to (\exists x)\Psi$;

(б) $(\forall x)(\Theta \to \Phi) \equiv \Theta \to (\forall x)\Phi$;

(в) $(\forall x)(\Phi \to \Theta) \equiv (\exists x)\Phi \to \Theta$. ▼

Глава 8

Логика предикатов и базы данных

Краткое содержание: реляционные базы данных, схемы отношений и предикаты, реляционная алгебра, представление формул реляционной алгебры формулами логики предикатов, язык запросов SQL и его связь с логикой предикатов, ограничения целостности: ограничения на ключи, ограничения на ссылки и ограничения на значения атрибутов.

Ключевые слова: реляционная база данных, атрибут, схема отношения, отношение, реляционная алгебра, фильтрация, проекция, соединение, язык запросов SQL, ограничение целостности.

§ 8.1. Реляционные базы данных

Одними из наиболее широко применяемых информационных технологий являются базы данных. Они практически всегда применяются, когда требуется хранение больши́х объёмов информации и её интенсивная обработка. Большинство современных промышленных

баз данных являются *реляционными* — данные в них представляют конечные отношения (relations), которые хранятся в таблицах.

Определение 63 (Схема базы данных)**.** *Атрибутом* $\mathcal{A}$ *называется пара* $(A, D_{\mathcal{A}})$*, где* A *— имя атрибута,* $D_{\mathcal{A}}$ *— непустое множество значений или тип атрибута.*

Схемой отношения называется упорядоченный набор атрибутов $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$*, причём все эти атрибуты должны иметь попарно различные имена. Обычно схему отношений записывают в виде* $(A_1, \ldots, A_n)$*, указывая только имена атрибутов и предполагая, что множества их значений так или иначе уже определены. Две схемы отношений совместны, если они содержат одинаковое число атрибутов и типы атрибутов, стоящих на одних и тех же позициях, совпадают.*

Схемой базы данных называется конечное множество имён отношений с указанием схемы каждого из них.

Для разных имён отношений могут встречаться атрибуты с одинаковыми именами. При этом не требуется совпадение их типов.

Определение 64 (Отношение, база данных)**.** *Пусть дана схема отношения* $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$*. Тогда мы будет говорить, что* n*-местное отношение* R *соответствует этой схеме, если* R *конечно и из* $(x_1, \ldots, x_n) \in R$ *следует, что* $x_i \in D_{\mathcal{A}_i}$ *для* $i = 1, \ldots, n$*. При этом говорят, что* x_i *является значением атрибута* $\mathcal{A}_i$*.*

Состоянием базы данных Δ *схемы* $\mathcal{S}$ *называем отображение множества* $\mathcal{S}$ *имён отношений в множество отношений такое, что* $\Delta(R)$ *соответствует схеме* R *в* $\mathcal{S}$*.*

Обычно схемы баз данных записывают, перечисляя имена отношений, а после них в скобках — имена атрибутов для каждого отношения.

В приложениях отношения чаще называют *таблицами*, их атрибуты — *столбцами*, строки таблиц — *записями*, а элементы записей — *полями*. В каждый момент времени состояние базы данных — это набор конечных таблиц, имеющих соответствующие схемы.

Пример 33. *Пусть, например, база данных со сведениями о сотрудниках некоторой организации имеет схему:*

$$\{\textit{Сотрудники}(\textit{Номер, ФИО, Отдел, Должность, Оклад}),$$
$$\textit{Комнаты}(\textit{НомерСотрудника, Этаж, НомерКомнаты}),$$
$$\textit{Оборудование}(\textit{Этаж, НомерКомнаты, Название})\}.$$

Здесь мы предполагаем, что атрибуты с именами «Номер», «Оклад», «НомерСотрудника», «Этаж», «НомерКомнаты» имеют в качестве области значений натуральные числа, а остальные — множество строк, составленных из букв русского языка.

Пример состояния этой базы данных приведён на рис. 17 на следующей странице.

Нетрудно видеть, что определение базы данных со схемой $\mathcal{S}$ очень похоже на определение интерпретации сигнатуры Σ в логике предикатов. Сигнатура Σ содержит имена отношений и их местность, схема базы данных содержит имена отношений и их схемы. Интерпретация ставит в соответствие каждому предикатному символу некоторое отношение, и то же самое делает состояние базы данных. Единственное отличие — в схеме базы данных область значений у каждого атрибута своя, а в интерпретации имеется единая предметная область. Кроме того, для отношений базы данных присутствует требование конечности.

Пример 34. *В предыдущем примере с точки зрения логики предикатов рассматривается интерпретация, сигнатура которой содержит три предикатных символа: Сотрудники*$^{(5)}$*, Комнаты*$^{(3)}$ *и Оборудование*$^{(3)}$*. Носитель этой интерпретации включает в себя строки русского языка и натуральные числа. Первая из приведённых таблиц задаёт значение предикатного символа Сотрудники*$^{(5)}$*, вторая — значение символа Комнаты*$^{(3)}$*, третья — символа Оборудование*$^{(3)}$*.*

Если мы считаем, что имена отношений из схемы базы данных образуют сигнатуру, а состояние базы данных Δ является интерпретацией I этой сигнатуры, то легко видеть, что n-ка $(a_1, \ldots, a_n)$ принадлежит отношению $\Delta(R)$ тогда и только тогда, когда в интерпретации $(I)^{x_1 \ldots x_n}_{a_1 \ldots a_n}$ истинна формула $R(x_1, \ldots, x_n)$.

Сотрудники				
Номер	ФИО	Отдел	Должность	Оклад
1	Иванов А.А.	торговый	менеджер	7000
2	Сидоров Н.П.	плановый	экономист	5000
3	Сидорова М.И.	торговый	зав. складом	6000
4	Ольгина Н.А.	плановый	экономист	5500
5	Горев С.В.	плановый	зав. отделом	10000

Комнаты		
НомерСотрудника	Этаж	НомерКомнаты
3	2	17
1	2	17
7	2	18
5	3	7
2	3	27

Оборудование		
Этаж	НомерКомнаты	Название
2	17	компьютер
2	17	принтер
3	7	ксерокс
3	25	принтер

Рис. 17: Состояние базы данных.

Далее мы будем предполагать, что элементы отношений входят в число предметных имён, а каждое имя «a» интерпретируется в I как объект a. Поэтому последнее предложение можно сформулировать так: $(a_1, \ldots, a_n)$ принадлежит отношению $\Delta(R)$ тогда и только тогда, когда $I \models R(a_1, \ldots, a_n)$.

Если формула имеет свободные переменные $x_1, \ldots, x_k$, то можно поставить вопрос о том, при каких значениях этих переменных формула Φ истинна.

Пример 35. *Рассмотрим формулу*

$$\Phi = (\exists n)(\exists o)(\exists d)(\exists z)(\exists e)(\mathit{Сотрудники}(n, f, o, d, z) \wedge \\ \wedge\, \mathit{Комнаты}(n, e, k) \wedge o \approx \text{«плановый»})$$

со свободными переменными k и f. Нетрудно проверить, что эта формула будет истинной при следующих вариантах значений для k и f:

f	k
Сидоров Н.П.	*27*
Горев С.В.	*7*

Отметим, что для конечных систем поиск значений свободных переменных формул логики предикатов, при которых они выполняются, и проверка истинности замкнутых формул производятся эффективно.

Определение 65 (Определение отношения формулой)**.** *Будет говорить, что в интерпретации $I = (A, \nu, \sigma)$ формула Φ со свободными переменными $x_1, \ldots, x_n$ определяет n-местное отношение R на множестве A, если для всех $a_1, \ldots, a_n \in A$ выполнено*

$$(a_1, \ldots, a_n) \in R \iff (I)^{x_1 \ldots x_n}_{a_1 \ldots a_n} \models \Phi.$$

Таким образом, каждая формула определяет в точности одно отношение, содержащее те наборы $(a_1, \ldots, a_n)$, на которых она истинна.

Как следствие получаем

Следствие 49. *Если R — имя отношения со схемой $(A_1, \ldots, A_n)$, то формула $R(x_1, \ldots, x_n)$ определяет отношение $\Delta(R)$.*

§ 8.2. Реляционная алгебра

Для манипуляции отношениями Э. Коддом в 1970 г. был предложен набор реляционных операций, позволяющих по одним отношениям получать другие. Каждая такая операция является функцией, аргументами и значениями которой являются отношения. Из базовых реляционных операций можно с помощью суперпозиции образовывать сложные формулы, которые тоже задают некоторые операции. Совокупность получаемых таким образом операций над отношениями называется реляционной алгеброй.

В этом разделе мы рассмотрим шесть реляционных операторов, введённых Коддом в качестве базиса реляционной алгебры, и покажем, как они определяются в терминах логики предикатов.

Первую группу реляционных операций представляют теоретико-множественные операции: объединение, пересечение, разность. В первой главе мы рассматривали все эти операции для множеств. Особенности их использования в реляционной алгебре состоят в том, что объединение, пересечение и вычитание применяются к отношениям, имеющим совместные схемы. Пусть имеются два отношения: R со схемой $(A_1, \ldots, A_n)$ и S со схемой $(B_1, \ldots, B_n)$, совместной с $(A_1, \ldots, A_n)$. Тогда

- о б ъ е д и н е н и е R и S — это отношение $R \cup S$ со схемой $(A_1, \ldots, A_n)$;
- п е р е с е ч е н и е R и S — это отношение $R \cap S$ со схемой $(A_1, \ldots, A_n)$;
- р а з н о с т ь R и S — это отношение $R \setminus S$ со схемой $(A_1, \ldots, A_n)$.

Из определения сразу можно получить

Предложение 50. *Если n-местные отношения R и S определяются формулами Φ и Ψ соответственно со свободными переменными $x_1, \ldots, x_n$, то*

1) *формула $\Phi \vee \Psi$ определяет объединение $R \cup S$;*
2) *формула $\Phi \wedge \Psi$ определяет пересечение $R \cap S$;*
3) *формула $\Phi \wedge \neg\Psi$ определяет разность $R \setminus S$.*

При построении декартова произведения схема отношения меняется. Пусть имеются два отношения: R со схемой $(A_1, \ldots, A_n)$ и S со схемой $(B_1, \ldots, B_m)$. Тогда

- д е к а р т о в ы м п р о и з в е д е н и е м R и S является $R \times S$ со схемой $(R.A_1, \ldots, R.A_n, S.B_1, \ldots, S.B_m)$.

Напомним, что декартово произведение состоит из пар вида (r, s), где $r \in R$, $s \in S$. В нашем случае, когда $r = (a_1, \ldots, a_n)$, $s = (b_1, \ldots, b_m)$,

получаем, что $R \times S$ состоит из наборов вида $(a_1, \ldots, a_n, b_1, \ldots, b_m)$, то есть соответствует схеме $(A_1, \ldots, A_n, B_1, \ldots, B_m)$.

Заметим, что схемы отношений R и S могли иметь атрибуты с одинаковыми именами, поэтому мы в схеме результата к каждому из атрибутов добавили имя отношения, из которого соответствующий атрибут получен, чтобы не было совпадений.

Аналогично получаем

Предложение 51. *Если n-местное отношение R определяется формулой Φ со свободными переменными $x_1, \ldots, x_n$, а m-местное отношение S определяется формулой Ψ со свободными переменными $y_1, \ldots, y_m$, и все переменные $x_1, \ldots, x_n, y_1, \ldots, y_m$ попарно различны, то формула $\Phi \wedge \Psi$ со свободными переменными $x_1, \ldots, x_n, y_1, \ldots, y_m$ определяет декартово произведение $R \times S$.*

Операция фильтрации для применения к отношению R со схемой $(A_1, \ldots, A_n)$ требует задания формулы Θ — булевой комбинации равенств, составленных из переменных, являющимися именами атрибутов, и констант. В этом случае

- результатом ф и л ь т р а ц и и (или в ы б о р а) является отношение R' со схемой $(A_1, \ldots, A_n)$, содержащее в точности те наборы R, которые удовлетворяют формуле Θ. Записываем это в виде $(R : \Theta)$.

Из определения получаем

Предложение 52. *Допустим, что n-местное отношение R определяется формулой Φ со свободными переменными $x_1, \ldots, x_n$. Тогда формула $\Phi \wedge (\Theta)^{A_1 \ldots A_n}_{x_1 \ldots x_n}$ определяет результат фильтрации $(R : \Theta)$.*

Пример 36. *Возьмём в качестве Θ формулу отдел $\approx$ «торговый». Тогда операции выбора (Сотрудники : Θ) соответствует формула*

$$\Psi = \textit{Сотрудники}(n, f, o, d, z) \wedge o \approx \text{«торговый»},$$

со свободными переменными n, f, o, d, z, определяющая отношение:

Номер	*ФИО*	*Отдел*	*Должность*	*Оклад*
1	*Иванов А.А.*	*торговый*	*менеджер*	*7000*
3	*Сидорова М.И.*	*торговый*	*зав. складом*	*6000*

Рассмотрим последнюю из базовых реляционных операций. Пусть имеется отношение R со схемой $(A_1, \ldots, A_n)$ и задана схема проекции $X = (A_{i_1}/B_1, \ldots, A_{i_m}/B_m)$ — произвольный набор атрибутов $A_{i_1}, \ldots, A_{i_m}$ с указанием попарно различных имён $B_1, \ldots, B_m$, $i_1, \ldots, i_m \in \{1, \ldots, n\}$ (каждый атрибут в X может встречаться сколько угодно раз, в том числе и ни одного). Тогда

- проекцией отношения R на X является отношение S со схемой $(B_1, \ldots, B_m)$, которое состоит из всех наборов вида $(a_{i_1}, \ldots, a_{i_m})$, где $(a_1, \ldots, a_n)$ — всевозможные элементы R. Обозначаем проекцию с помощью $R[X]$.

Имена B_j необходимы, если какой-либо из атрибутов A_i встречается в X более чем один раз, чтобы не допустить повторения. Если же при проекции атрибут A_i встречается только один раз, то новое имя для него можно не указывать.

Для проекции тоже можно построить определяющую её формулу. Поскольку это делается менее тривиально, чем в предыдущих случаях, то приведём следующее предложение с доказательством.

Предложение 53. *Предположим, что n-местное отношение R со схемой $(A_1, \ldots, A_n)$ определяется формулой Φ со свободными переменными $x_1, \ldots, x_n$. Тогда формула*

$$(\exists x_1) \ldots (\exists x_n) \left(\Phi \wedge \bigwedge_{j=1}^{m} y_j \approx x_{i_j} \right) \tag{20}$$

определяет результат проекции $R[X]$.

Доказательство. Пусть $(b_1, \ldots, b_m) \in R[X]$. Это означает, что $(b_1, \ldots, b_m) = (a_{i_1}, \ldots, a_{i_m})$ для какого-то набора $(a_1, \ldots, a_n) \in R$. Тогда в качестве значений $x_1, \ldots, x_n$ в формуле (20) можно взять $a_1, \ldots, a_n$ соответственно, чтобы подкванторная конъюнкция стала истинной: Φ истинна, так как она определяет R, а равенства истинны по определению наборов.

Наоборот, пусть формула (20) истинна, если переменные $y_1, \ldots, y_m$ имеют значения $b_1, \ldots, b_m$. Выберем некоторые $a_1, \ldots, a_n$ — значения

переменных $x_1, \ldots, x_n$ соответственно, при которых истинна внутренняя формула. Так как истинна конъюнкция, то истинна и формула Φ, что означает $(a_1, \ldots, a_n) \in R$. Из истинности равенств получаем совпадение наборов $(b_1, \ldots, b_m) = (a_{i_1}, \ldots, a_{i_m})$. Тогда заключаем, что $(b_1, \ldots, b_m) \in R[X]$. □

Пример 37. *Рассмотрим, например, проекцию Сотрудники*$[$*ФИО*, *оклад*$]$. *Ему соответствует формула*

$$\Psi = (\exists n)(\exists f)(\exists o)(\exists d)(\exists z)(\mathit{Сотрудники}(n, f, o, d, z) \wedge f_1 \approx f \wedge z_1 \approx z)$$

со свободными переменными f_1 *и* z_1, *задающая двухместное отношение:*

ФИО	*Оклад*
Иванов А.А.	*7000*
Сидоров Н.П.	*5000*
Сидорова М.И.	*6000*
Ольгина Н.А.	*5500*
Горев С.В.	*10000*

Заметим, что в предложении 53 на предыдущей странице мы построили формулу для проекции в общем случае. Если в схеме проектирования X каждый из атрибутов A_i встречается не более одного раза, то формулу можно упростить.

Предложение 54. *Если в условиях предыдущего предложения* $A_{j_1}, \ldots, A_{j_k}$ *— это атрибуты, не входящие в* X, *а остальные атрибуты встречаются в* X *в точности один раз, то следующая формула определяет проекцию* $R[X]$:

$$(\exists x_{j_1}) \ldots (\exists x_{j_k})\Phi.$$

Следовательно, в предыдущем примере можно взять такую формулу:

$$(\exists n)(\exists o)(\exists d)\text{Сотрудники}(n, f, o, d, z).$$

Широко применяемые на практике операции соединения не входят в число базовых, они легко выражаются через другие. В реляционной алгебре эти операции представлены в нескольких формах.

Пусть R и S — это отношения со схемами $(A_1, \ldots, A_n, B_1, \ldots, B_m)$ и $(B_1, \ldots, B_m, C_1, \ldots, C_k)$ соответственно. Атрибуты $B_1, \ldots, B_m$ являются общими. Здесь общие атрибуты изображены для удобства по краям, но вообще это не требуется, они могут как угодно чередоваться с остальными. Тогда

- естественное соединение $P = R \bowtie S$ отношений R и S имеет схему $(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_k)$ и состоит из всевозможных наборов вида $(a_1, \ldots, a_n, b_1, \ldots, b_m, c_1, \ldots, c_k)$, где $(a_1, \ldots, a_n, b_1, \ldots, b_m) \in R$, $(b_1, \ldots, b_m, c_1, \ldots, c_k) \in S$.

Иными словами, мы для каждого набора из R ищем всевозможные наборы из S, у которых совпадают все одноимённые атрибуты, и соединяем их в единые наборы.

Пример 38. *Рассмотрим отношение*

$$\textit{Доступ} = \textit{Комнаты} \bowtie \textit{Оборудование},$$

определяющее доступность тех или иных аппаратов сотрудникам в их комнатах. Оно имеет схему

$$(\textit{НомерСотрудника}, \textit{Этаж}, \textit{НомерКомнаты}, \textit{Название})$$

и представлено в следующей таблице:

НомерСотрудника	*Этаж*	*НомерКомнаты*	*Название*
3	*2*	*17*	*компьютер*
3	*2*	*17*	*принтер*
1	*2*	*17*	*компьютер*
1	*2*	*17*	*принтер*
5	*3*	*7*	*ксерокс*

Здесь соединение производится по двум общим атрибутам: Этаж и НомерКомнаты. При этом в соединение не попали сведения о сотрудниках с номерами 2 *и* 7*, в комнатах которых нет оборудования, и о принтере в комнате* 25*, так как в ней нет сотрудников.*

Нетрудно видеть, что естественное соединение есть просто комбинация декартова произведения, фильтрации и проекции:

$$R \bowtie S = \left((R \times S) : \bigwedge_{i=1}^{m} R.B_i \approx S.B_i \right) \\ [A_1, \ldots, A_n, R.B_1/B.1, \ldots, R.B_m/B_m, C_1, \ldots, C_k].$$

Здесь мы сначала соединяем каждую строку R со всеми строками из S (декартово произведение), затем — отбираем только те строки, в которых значения одноимённых атрибутов совпадают, наконец — с помощью проекции избавляемся от повторяющихся атрибутов, переименовывая оставшиеся.

Если R и S определяются формулами Φ и Ψ со свободными переменными $(x_1, \ldots, x_n, y_1, \ldots, y_m)$ и $(y_1, \ldots, y_m, z_1, \ldots, z_k)$ соответственно, то общая конструкция даст для естественного соединения такую формулу:

$$(\exists s.y_1) \ldots (\exists s.y_m) \left((\Phi)^{y_1 \ \ldots y_m}_{r.y_1 \ldots r.y_m} \wedge (\Psi)^{y_1 \ \ldots y_m}_{s.y_1 \ldots s.y_m} \wedge \bigwedge_{i=1}^{m} r.y_i \approx s.y_i \right).$$

Учитывая эквивалентности из задачи 131 на стр. 160, её можно упростить до $(\Phi \wedge \Psi)^{y_1 \ \ldots y_m}_{r.y_1 \ldots r.y_m}$ или просто $\Phi \wedge \Psi$, если имена переменных неважны.

Формула для предыдущего примера, следовательно, имеет вид:

$$\text{Комнаты}(n, e, k) \wedge \text{Оборудование}(e, k, o).$$

Другой, более общий, вариант оператора соединения — Θ-соединение отношений R и S, $P_1 = R \overset{\Theta}{\bowtie} S$. Отличий от естественного соединения здесь два: во-первых, вместо приравнивания атрибутов с одинаковыми именами используется явно заданное условие Θ, во-вторых, никакие атрибуты отношений не отбрасываются (поэтому если отношения имели атрибуты с одинаковыми именами, их нужно переименовывать).

Легко понять, что Θ-соединение является комбинацией декартова произведения и фильтрации:

$$R \overset{\Theta}{\bowtie} S = ((R \times S) : \Theta).$$

Операторы реляционной алгебры можно соединять и в более сложные выражения, позволяющие выражать необходимые пользователям запросы.

Пример 39. *Покажем, как получить список ФИО сотрудников, у которых нет компьютера. Для этого сначала найдём номера сотрудников, у которых компьютер есть: можно отношение Доступ из примера 38 на стр. 170 спроектировать на (НомерСотрудника):*

$$((\mathit{Комнаты} \bowtie \mathit{Оборудование}) : \mathit{Название} \approx \text{«компьютер»})[\mathit{НомерСотрудника}]. \quad (21)$$

Далее при помощи вычитания найдём номера оставшихся сотрудников, у которых компьютера нет:

$$\mathit{Сотрудники}[\mathit{Номер}] \setminus (21). \quad (22)$$

Наконец, по этим номерам при помощи соединения и проекции найдём ФИО:

$$(\mathit{Сотрудники} \bowtie (22))[\mathit{ФИО}].$$

§ 8.3. SQL-запросы

Пользователи извлекают информацию из баз данных с помощью запросов. Реляционная алгебра позволяет формулировать запросы в виде соответствующих выражений. Однако их синтаксис не очень удобен для непрофессиональных пользователей. Поэтому современные системы управления базами данных используют языки с более простым и понятным синтаксисом. Одним из самых популярных таких языков является язык SQL (Structured Query Language — Структурный Язык Запросов). Сейчас он служит стандартным языком запросов для всех промышленных баз данных. Мы опишем здесь простейшую форму основного оператора этого языка SELECT, предназначенного для извлечения данных. Она имеет вид:

```
SELECT DISTINCT <список атрибутов>
FROM <список таблиц и запросов>
[WHERE <условие фильтрации>]
```

Часть WHERE, заключённая в квадратные скобки, необязательна. Результат выполнения этого оператора — это декартово произведение всего перечисленного после FROM, к которому затем применён фильтр WHERE (если он есть), а затем — проекция на перечисленные после SELECT DISTINCT атрибуты.

Условие, записанное после WHERE, представляет собой бескванторную формулу, построенную из сравнений, в которых логические знаки $\neg$, $\wedge$, $\vee$ заменены на их английские аналоги NOT, AND, OR соответственно.

Объединение, пересечение и разность реализуются в SQL соответственно:

```
<запрос1> UNION <запрос2>
<запрос1> INTERSECT <запрос2>
<запрос1> EXCEPT <запрос2>
```

Пример 40. *Пусть мы хотим получить для базы данных из примера 33 на стр. 163 список фамилий всех сотрудников, работающих на втором этаже. Соответствующий запрос выглядит так:*

```
SELECT DISTINCT ФИО
FROM Сотрудники, Комнаты
WHERE Номер = НомерСотрудника AND Этаж = 2
```

С помощью реляционной алгебры тот же запрос записывается так:

$$((\mathit{Сотрудники} \times \mathit{Комнаты}) : \mathit{Номер} \approx \mathit{НомерСотрудника} \wedge \mathit{Этаж} \approx 2)[\mathit{ФИО}].$$

Здесь продемонстрирована именно описанная выше последовательность реляционных операций: декартово произведение, фильтрация, проекция.

Нетрудно построить формулу и логики предикатов, определяющую тот же результат:

$$(\exists n)(\exists o)(\exists d)(\exists z)(\exists ns)(\exists e)(\exists k)(\mathit{Сотрудники}(n, f, o, d, z) \wedge \\ \wedge \mathit{Комнаты}(ns, e, k) \wedge n \approx ns \wedge e \approx 2).$$

В построенной формуле первая конъюнкция определяет декартово произведение, следующие — фильтр, а кванторы существования — проекцию.

Нетрудно проверить, что эта формула будет истинна при значениях f равных «Иванов А.А.» и «Сидорова М.И.», которые и будут результатом SQL-запроса.

Пример 41. *Пусть необходимо получить список сотрудников планового отдела и комнат, в которых они трудятся. Запрос на SQL выглядит так:*

```
SELECT DISTINCT ФИО, НомерКомнаты
FROM Сотрудники, Комнаты
WHERE Номер = НомерСотрудника AND Отдел = 'плановый'
```

По аналогии можно построить выражение реляционной алгебры:

$$((\mathit{Сотрудники} \times \mathit{Комнаты}) : \mathit{Номер} \approx \mathit{НомерСотрудника} \wedge \\ \wedge\ \mathit{Отдел} \approx \text{«плановый»})[\mathit{ФИО}, \mathit{НомерКомнаты}].$$

По описанным выше правилам построим формулу логики предикатов, реализующую этот запрос:

$$(\exists n)(\exists o)(\exists d)(\exists z)(\exists ns)(\exists e)(\mathit{Сотрудники}(n, f, o, d, z) \wedge \\ \wedge\ \mathit{Комнаты}(ns, e, k) \wedge n \approx ns \wedge e \approx \text{«плановый»}).$$

Эта формула истинна, когда пара свободных переменных (f, k) принимает следующие значения:

f	k
Сидоров Н.П.	*27*
Горев С.В.	*7*

Таким образом, описанные выше SQL-запросы являются не чем иным, как «синтаксическим сахаром», за которым скрываются выражения реляционной алгебры и, следовательно, формулы логики предикатов, причём весьма специального вида.

Конечно, мы рассмотрели только простейшие варианты запросов языка SQL. Имеются и более сложные формы SQL-запросов, которым соответствуют формулы логики предикатов более общего вида, и даже выходящие за рамки логики предикатов. Однако последние используются крайне редко и только для решения весьма специфических задач. В основном же можно считать, что язык SQL реализует те же возможности, что и логика первого порядка.

§ 8.4. Ограничения целостности

При работе с базой данных её состояние постоянно меняется: добавляются новые записи, изменяются и удаляются старые. При этом нужно, чтобы при всех этих модификациях информация, хранимая в базе, оставалась корректной. Например, для каждого сотрудника имелась лишь одна строка в отношении Сотрудники.

Определение 66 (Ограничение целостности)**.** *Ограничение целостности — это условие, задаваемое для схемы базы данных, которое ограничивает множество возможных состояний базы данных.*

Вот типичные виды ограничений целостности:

- ограничение на ключи: в отношении не должно быть двух строк с одинаковым значением некоторого атрибута (этот атрибут и называется ключом). В общем виде ключ может содержать несколько атрибутов, тогда ограничение утверждает, что в отношении не должно быть двух разных строк с одинаковыми значениями всех этих атрибутов;
- ограничение на ссылки: значение атрибута в одном отношении должно обязательно находиться среди значений некоторого атрибута в другом (или даже в том же самом) отношении. Это ограничение не позволит, например, удалить из второго отношения набор, на который имеется ссылка из первого отношения;
- ограничение на значение: значения некоторых атрибутов должны быть связаны определённым условием. Например, принадлежать определённому интервалу, быть строкой длины не больше (не меньше) заданной, быть строкой, удовлетворяющей некоторому образцу и т. д. Как правило, для задания таких ограничений уже привлекаются не только отношения базы данных, но и отношения и функции, заданные на универсуме, которые в базу данных не входят (например, сравнение, арифметические операции и т. д.).

Современные системы управления базами данных позволяют задавать такие ограничения при конструировании базы данных, а затем автоматически поддерживают их выполнение, не давая пользователям производить модификации, нарушающие эти ограничения.

С точки зрения логики предикатов, ограничение целостности — это замкнутая формула, которая должна быть истинна на допустимых состояниях базы данных.

Пример 42. *Рассмотрим, как можно задать ограничения целостности указанных видов для нашей базы данных из примера 33 на стр. 163.*

В отношении Сотрудники ключом является атрибут Номер, поэтому ограничение на ключ можно сформулировать так:

$$(\forall n)(\forall x)(\forall y)(\forall z)(\forall v)(\forall x_1)(\forall y_1)(\forall z_1)(\forall v_1)$$
$$(\mathit{Сотрудники}(n,x,y,z,v) \wedge \mathit{Сотрудники}(n,x_1,y_1,z_1,v_1) \rightarrow$$
$$\rightarrow x \approx x_1 \wedge y \approx y_1 \wedge z \approx z_1 \wedge v \approx v_1).$$

Каждый сотрудник, для которого определено помещение в отношении Комнаты, должен присутствовать в отношении Сотрудники:

$$\Phi_2 = (\forall ns)(\forall x)(\forall y)(\mathit{Комнаты}(ns,x,y) \rightarrow$$
$$\rightarrow (\exists z)(\exists u)(\exists v)(\exists w)\mathit{Сотрудники}(ns,z,u,v,w)).$$

Оклад каждого сотрудника должен лежать в интервале $(1000, 25000)$*:*

$$(\forall x)(\forall y)(\forall z)(\forall v)(\forall o)(\mathit{Сотрудники}(x,y,z,v,o) \rightarrow 1000 < o \wedge o < 25000).$$

В последнем примере было использовано отношение порядка $<$ на числах, которое является внешним по отношению к базе данных.

Задачи

136. Для базы данных из примера 33 на стр. 163 построить выражение реляционной алгебры, задающее список фамилий сотрудников, в комнатах которых нет никакого оборудования. Построить формулу логики предикатов, определяющую то же отношение.

137. Пусть имеются отношения R и S со схемами $(A_1, \ldots, A_n, B_1, \ldots, B_m)$ и $(B_1, \ldots, B_m)$ соответственно. Частным R/S от деления отношения R на отношение S называется наибольшее отношение Q с атрибутами $A_1, \ldots, A_n$, каждый

набор t которого в соединении с каждым набором $s \in S$ входит в R, то есть $Q \times S \subseteq R$. Построить выражение реляционной алгебры, эквивалентное R/S, и формулу логики предикатов, определяющую это отношение.

138. Написать SQL-запросы и соответствующие формулы для получения следующей информации из базы данных из примера 33 на стр. 163, вычислить результат:

(а) найти всех сотрудников с окладом больше 5500;

(б) найти все отделы, в которых есть сотрудники с окладом больше 8000;

(в) составить список должностей и получаемых по ним окладов;

(г) составить список сотрудников торгового отдела, получающих зарплату от 6000 до 6500 и работающих не на третьем этаже;

(д) составить список комнат, где все сотрудники получают оклад меньше 7500.

139. Определить, какие из приведённых в примере 42 на противоположной странице ограничений целостности выполняются для состояния базы данных из примера 33 на стр. 163.

140. Написать формулы, выражающие следующие ограничения целостности для базы данных из примера 33 на стр. 163, определить, какие из них выполняются для приведённого её состояния:

(а) в отношении Комнаты набор атрибутов (НомерСотрудника, НомерКомнаты) является ключом;

(б) для каждого человека из отношения Сотрудники в отношении Комнаты определено его место работы;

(в) номера всех комнат на втором этаже больше 10, но меньше 20, а номера всех комнат на третьем этаже больше 20.

Глава 9

Ориентированные графы

Краткое содержание: ориентированные графы, представление графа с помощью матрицы смежности и списков смежности, граф достижимости (транзитивное замыкание), сильная достижимость, компоненты сильной связности, база ориентированного графа.

Ключевые слова: ориентированный граф, вершина графа, ребро графа, изоморфизм графов, путь в графе, цикл, достижимость, матрица смежности, списки смежности, компонента сильной связности, база графа.

§ 9.1. Основные определения

Мы часто сталкиваемся с задачами, когда в условиях задано множество объектов, между некоторыми из которых могут существовать определённые связи. Иначе говоря, на множестве этих объектов имеется какое-то бинарное отношение. Объекты можно изобразить точками (вершинами), а связи — линиями или стрелками (рёбрами), соединяющими соответствующие точки. В результате получится рисунок, называемый графом.

Историю теории графов исчисляют с 1736 г., когда Л. Эйлер исследовал «задачу о кёнигсбергских мостах»: построить в графе циклический путь, проходящий по одному разу через каждое ребро. В середине XIX в. Р. Гамильтон заинтересовался задачей построения циклического пути, проходящего по одному разу через каждую вершину графа. К тому же времени относится использование графов для анализа электрических цепей (Г. Кирхгоф) и химических молекул (А. Кэли). Развитие современной теории графов относится к 30-м годам XX в. Они нашли многочисленные применения в электротехнике, электронике, биологии, экономике, программировании и в других областях.

В этой и следующих главах мы рассмотрим основные понятия теории графов и несколько широко используемых в различных приложениях типовых задач таких, как представления графов, отношение достижимости и транзитивное замыкание графа, компоненты сильной связности ориентированного графа и его базы, планарность и раскраски графов, деревья и их обходы, минимальные остовные деревья, поиск в глубину и поиск кратчайших путей. Значительное внимание будет уделено алгоритмическим процедурам для решения указанных задач.

В главах 13, 14 и 15 мы покажем, как применяются графы для реализации булевых функций и представления конечных автоматов.

Приведём основные определения.

Определение 67 (Ориентированный граф)**.** *Ориентированный граф — это пара (V, E), где V — конечное множество вершин (говорят также узлов или точек) графа, а E — бинарное отношение на V, то есть некоторое множество упорядоченных пар вершин. Элементы E называют рёбрами (говорят ещё дугами, стрелками или связями).*

Для ребра $e = (u, v) \in E$ вершина u называется началом e, а вершина v — концом e, в этом случае также говорят, что ребро e ведёт из u в v. Ребро вида (u, u) называется петлёй.

Полустепень исхода вершины — это количество исходящих из неё рёбер, а полустепень захода — это количество

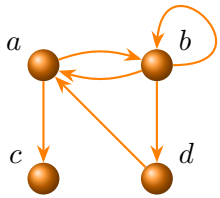


Рис. 18: Ориентированный граф из примера 43.

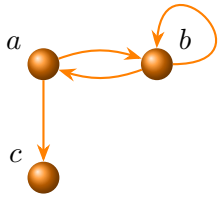


Рис. 19: Подграф.

входящих в данную вершину рёбер. Если полустепень захода (соответственно исхода) равна нулю, то вершина называется *истоком* *(соответственно* *стоком*).

Пример 43. *Рассмотрим ориентированный граф* $\mathfrak{G} = (V, E)$, *который изображён на рис. 18. В этом графе* $V = \{a, b, c, d\}$, $E = \{(a, b), (b, a), (a, c), (b, b), (b, d), (d, a)\}$. *Ребро* (b, b) *является петлёй, полустепени исхода и захода вершин* a *и* b *равны 2, для вершины* d *они обе равны 2, а для вершины* c *полустепени исхода и захода равны* 0 *и* 1 *соответственно.*

Во многих задачах в заданном графе нужно выделить некоторую часть, обладающую тем или иным свойством.

Определение 68 (Подграф)**.** *Граф* $\mathfrak{G}_1 = (V_1, E_1)$ *называется* *подграфом* *графа* $\mathfrak{G} = (V, E)$, *если* $V_1 \subseteq V$ *и* $E_1 \subseteq E$.

На рис. 19 изображён подграф графа из примера 43.

Во многих случаях с вершинами и рёбрами графов требуется связать некоторую дополнительную информацию. Обычно она представляется с помощью функций разметки вершин и рёбер.

Определение 69 (Разметка графа). *Размеченный граф — это граф $\mathfrak{G} = (V, E)$, снабжённый одной или двумя функциями разметки вида: $\ell : V \to M$ и $c : E \to L$, где M и L — множества меток вершин и рёбер соответственно.*

В качестве множества меток рёбер L часто выступают числа, задающие «веса», «длины», «стоимости» рёбер. Графы с такой разметкой часто называют взвешенными.

Важным частным случаем разметки является такой.

Определение 70 (Упорядоченный граф). *Упорядоченный граф — это размеченный граф $\mathfrak{G} = (V, E)$, в котором рёбра, выходящие из каждой вершины $v \in V$, упорядочены, то есть помечены номерами $1, \ldots, k_v$, где k_v — полустепень исхода вершины v: $k_v = |\{w : (v, w) \in E\}|$.*

Особенно часто упорядочение рёбер встречается при рассмотрении ориентированных деревьев (глава 11).

Во многих случаях естественно не различать графы, отличающиеся лишь именами (порядком) вершин.

Определение 71 (Изоморфизм графов). *Два графа $\mathfrak{G}_1 = (V_1, E_1)$ и $\mathfrak{G}_2 = (V_2, E_2)$ называются изоморфными, если между их вершинами существует взаимно однозначное соответствие $\varphi : V_1 \to V_2$ такое, что для всех вершин $u, v \in V_1$ ребро $(u, v) \in E_1$ существует тогда и только тогда, когда существует ребро $(\varphi(u), \varphi(v)) \in E_2$.*

Если графы размечены, то для изоморфизма требуется также совпадение меток соответствующих вершин и рёбер: $\ell_1(v) = \ell_2(\varphi(v))$, $c_1((u, v)) = c_2((\varphi(u), \varphi(v)))$.

Можно считать, что изоморфные графы — это один и тот же граф, только, возможно, по разному изображённый. Например, граф на рис. 18 на противоположной странице изоморфен графу на рис. 20 на следующей странице: $\varphi(a) = u$, $\varphi(b) = x$, $\varphi(c) = z$, $\varphi(d) = y$.

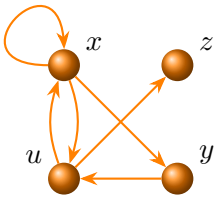


Рис. 20: Граф, изоморфный графу на рис. 18.

Многие приложения графов связаны с изучением путей между их вершинами.

Определение 72 (Путь в графе)**.** *Путь в ориентированном графе — это последовательность вершин* $(v_0, v_1, \ldots, v_n)$ *такая, что* $(v_i, v_{i+1}) \in E$ *для* $i = 0, \ldots, n-1$*. Этот путь ведёт из начальной вершины* v_0 *в конечную вершину* v_n*. Говорят, что этот путь проходит по вершинам* $v_0, v_1, \ldots, v_n$ *и по рёбрам* $(v_0, v_1), \ldots, (v_{n-1}, v_n)$*. Число* n *называется длиной пути.*

Если из вершины v_0 *в вершину* v_n *есть путь, то* v_n *достижима из* v_0*. В частности, каждая вершина достижима сама из себя путём длины* 0*.*

Циклом в ориентированном графе называется путь, в котором начальная вершина совпадает с конечной и который содержит хотя бы одно ребро.

Путь (цикл) называется простым, если все вершины в нём, кроме, быть может, первой и последней, различны.

Если в графе нет циклов, то он называется ациклическим.

В графе, который изображён на рис. 18 на стр. 180, есть, например, пути (b, d, a, c) длины 3 и (a, b, d, a, b, b) длины 5. Первый из них является простым, а второй — нет. Примерами циклов будут (a, b, d, a) и (a, b, a, b, d, a), из которых только первый будет простым. Циклы, которые отличаются только исходной вершиной, как правило, не раз-

личают. То есть первый из циклов можно записать и так: (b, d, a, b), и так: (d, a, b, d).

Следующее утверждение непосредственно следует из определений.

Лемма 55. *Если в ориентированном графе $\mathfrak{G}$ имеется путь (цикл) из вершины u в вершину v, $u \neq v$, то в нём имеется и простой путь (цикл) из u в v.*

Доказательство. Рассмотрим самый короткий из путей из u в v: $(u_0 = u, \ldots, u_n = v)$. Если бы этот путь не являлся простым, например, $u_i = u_j$ при $i < j$, то его можно было бы сократить: $(u_0 = u, \ldots, u_i, u_{j+1}, \ldots, u_n = v)$ и получить путь из u в v короче исходного. Это приводит к противоречию, так как путь изначально был выбран самым коротким.

Аналогично для циклов. □

Следствие 56. *Если вершина v достижима из вершины u в графе $\mathfrak{G}$, то существует путь из u в v длины не больше $n - 1$, где n — количество вершин.*

Доказательство. Если $u = v$, то длина пути равна нулю. Если $u \neq v$, то простой путь $(u, \ldots, v)$ содержит не больше $n - 1$ рёбер. □

Замечание 8 (О мультиграфах)**.** Часто встречаются объекты, похожие на графы, отличие которых заключается в том, что между двумя вершинами могут существовать несколько одинаково направленных рёбер. Типичный пример: между двумя транспортными узлами может существовать несколько разных путей. Такие объекты называют мультиграфами.

Точное определение этих объектов следующее: мультиграф $\mathfrak{G}$ — это тройка (V, E, θ), где V и E — это непересекающиеся множества вершин и рёбер соответственно, а $\theta : E \to V \times V$ — функция, определяющая направление каждого ребра. Таким образом, ребро e ведёт из вершины u в вершину v, если $\theta(e) = (u, v)$. Рёбра e, для которых существуют «параллельные»: $\theta(e) = \theta(e')$ при $e \neq e'$, называются кратными.

«Задача о кёнигсбергских мостах», упомянутая в начале главы, приводит именно к мультиграфу (неориентированному), изображённому на рис. 34 на стр. 210.

Определение графа является частным случаем мультиграфа, когда $E \subseteq V \times V$, а θ — тождественная функция на E, которую мы для графов не указываем в силу её единственности.

При определении путей и циклов в мультиграфах указывать последовательность вершин уже недостаточно, так как при наличии кратных рёбер путь определяется неоднозначно. В этом случае между вершинами указывают и рёбра, которые используются для обхода:

$$(v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n).$$

Многие результаты о графах переносятся на мультиграфы либо вообще без изменений, либо с минимальными изменениями, учитывающими наличие нескольких рёбер между вершинами. Например, наличие кратных рёбер никак не влияет на свойства связности и достижимости, на планарность графа и хроматическое число. Для мультиграфов остаются справедливыми формула Эйлера и критерий эйлеровости. Деревья не могут быть мультиграфами по определению. Алгоритмы из главы 12 можно применять к мультиграфам либо непосредственно, либо с небольшими изменениями.

В тех местах, где свойства мультиграфов отличаются от свойств обычных графов, мы делаем специальные замечания.

§ 9.2. Представления графов

Из определения графа следует, что каждый граф $\mathfrak{G} = (V, E)$ можно задать, непосредственно перечислив его множество вершин V и множество рёбер E. Однако такое представление неудобно для решения многих задач о графах. Например, чтобы проверить наличие ребра между двумя вершинами, придётся, вообще говоря, просмотреть всё множество E. Хорошее представление, по крайней мере, должно позволить легко переходить от вершины к её соседу и перечислять всех её соседей.

В этом параграфе мы рассмотрим два разных способа представления графов, которые более эффективны при решении типичных для теории графов задач.

Определение 73 (Матрица смежности). *Матрицей смежности ориентированного графа $\mathfrak{G} = (V, E)$ с множеством вершин $V = \{v_1, \ldots, v_n\}$ называется булева матрица $A_{\mathfrak{G}}$ размера $n \times n$ с*

элементами

$$a_{i,j} = \begin{cases} 1, & \text{если } (v_i, v_j) \in E, \\ 0 & \text{в противном случае.} \end{cases}$$

Это представление позволяет легко проверять наличие рёбер между заданными парами вершин. Для поиска всех соседей, в которые ведут рёбра из вершины v_i, необходимо просмотреть соответствующую ей i-ю строку матрицы $A_{\mathfrak{G}}$, а чтобы найти вершины, из которых рёбра идут в v_i, необходимо просмотреть её i-й столбец. Требуемая для $A_{\mathfrak{G}}$ память — по порядку n^2 битов — не может быть существенно уменьшена для графов, у которых «много» рёбер. Но для разреженных графов с числом рёбер значительно меньшим по порядку n^2 в матрице смежности много «ненужных» нулей. Для таких графов более эффективными могут оказаться другие представления.

Определение 74 (Списки смежности)**.** *Пусть $\mathfrak{G} = (V, E)$ — ориентированный граф, $v \in V$. Список смежности L_v для вершины v — это список, содержащий по одному разу все вершины, в которые из v ведёт ребро, то есть $L_v = (w_1, \ldots, w_k)$, где k — полустепень исхода v, а $(v, w_1), \ldots, (v, w_k)$ — все выходящие из v рёбра.*

Представление графа $\mathfrak{G} = (V, E)$ с n вершинами $V = \{v_1, \ldots, v_n\}$ с помощью списков смежности — это функция L, определённая на множестве V, такая, что $L(v) = L_v$.

Иными словами, при представлении графа с вершинами $v_1, \ldots, v_n$ в виде списков смежности мы должны хранить списки смежности $L_1, \ldots, L_n$ для соответствующих вершин.

Заметим, что каждое ребро (u, v) даёт в точности один элемент из списков смежности — вершину v в списке L_u. Следовательно, размер этого представления примерно равен сумме числа вершин и числа рёбер графа. Оно позволяет легко переходить по рёбрам от вершины к её соседям. В программах списки смежности представляются списковыми структурами, которые легко реализуются во всех языках программирования.

Пример 44. *Рассмотрим следующий граф $\mathfrak{G} = (V, E)$:*

$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$;

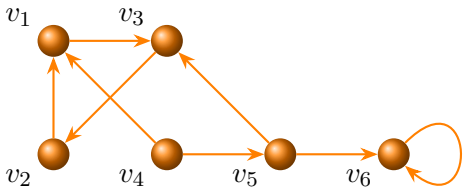


Рис. 21: Граф из примера 44.

$$E = \{e_1 = (v_1, v_3), e_2 = (v_3, v_2), e_3 = (v_4, v_1), e_4 = (v_4, v_5),$$
$$e_5 = (v_5, v_3), e_6 = (v_5, v_6), e_7 = (v_6, v_6), e_8 = (v_2, v_1)\}.$$

Он показан на рис. 21.

Построим для него определённые выше представления.

1) *Матрица смежности.*

$$A_{\mathfrak{G}} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

2) *Списки смежности.*

$$L_{v_1} = (v_3); \quad L_{v_3} = (v_2); \quad L_{v_5} = (v_3, v_6);$$
$$L_{v_2} = (v_1); \quad L_{v_4} = (v_1, v_5); \quad L_{v_6} = (v_6).$$

§ 9.3. Граф достижимости

Один из первых вопросов, возникающих при изучении графов, — это вопрос о существовании путей между заданными или всеми парами вершин. Ответом на этот вопрос является введённое выше отношение достижимости на вершинах графа $\mathfrak{G} = (V, E)$: вершина w достижима из вершины v, если $v = w$ или в $\mathfrak{G}$ есть путь из v в w.

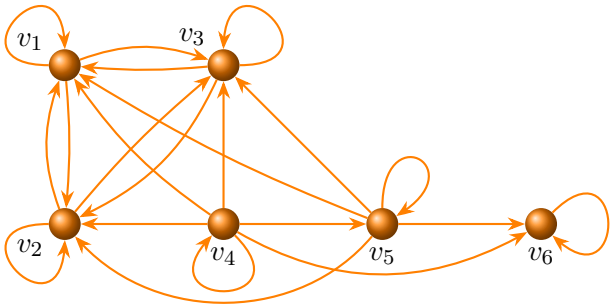


Рис. 22: Граф достижимости из примера 45.

Иначе говоря, отношение достижимости является рефлексивным и транзитивным замыканием отношения E.

Рассмотрим вначале вопрос о построении отношения достижимости. Определим для каждого графа его граф достижимости (называемый иногда также графом транзитивного замыкания), рёбра которого соответствуют путям исходного графа.

Определение 75 (Граф достижимости)**.** *Пусть* $\mathfrak{G} = (V, E)$ *— ориентированный граф. Граф достижимости* $\mathfrak{G}^* = (V, E^*)$ *для* $\mathfrak{G}$ *имеет то же множество вершин* V *и следующее множество рёбер:*

$$E^* = \{(u, v) : \text{в графе } \mathfrak{G} \text{ вершина } v \text{ достижима из вершины } u\}.$$

Пример 45. *Рассмотрим граф* $\mathfrak{G}$ *из примера 44 на стр. 185 (рис. 21 на противоположной странице). Тогда можно проверить, что граф достижимости* $\mathfrak{G}^*$ *для* $\mathfrak{G}$ *выглядит так, как изображено на рис. 22.*

Каким образом по графу $\mathfrak{G}$ можно построить его граф достижимости $\mathfrak{G}^*$? Один способ заключается в том, чтобы для каждой вершины графа $\mathfrak{G}$ определить множество достижимых из неё вершин, последовательно добавляя в него вершины, достижимые из неё путями длины 0, 1, 2 и т. д.

Опишем другой способ, основанный на использовании матрицы смежности $A_{\mathfrak{G}}$ графа $\mathfrak{G}$ и булевых операций. Если $\mathfrak{G}$ содержит n вершин: $v_1, \ldots, v_n$, то $A_{\mathfrak{G}}$ — это булева матрица размера $n \times n$.

Над булевыми матрицами можно выполнять покомпонентные операции $\vee$ и $\wedge$. Обозначим через I_n единичную матрицу размера $n \times n$:

$$I_n = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \ldots & 1 \end{pmatrix}.$$

Положим $\tilde{A} = A_{\mathfrak{G}} \vee I_n$. Рассмотрим булевы степени матрицы $\tilde{A}$:

$$\tilde{A}^0 = I_n, \quad \tilde{A}^1 = \tilde{A}, \quad \tilde{A}^2 = \tilde{A} \times \tilde{A}, \quad \ldots, \quad \tilde{A}^{k+1} = \tilde{A}^k \times \tilde{A}.$$

Булево произведение матриц вычисляется по тем же правилам, что и обычное, только вместо сложения и умножения используются дизъюнкция и конъюнкция соответственно:

$$(A \times B)_{i,j} = \bigvee_{\ell=1}^{n} (A_{i,\ell} \wedge B_{\ell,j}).$$

Процедура построения $\mathfrak{G}^*$ основана на следующем утверждении.

Лемма 57. *В графе $\mathfrak{G}$ из вершины v_i имеется путь в вершину v_j длины не больше k тогда и только тогда, когда $(\tilde{A}^k)_{i,j} = 1$.*

Доказательство. Используем индукцию по k.

Базис индукции. При $k = 0$ и $k = 1$ утверждение справедливо по определению $\tilde{A}^0$ и $\tilde{A}^1$.

Индукционный шаг. Пусть лемма справедлива для k. Покажем, что она остаётся справедливой и для $k + 1$. По определению $\tilde{A}^{k+1}$ имеем:

$$(\tilde{A}^{k+1})_{i,j} = (\tilde{A}^k \times \tilde{A})_{i,j} = \bigvee_{\ell=1}^{n} ((\tilde{A}^k)_{i,\ell} \wedge \tilde{A}_{\ell,j}).$$

Предположим, что в графе $\mathfrak{G}$ из v_i в v_j имеется путь длины не больше $k + 1$. Рассмотрим кратчайший из таких путей: $(v_i, \ldots, v_r, v_j)$.

Если его длина не превосходит k, то по предположению индукции $(\tilde{A}^k)_{i,j} = 1$. Кроме того, $\tilde{A}_{j,j} = 1$. Поэтому $(\tilde{A}^{(k)})_{i,j} \wedge \tilde{A}_{j,j} = 1$ и, следовательно, $(\tilde{A}^{k+1})_{i,j} = 1$. Если длина пути $(v_i, \ldots, v_r, v_j)$ равна $k+1$, то длина пути $(v_i, \ldots, v_r)$ равна k. По предположению индукции $(\tilde{A}^k)_{i,r} = 1$. Так как $(v_r, v_j) \in E$, то $\tilde{A}_{r,j} = 1$. Поэтому $(\tilde{A}^{(k)})_{i,r} \wedge \tilde{A}_{r,j} = 1$ и $(\tilde{A}^{k+1})_{i,j} = 1$.

Обратно, если $(\tilde{A}^{k+1})_{i,j} = 1$, то хотя бы для одного ℓ конъюнкция $(\tilde{A}^{(k)})_{i,\ell} \wedge \tilde{A}_{\ell,j}$ должна быть равна 1, что означает $(\tilde{A}^{(k)})_{i,\ell} = 1$ и $\tilde{A}_{\ell,j} = 1$. Из первого по индукционному предположению получаем, что существует путь $(v_i, \ldots, v_\ell)$ длины не больше k. Второе значит, что $\ell = j$, тогда $(v_i, \ldots, v_\ell = v_j)$ является искомым путём, или в графе $\mathfrak{G}$ есть ребро (v_ℓ, v_j), тогда получим путь $(v_i, \ldots, v_\ell, v_j)$ длины не больше $k+1$. □

Из следствия 56 на стр. 183 и леммы 57 на предыдущей странице непосредственно получаем

Следствие 58. *Пусть $\mathfrak{G} = (V, E)$ — ориентированный граф с n вершинами, а $\mathfrak{G}^*$ — его граф достижимости. Тогда $A_{\mathfrak{G}^*} = \tilde{A}^{n-1}$.*

Доказательство. Из следствия 56 на стр. 183 получаем, что, длина кратчайшего пути (если он вообще есть) из u в $v \neq u$ не превосходит $n-1$. А по лемме 57 на предыдущей странице наличие таких путей представлено единицами в матрице $\tilde{A}^{n-1}$. □

Заметим, что из леммы можно вывести ещё один факт.

Следствие 59. *$\tilde{A}^k = \tilde{A}^{n-1}$ для $k \geqslant n$.*

Доказательство. Получаем такую цепочку эквивалентностей:

$$(\tilde{A}^k)_{ij} = 1 \iff$$
$$\iff \text{существует путь из } v_i \text{ в } v_j \text{ длины не больше } k \iff$$
$$\iff \text{существует путь из } v_i \text{ в } v_j \text{ длины не больше } n-1 \iff$$
$$\iff (\tilde{A}^{n-1})_{i,j} = 1.$$

Средняя эквивалентность получается из следствия 56 на стр. 183, две другие — из леммы 57 на предыдущей странице. □

Таким образом, процедура построения матрицы смежности $A_{\mathfrak{G}^*}$ графа достижимости для $\mathfrak{G}$ сводится к возведению матрицы $\tilde{A}$ в степень $n-1$. Сделаем несколько замечаний, позволяющих упростить эту процедуру.

1) Для возведения матрицы $\tilde{A}$ в произвольную степень n достаточно выполнить не более $[\log_2 n]+1$ возведений в квадрат:

$$\tilde{A} \Rightarrow \tilde{A}^2 \Rightarrow \tilde{A}^{2^2} \Rightarrow \ldots \Rightarrow \tilde{A}^{2^k},$$

где k — это наименьшее число такое, что $2^k \geqslant n-1$.

2) Иногда количество возведений в квадрат можно ещё сократить: если $A^i = (A^i)^2$, то, очевидно, и при дальнейших возведениях в квадрат матрица не изменится, следовательно, процесс можно завершить.

3) Так как на диагонали в матрице $\tilde{A}$ стоят единицы, то для любых $i < j$ все единицы матрицы $\tilde{A}^i$ сохраняются в матрице $\tilde{A}^j$, в частности, и в матрице $(\tilde{A}^i)^2$.

4) Если при вычислении элемента $a^{(2)}_{i,j}$ матрицы $(\tilde{A}^k)^2$ по стандартной формуле

$$(\tilde{A}^{2k})_{i,j} = \bigvee_{\ell=1}^{n} ((\tilde{A}^k)_{i,\ell} \wedge (\tilde{A}^k)_{\ell,j}).$$

обнаруживается такое ℓ, что $(\tilde{A}^k)_{i,\ell} = (\tilde{A}^k)_{\ell,j} = 1$, то и для всей дизъюнкции $(\tilde{A}^{2k})_{i,j} = 1$. Поэтому остальные слагаемые можно не рассматривать.

Пример 46. *Рассмотрим в качестве примера вычисление матрицы графа достижимости $A_{\mathfrak{G}^*}$ для графа $\mathfrak{G}$, представленного на рис. 21 на стр. 186. В этом случае*

$$\tilde{A} = A_{\mathfrak{G}} \vee I_6 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Так как у $\mathfrak{G}$ *имеется* 6 *вершин, то* $A_{\mathfrak{G}^*} = \tilde{A}^5$. *Вычислим эту матрицу:*

$$\tilde{A}^2 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad \tilde{A}^4 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

и $\tilde{A}^5 = \tilde{A}^8 = \tilde{A}^4 \times \tilde{A}^4 = \tilde{A}^4$ *(последнее равенство нетрудно проверить). Таким образом,*

$$A_{\mathfrak{G}^*} = \tilde{A}^4 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Как видим, эта матрица действительно задаёт граф $\mathfrak{G}^*$, *представленный на рис.* *22 на стр. 187*.

§ 9.4. Взаимная достижимость, компоненты сильной связности и базы графа

Для ориентированных графов достижимость, вообще говоря, не должна быть симметричным отношением. Симметричной является взаимная достижимость.

Определение 76 (Взаимная достижимость)**.** *Вершины* v *и* w *ориентированного графа* $\mathfrak{G} = (V, E)$ *называются взаимно достижимыми, если в* $\mathfrak{G}$ *есть путь из* v *в* w *и путь из* w *в* v.

Ясно, что отношение взаимной достижимости является рефлексивным, симметричным и транзитивным и, следовательно, эквивалентностью на множестве вершин графа.

Определение 77 (Компоненты сильной связности)**.** *Классы эквивалентности по отношению взаимной достижимости называются компонентами сильной связности графа.*

Ориентированный граф называется сильно связным, если каждая пара вершин в нём соединена путём.

Следовательно, сильно связный граф состоит всего из одной компоненты сильной связности.

По аналогии с графом достижимости определим граф сильной достижимости.

Определение 78 (Граф сильной достижимости)**.** *Пусть дан ориентированный граф $\mathfrak{G} = (V, E)$. Граф сильной достижимости $\mathfrak{G}^*_* = (V, E^*_*)$ для $\mathfrak{G}$ имеет то же множество вершин V и следующее множество рёбер:*

$$E^*_* = \{(u, v) : \text{в графе } \mathfrak{G} \text{ вершины } v \text{ и } u \text{ взаимно достижимы}\}.$$

По матрице смежности графа достижимости $A_{\mathfrak{G}^*}$ легко построить матрицу $A_{\mathfrak{G}^*_*}$ графа сильной достижимости. Действительно, из определений достижимости и сильной достижимости непосредственно следует, что для всех пар (i, j), $1 \leqslant i, j \leqslant n$, значение элемента матрицы $(A_{\mathfrak{G}^*_*})_{i,j}$ равно 1 тогда и только тогда, когда оба элемента $(A_{\mathfrak{G}^*})_{i,j}$ и $(A_{\mathfrak{G}^*})_{j,i}$ равны 1. Значит, $A_{\mathfrak{G}^*_*} = A_{\mathfrak{G}^*} \wedge A^T_{\mathfrak{G}^*}$, где T означает транспонирование матрицы.

По матрице $A_{\mathfrak{G}^*_*}$ можно выделить компоненты сильной связности графа $\mathfrak{G}$ следующим образом:

1) Поместим в компоненту K_1 вершину v_1 и все такие вершины v_j, что $(A_{\mathfrak{G}^*_*})_{1,j} = 1$.
2) Пусть уже построены компоненты $K_1, \ldots, K_i$ и v_k — это вершина с минимальным номером, ещё не попавшая в компоненты. Тогда поместим в компоненту K_{i+1} вершину v_k и все такие вершины v_j, что $(A_{\mathfrak{G}^*_*})_{k,j} = 1$.
3) Повторяем шаг 2) до тех пор, пока все вершины не будут распределены по компонентам.

Пример 47. *Для графа $\mathfrak{G}$ из примера 44 на стр. 185 (рис. 21 на стр. 186) по матрице $A_{\mathfrak{G}^*}$ получаем следующую матрицу графа сильной достижимости*

$$A_{\mathfrak{G}^*_*} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Используя описанную выше процедуру, находим, что вершины графа $\mathfrak{G}$ разбиваются на 4 компоненты сильной связности: $K_1 = \{v_1, v_2, v_3\}$, $K_2 = \{v_4\}$, $K_3 = \{v_5\}$ и $K_4 = \{v_6\}$.

На множестве компонент сильной связности также определим отношение достижимости.

Определение 79 (Достижимость компонент, минимальная компонента)**.** *Пусть K и K' — компоненты сильной связности графа $\mathfrak{G}$. Компонента K достижима из K', если существуют такие две вершины $u \in K$ и $v \in K'$, что u достижима из v. Компонента K строго достижима из K', если при этом $K \neq K'$.*

Компонента K называется минимальной, если она не является строго достижимой ни из какой компоненты.

Так как все вершины в одной компоненте взаимно достижимы, то нетрудно понять, что отношения достижимости и строгой достижимости на компонентах не зависят от выбора вершин $u \in K$ и $v \in K'$.

Из определения легко выводится следующая характеристика строгой достижимости.

Лемма 60. *Отношение строгой достижимости на компонентах сильной связности является отношением частичного порядка, то есть оно антирефлексивно, антисимметрично и транзитивно.*

Доказательство. См. задачу 151 на стр. 196. □

Это отношение тоже можно представлять в виде ориентированного графа, вершинами которого являются компоненты сильной связности, а ребро (K', K) означает, что компонента K строго достижима из K'.

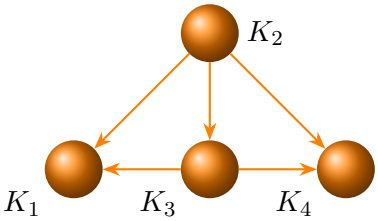


Рис. 23: Граф строгой достижимости на компонентах $\mathfrak{G}$

Пример 48. *На рис. 23 показан граф строгой достижимости компонент для графа из предыдущего примера. В данном случае имеется одна минимальная компонента K_2.*

Во многих приложениях ориентированный граф представляет собой сеть распространения некоторого ресурса: продукта, товара, информации и т. д. В таких случаях естественно возникает задача поиска минимального множества таких точек (вершин), из которых этот ресурс может быть доставлен в любую точку сети.

Определение 80 (Порождающее множество, база)**.** *Пусть дан ориентированный граф $\mathfrak{G} = (V, E)$. Подмножество вершин $W \subseteq V$ называется порождающим, если из вершин W можно достичь все вершины графа.*

Подмножество вершин $W \subseteq V$ называется базой графа, если оно является порождающим, но никакое его собственное подмножество порождающим не является.

Следующая теорема позволяет легко находить все базы графа.

Теорема 61. *Пусть $\mathfrak{G} = (V, E)$ — ориентированный граф. Подмножество вершин $W \subseteq V$ является базой $\mathfrak{G}$ тогда и только тогда, когда W содержит в точности по одной вершине из каждой минимальной компоненты сильной связности $\mathfrak{G}$.*

Доказательство. Заметим вначале, что каждая вершина графа достижима из вершины, принадлежащей некоторой минимальной компоненте. Поэтому множество вершин W, содержащих по одной вершине из каждой минимальной компоненты, является порожда-

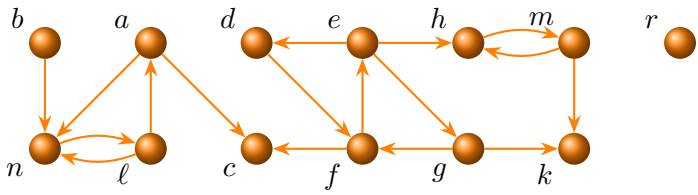


Рис. 24: Граф из примера 49.

ющим. Если же из W удалить любую вершину, то оно перестаёт быть таковым, так как вершины из соответствующей минимальной компоненты становятся недостижимы. Поэтому W является базой.

Обратно, если W является базой, то оно обязано включать хотя бы по одной вершине из каждой минимальной компоненты, иначе вершины такой минимальной компоненты окажутся недоступны. Никаких других вершин W содержать не может, так как каждая из них достижима из уже включённых вершин. □

Из этой теоремы вытекает следующая процедура построения одной или перечисления всех баз графа $\mathfrak{G}$:

1) найти все компоненты сильной связности $\mathfrak{G}$;
2) определить отношение достижимости на них и выделить минимальные компоненты;
3) породить одну (или все) базы графа, выбирая по одной вершине из каждой минимальной компоненты.

Пример 49. *Найдём все базы графа $\mathfrak{G}$, показанного на рис. 24.*

На первом этапе найдём компоненты сильной связности $\mathfrak{G}$: $K_1 = \{a, n, \ell\}$, $K_2 = \{b\}$, $K_3 = \{c\}$, $K_4 = \{d, e, f, g\}$, $K_5 = \{h, m\}$, $K_6 = \{k\}$, $K_7 = \{r\}$. На втором этапе строим граф строгой достижимости на этих компонентах (рис. 25 на следующей странице). Далее определяем минимальные компоненты: $K_2 = \{b\}$, $K_4 = \{d, e, f, g\}$ и $K_7 = \{r\}$. Наконец перечисляем все четыре базы $\mathfrak{G}$: $B_1 = \{b, d, r\}$, $B_2 = \{b, e, r\}$, $B_3 = \{b, f, r\}$ и $B_1 = \{b, g, r\}$.

Задачи

141. Найти общее количество графов с вершинами $v_1, \ldots, v_n$. ▼

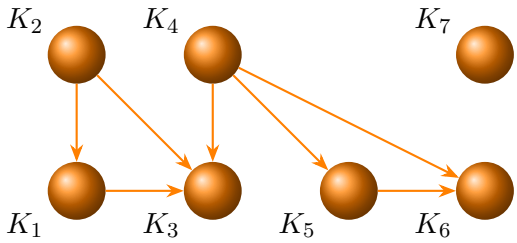


Рис. 25: Достижимость компонент из примера 49.

142. Доказать, что в ориентированном графе без циклов есть хотя бы один исток и хотя бы один сток. ▼

143. Пусть $\mathfrak{G} = (V, E)$ — ациклический граф. Доказать, что вершины можно пронумеровать $V = \{v_1, \ldots, v_n\}$ так, что если $(v_i, v_j) \in E$, то $i < j$. ▼

144. Пусть $\mathfrak{G} = (V, E)$ — граф, $V = \{v_1, \ldots, v_n\}$, A — матрица смежности, $W \subseteq V$, а вектор-столбец w содержит нули и единицы, причём $w_i = 1$ означает $v_i \in W$. Определить смысл булевых произведений Aw и $w^T A$. ▼

145. Т у р н и р — это граф без петель, в котором любые две вершины соединены в точности одним ребром. Доказать, что в турнире из вершины v с наибольшей полустепенью исхода в любую другую ведёт путь длины не более двух. ▼

146. Индукцией по количеству вершин доказать следующее утверждение: в турнире есть путь, проходящий через все вершины. ▼

147. Чемпионат организован по круговой системе (каждая команда играет по одному матчу с каждой, победитель определяется по количеству выигранных матчей). Доказать, что если победитель чемпионата проиграл команде A, то A проиграла некоторой команде B, которая, в свою очередь, проиграла победителю. ▼

148. Определить, что представляет собой граф достижимости для

(а) графа с n вершинами и пустым множеством рёбер;

(б) графа с n вершинами: $V = \{v_1, \ldots, v_n\}$, рёбра которого образуют цикл: $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n), (v_n, v_1)\}$. ▼

149. Вычислить матрицу графа достижимости $A_{\mathfrak{G}^*}$ для следующего графа

$$\mathfrak{G} = (\{a, b, c, d, e, f, g\}, \{(a, b), (b, a), (a, c), (b, d), (e, d), (d, f), (f, c), (c, f), (g, e)\})$$

и построить соответствующий ей граф достижимости. Найти все базы графа $\mathfrak{G}$. ▼

150. Построить для изображённых на рис. 26 на противоположной странице ориентированных графов $\mathfrak{G}_1$ и $\mathfrak{G}_2$ их матрицы смежности и списки смежности. Вычислить матрицы достижимости и построить соответствующие графы достижимости. ▼

151. Доказать лемму 60 на стр. 193. ▼

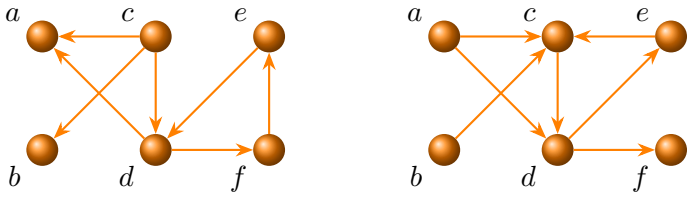


Рис. 26: Графы $\mathfrak{G}_1$ и $\mathfrak{G}_2$.

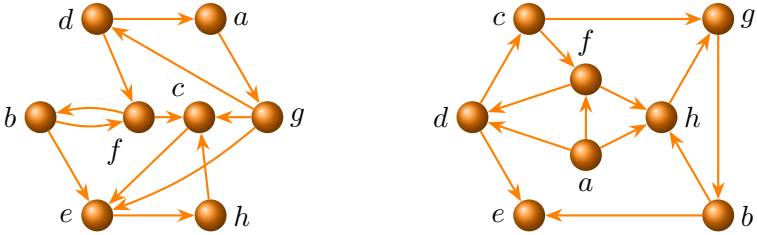


Рис. 27: Графы $\mathfrak{G}_1$ и $\mathfrak{G}_2$.

152. Пусть граф $\mathfrak{G} = (V, E)$ задан своей матрицей смежности:

$$A_{\mathfrak{G}} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Построить граф достижимости $\mathfrak{G}^* = (V, E^*)$ для $\mathfrak{G}$ и определить, сколько в нём новых рёбер, то есть чему равна разность $|E^*| - |E|$. ▼

153. Доказать, что граф без петель ациклический тогда и только тогда, когда все компоненты сильной связности содержат по одному элементу. ▼

154. Показать, что для каждого положительного n существует граф без циклов с n вершинами и $\frac{n(n-1)}{2}$ рёбрами. Доказать, что в любом графе, в котором рёбер больше, обязательно есть цикл. ▼

155. Доказать, что в ориентированном графе без циклов существует единственная база, состоящая из всех истоков. ▼

156. Для графов $\mathfrak{G}_1$ и $\mathfrak{G}_2$ на рис. 27 определить компоненты сильной связности и отношение строгой достижимости на них. ▼

Глава 10

Неориентированные графы

Краткое содержание: неориентированные графы, компоненты связности, плоские графы, формула Эйлера для плоских графов, эйлеровы и полуэйлеровы графы, раскраска графов, двудольные графы.

Ключевые слова: неориентированный граф, компонента связности, связный граф, плоский граф, формула Эйлера, многогранник, эйлеров цикл, эйлеров граф, эйлеров путь, полуэйлеров граф, гамильтонов цикл, гамильтонов граф, раскраска графа, хроматическое число, двудольный граф.

§ 10.1. Основные определения

До сих пор мы считали, что у каждого ребра есть направление. На практике часто встречаются и такие задачи, в которых направления рёбер отсутствуют.

Рассмотрим такой пример. Пусть вершинами графа являются перекрёстки, а рёбрами — дороги, их соединяющие. Если не принимать во внимание возможность одностороннего движения, то в таком графе рёбра не имеют явно определённых направлений.

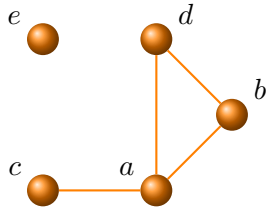


Рис. 28: Неориентированный граф $\mathfrak{G}$.

Определение 81 (Неориентированный граф)**.** *Неориентированный граф $\mathfrak{G} = (V, E)$ — это ориентированный граф, у которого для каждого ребра $(u, v) \in E$ имеется противоположное ребро $(v, u) \in E$, то есть отношение E симметрично.*

Пара рёбер $\{(u, v), (v, u)\}$ называется неориентированным ребром. Для его задания можно использовать обозначение для множества концов: $\{u, v\}$, но чаще используют обычное указание одной из пар в круглых скобках: (u, v) или (v, u), автоматически подразумевая наличие второй пары.

Если $e = (u, v) \in E$, то вершины u и v называются смежными в $\mathfrak{G}$, а ребро e и эти вершины называются инцидентными. Степенью вершины v в неориентированном графе называется количество инцидентных v рёбер, петля у вершины учитывается два раза. Обозначается степень вершины v при помощи $\deg v$. Вершина степени 0 называется изолированной, степени 1 — висячей.

Пример 50. *На рис. 28 изображён неориентированный граф $\mathfrak{G} = (V, E)$ с множеством вершин $V = \{a, b, c, d, e\}$ и множеством рёбер $E = \{(a, b), (a, c), (a, d), (b, d)\}$. В этом графе $\mathfrak{G}$ степень вершины a равна 3, вершин b и d — 2, вершины c — 1, а вершины e — 0. Таким образом, вершина e является изолированной, а вершина c — висячей.*

При разметке неориентированных графов каждое неориентированное ребро получает одну метку.

В дальнейшем, говоря о неориентированных графах, мы под словом «ребро» всегда будем подразумевать неориентированное ребро, не упоминая об этом специально.

Определение пути в неориентированном графе остаётся таким же, как в ориентированном, а на цикл накладывается дополнительное ограничение.

Определение 82 (Цикл в неориентированном графе)**.** *Циклом в неориентированном графе называется путь, который содержит хотя бы одно ребро, начальная вершина совпадает с конечной, и никакое ребро не проходится в противоположных направлениях подряд.*

Таким образом, путь вида (u, v, u) для вершин $u \neq v$ в неориентированном графе циклом не будет, так как одно и то же ребро (u, v) проходится сначала в одном направлении, а сразу же следом за этим — в другом. К циклу вида (u, u, u) это не относится, так как можно полагать, что петля (u, u) каждый раз проходится в одном и том же направлении.

Также это ограничение не относится к случаю, когда прохождения одного и того же ребра в противоположных направлениях расположены не подряд, а разделены другими вершинами. Например, если все вершины x, y, z, u, v, w попарно различны, то рассмотрим путь $(x, y, z, x, u, v, w, u, x)$. Здесь ребро (x, u) проходится два раза в противоположных направлениях, но не подряд, поэтому такой путь тоже является циклом, хотя, конечно, не простым.

Из последнего определения следует, что длина цикла в неориентированном графе не меньше трёх (если только этот цикл не образован одной петлёй). В графе на рис. 28 на предшествующей странице есть только один простой цикл: (a, b, d, a).

Замечание 9 (О циклах в неориентированных мультиграфах)**.** Последнее не относится к мультиграфам. Если есть два ребра e_1 и e_2 между одними и теми же вершинами u и v, то можно построить цикл длины два, в котором два ребра различны: (u, e_1, v, e_2, u).

Другим следствием этого ограничения является то, что никакая висячая вершина v в цикл входить не может. В самом деле, вернуться из v можно будет только в ту же вершину, из которой в v пришли, то есть придётся пройти по тому же самому ребру, но в противоположном направлении.

Определение 83 (Связный граф)**.** *Неориентированный граф называется с в я з н ы м, если каждая пара вершин в нём соединена путём.*

Сразу сделаем такое замечание.

Предложение 62. *Если в связном графе нет висячих вершин, то в нём есть цикл.*

Доказательство. Рассмотрим простой путь наибольшей длины, исходящий из любой вершины v_0: $(v_0, v_1, \ldots, v_n)$. Бесконечным он быть не может в силу конечности числа вершин и невозможности их повторения.

Поскольку длина является наибольшей, то этот простой путь нельзя продолжить так, чтобы он остался простым. Рассмотрим последнюю вершину v_n этого пути. Она не может быть висячей, следовательно, её степень не равна 1. Существует ребро (v_{n-1}, v_n), следовательно, её степень не равна 0. Получаем, что степень вершины v_n должна быть не меньше двух. Поэтому кроме ребра (v_{n-1}, v_n) должно быть ещё какое-то ребро вида (u, v_n). Если бы вершина u не совпадала ни с одной из уже пройденных v_i, $i = 0, \ldots, n-2$, то простой путь можно было бы удлинить: $(v_0, v_1, \ldots, v_n, u)$, что невозможно. Поэтому $u = v_{i_0}$ для некоторого v_{i_0}, $i_0 = 0, \ldots, n-2$, и в графе есть цикл $(v_{i_0}, v_{i_0+1}, \ldots, v_{n-1}, v_n, v_{i_0})$. □

Другое несложное наблюдение:

Предложение 63. *Если из связного графа выбросить любое ребро, входящее в простой цикл, то граф останется связным.*

Доказательство. См. задачу 159 на стр. 217. □

Так как в неориентированном графе $\mathfrak{G} = (V, E)$ отношение E симметрично, то матрица смежности неориентированного графа симметрична.

Для неориентированных графов отношение достижимости симметрично (если есть путь из вершины a в вершину b, то есть и обратный) и, следовательно, является отношением эквивалентности на множестве вершин V.

Определение 84 (Компоненты связности)**.** *В неориентированном графе классы эквивалентности по отношению достижимости называются* к о м п о н е н т а м и с в я з н о с т и.

В графе на рис. 28 на стр. 199 имеются две компоненты связности: $\{a, b, c, d\}$ и $\{e\}$.

§ 10.2. Плоские графы

Одну из важнейших групп графов образуют плоские графы. Приведённый в начале предыдущего параграфа пример с дорогами как раз является таким: его элементы размещаются на плоскости, а разные рёбра (дороги) не пересекаются (если есть пересечение, то это — перекрёсток и новая вершина). Другой задачей, приводящей к тому же понятию, является размещение радиодеталей на печатной плате: токопроводящие дорожки не должны пересекаться.

Определение 85 (Плоский граф)**.** *Граф называется* п л о с к и м *(или* п л а н а р н ы м*), если его можно изобразить на евклидовой плоскости так, чтобы рёбра попарно не пересекались.*

Часть плоскости, не содержащая рёбер, но ограниченная со всех сторон (извне или снаружи) рёбрами (если они есть), называется г р а н ь ю*.*

Иногда плоским графом называют непосредственно само такое изображение графа, а планарным — граф, который допускает такое изображение.

Пример 51. *На рис. 29 на следующей странице изображён плоский граф. У него три грани, обозначенные буквами A, B и C.*

Для связных плоских графов справедлива формула Эйлера.

Теорема 64 (Формула Эйлера для плоских графов)**.** *Если $\mathfrak{G}$ — связный плоский граф, то $e + 2 = v + f$, где e — количество рёбер, v — вершин, f — граней.*

Доказательство. Используем индукцию по количеству рёбер.

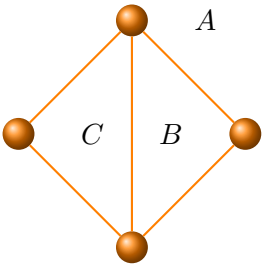


Рис. 29: Грани плоского графа.

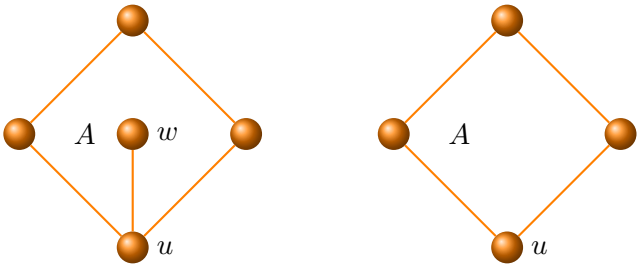


Рис. 30: Формула Эйлера, первый случай.

Если $e = 0$, то, чтобы граф был связным, нужно чтобы вершина была единственной, то есть $v = 1$. Грань в этом случае тоже одна: $f = 1$, и мы получаем равенство $0 + 2 = 1 + 1$.

Пусть теперь для любых связных плоских графов с меньше чем e рёбрами утверждение доказано. Возможны два случая.

Первый — в графе есть висячая вершина w с ребром (u, w), рис. 30. Удалим w и (u, w) из графа, получим граф $\mathfrak{G}'$ с $e - 1$ ребром и $v - 1$ вершиной. Заметим, что граф остался связным, так как любой простой путь в графе $\mathfrak{G}$ может проходить по ребру (u, w), только если он ведёт в вершину w, которую мы тоже удалили. Также очевидно, что граф остался плоским, а количество граней не изменилось: обе стороны удалённого ребра принадлежат одной и той же грани A, поэтому при удалении этого ребра две грани в одну не сольются. По

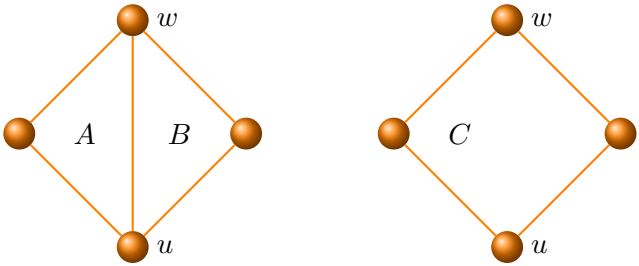


Рис. 31: Формула Эйлера, второй случай.

индукционному предположению $(e-1)+2=(v-1)+f$. Но тогда $e+2=v+f$, что и требуется.

Пусть теперь в графе висячих вершин нет. Тогда по предложению 62 на стр. 201 в графе есть цикл и, следовательно, простой цикл. Если из графа выбросить любое ребро (u,w) этого простого цикла, то полученный граф $\mathfrak{G}'$ останется связным (рис. 31). Заметим, что в этом случае две стороны ребра принадлежат разным граням A и B (одна из них — «внутри» цикла, а другая — вне его), то есть при его удалении две грани сольются в одну грань C и их количество уменьшится на единицу. Количество вершин в $\mathfrak{G}'$ останется тем же. По индукционному предположению $(e-1)+2=v+(f-1)$, поэтому $e+2=v+f$. □

Из этой теоремы можно вывести некоторые другие количественные соотношения для плоских графов.

Предложение 65. *В любом связном плоском графе без петель выполняется неравенство* $e+6\leqslant 3v$.

Доказательство. Прежде всего заметим, что ограничивающие грань рёбра образуют цикл. Как мы уже отметили, в неориентированном графе минимальная длина цикла равна трём. Следовательно, каждая грань ограничена тремя или более рёбрами. Каждое ребро ограничивает не более двух граней. Получаем $3f\leqslant 2e$. Умножим формулу Эйлера на 3: $3e+6=3f+3v$, применим полученное неравенство: $3e+6=3f+3v\leqslant 2e+3v$. Другими словами, $e+6\leqslant 3v$. □

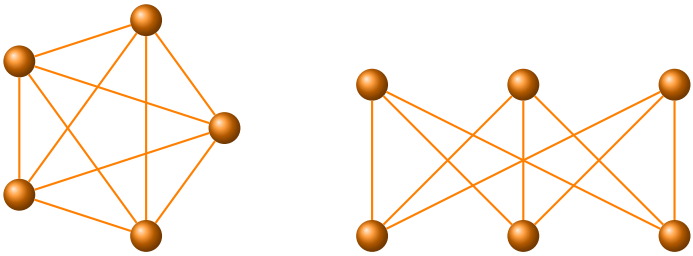


Рис. 32: Неплоские графы $\mathfrak{G}_5$ и $\mathfrak{G}_{3,3}$.

Следствие 66. *Каждый связный плоский граф без петель содержит вершину, степень которой не больше 5.*

Доказательство. В противном случае получим, что все вершины имеют степень 6 или больше. Сумма степеней всех вершин равна удвоенному количеству рёбер. Следовательно,

$$6v \leqslant \sum_{v \in V} \deg v = 2e,$$

$3v \leqslant e$, что невозможно для плоских графов, согласно предыдущему предложению. □

Следствие 67. *Граф $\mathfrak{G}_5$ не является плоским (рис. 32, слева).*

Доказательство. Для $\mathfrak{G}_5$ получаем $v = 5$ и $e = 10$, поэтому неравенство из предыдущего предложения не выполнено. □

Чуть сложнее доказывается невозможность изображения на плоскости ещё одного графа.

Предложение 68. *Граф $\mathfrak{G}_{3,3}$ неплоский (рис. 32, справа).*

Доказательство. Допустим, этот граф плоский. Для $\mathfrak{G}_{3,3}$ получаем $v = 6$ и $e = 9$. Заметим, что в этом графе все циклы имеют чётную длину (каждое ребро соединяет одну из верхних вершин с одной из нижних, пройдя нечётное число рёбер мы окажемся в другой половине). Поскольку длина цикла не меньше трёх, то получаем, что в графе $\mathfrak{G}_{3,3}$ минимальная длина цикла равна четырём. Следовательно, каждая грань окружена не менее чем четырьмя рёбрами.

Далее, рассуждая как в доказательстве предыдущего предложения, мы получим $4f \leqslant 2e$, $2f \leqslant e$ и $2e + 4 = 2v + 2f \leqslant 2v + e$, откуда $e + 4 \leqslant 2v$, что для графа не выполняется. □

Интересно, что приведённые два графа $\mathfrak{G}_5$ и $\mathfrak{G}_{3,3}$ являются в определённом смысле «единственными» причинами, по которым произвольный граф не является плоским. К. Куратовским доказана теорема о том, что в любом неплоском графе можно обнаружить часть, которая после исключений вершин (см. задачу 169 на стр. 217) превратится в один из этих графов.

Как следствие формулы Эйлера для плоских графов можно получить формулу Эйлера для выпуклых многогранников. Для этого достаточно показать, что вершины и рёбра многогранника образуют плоский граф.

Теорема 69. *Вершины и рёбра выпуклого многогранника G образуют плоский граф.*

Доказательство. Покажем, как перенести вершины и рёбра многогранника G на плоскость без пересечений в два этапа.

Сначала «изобразим» их без пересечений на сфере. Для этого поместим многогранник G внутрь сферы S так, чтобы центр сферы O находился внутри G. После этого построим проекции на сферу S всех вершин и рёбер G с помощью лучей, выходящих из центра O. Заметим, что каждый такой луч может пересечь многогранник G только один раз, так как он является выпуклым. Следовательно, каждая точка сферы может быть проекцией не более чем одной точки многогранника. Поскольку в реальном трёхмерном многограннике рёбра не пересекаются, то получаем, что и их проекции на сфере тоже не будут пересекаться. Следовательно, граф из вершин и рёбер G можно изобразить на сфере S без пересечений.

Теперь спроектируем на плоскость P изображение графа на сфере S. Для этого разместим сферу так, чтобы она касалась плоскости P, а противоположная точка N, «полюс» сферы, не была бы вершиной и не лежала ни на каком ребре. Спроектируем на плоскость P изображение графа на сфере S, с помощью лучей, исходящих из N. Такое проектирование снова будет взаимно-однозначным, если не брать в

расчёт саму точку N (но она изображению графа не принадлежит). Следовательно, раз изображение на сфере не имело пересечений рёбер, то и изображение на плоскости их иметь не будет. □

Следствие 70. *В выпуклом многограннике выполнено* $e+2=v+f$, *где* e *— количество рёбер,* v *— граней,* v *— вершин.*

§ 10.3. Эйлеровы графы

Одна из важнейших задач, которую приходится решать при работе с графами — это задача поиска тех или иных путей или циклов. Например, если граф представляет собой схему дорог и мы хотим проконтролировать их качество, то нам требуется пройти по каждой из них как минимум один раз. Наилучший вариант — пройти по каждой дороге в точности по одному разу, чтобы сэкономить время. Эта задача приводит нас к следующему понятию.

Определение 86 (Эйлеров путь, цикл)**.** *Путь (цикл) в графе называется эйлеровым, если он проходит по каждому ребру в точности один раз.*

Граф в котором есть эйлеров путь, называют полуэйлеровым, граф, в котором есть эйлеров цикл, — эйлеровым.

Эйлеров путь является математической формализацией популярной головоломки: изобразить ту или иную фигуру, не отрывая карандаша от бумаги (предполагается, что нельзя повторно обводить уже нарисованную линию). Например, для графа в виде «распечатанного конверта» (рис. 33 на следующей странице) эйлеров путь такой: $(e, d, b, a, c, e, a, d, c)$.

Эйлеров путь или цикл вовсе не обязательно единственен. Например, для того же графа можно указать другой эйлеров путь: $(e, d, a, c, e, a, b, d, c)$.

Существуют простые критерии, позволяющие определить, есть ли в графе эйлеров путь или цикл.

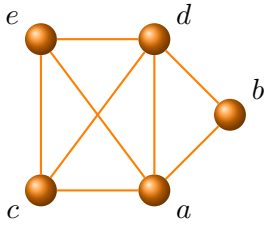


Рис. 33: Полуэйлеров граф.

Определение 87 (Чётные и нечётные вершины)**.** *Вершина графа называется чётной (соответственно нечётной), если её степень является чётной (соответственно нечётной).*

Теорема 71. *Связный граф является эйлеровым тогда и только тогда, когда все его вершины чётны.*

Доказательство. Если в графе $\mathfrak{G}$ есть эйлеров цикл, то, пройдя по нему, мы в каждую вершину войдём столько же раз сколько и выйдем. Поскольку входить и выходить мы обязаны по разным рёбрам (рёбра в эйлеровом цикле не повторяются), а с другой стороны все рёбра должны быть пройдены, получаем, что степень любой вершины является чётной.

В обратную сторону будем доказывать существование эйлерова цикла индукцией по количеству рёбер. Сейчас нам будет удобно считать циклом любой путь, начинающийся и оканчивающийся в одной вершине, даже если он не содержит рёбер.

Если рёбер нет, то из связности графа получаем, что единственная вершина v образует эйлеров цикл (v).

Пусть теперь граф содержит рёбра. Поскольку граф связен, то наименьшая степень вершины равна двум, висячих вершин нет. Согласно предложению 62 на стр. 201, это означает, что в графе есть простой цикл $(v_0, \ldots, v_n, v_0)$, где $n > 0$. Выкинем из $\mathfrak{G}$ все рёбра этого цикла, получим граф $\mathfrak{G}'$. Поскольку мы удалили по два ребра, ведущих к каждой вершине, и ранее все вершины были чётными, то в новом графе $\mathfrak{G}'$ все вершины снова будут чётными. Этот граф не

обязан быть связным, он может состоять из нескольких компонент связности: $\mathfrak{G}_1, \ldots, \mathfrak{G}_m$. Но тогда каждая компонента связности удовлетворяет условиям индукционного предположения: это связный граф, в котором все вершины чётные. Значит, в каждом из $\mathfrak{G}_i$ существует эйлеров цикл. Заметим, что каждая из компонент $\mathfrak{G}_i$ должна содержать хотя бы одну вершину цикла $(v_0, \ldots, v_n, v_0)$, обозначим её v_{j_i}. Без ограничения общности можно полагать, что компоненты $\mathfrak{G}_i$ пронумерованы так, чтобы v_{j_i} шли по возрастанию. Пусть эйлеров цикл в $\mathfrak{G}_i$ имеет вид $(v_{j_i}, v_{i1}, \ldots, v_{ik_i}, v_{j_i})$. Тогда эйлеров цикл в графе $\mathfrak{G}$ имеет следующий вид:

$$(v_0, \ldots, v_{j_1}, v_{11}, \ldots, v_{1k_1}, v_{j_1}, \ldots, v_{j_2}, v_{21}, \ldots, v_{2k_2}, v_{j_2}, \ldots \\ \ldots, v_{j_m}, v_{m1}, \ldots, v_{mk_m}, v_{j_m}, \ldots, v_0).$$

То есть мы выходим из вершины v_0, доходим по циклу до вершины v_{j_1} компоненты $\mathfrak{G}_1$, обходим $\mathfrak{G}_1$ и возвращаемся в v_{j_1}, далее доходим по циклу до вершины v_{j_2} компоненты $\mathfrak{G}_2$, обходим $\mathfrak{G}_2$ и т. д. □

Теорема 72. *Связный неэйлеров граф $\mathfrak{G}$ является полуэйлеровым тогда и только тогда, когда он содержит в точности две нечётных вершины. При этом эйлеров путь начинается в одной нечётной вершине и заканчивается в другой.*

Доказательство. Пусть граф $\mathfrak{G}$ неэйлеров, но полуэйлеров. Значит, в нём есть эйлеров путь $(v_0, \ldots, v_k)$, который не является циклом, то есть вершины v_0 и v_k различны. В каждую из промежуточных вершин $v_1, \ldots, v_{k-1}$ мы вошли столько же раз сколько и вышли из неё, следовательно, все они являются чётными. Из вершины v_0 мы вышли на один раз больше, чем вошли в неё, значит, она нечётная. В вершину v_k мы вошли на один раз больше, чем вышли из неё, значит, и она нечётная.

Пусть теперь связный граф $\mathfrak{G}$ содержит ровно две нечётные вершины. Так как граф связен, то между этими вершинами есть простой путь $v_0, \ldots, v_n$. Построим граф $\mathfrak{G}'$, выкинув из $\mathfrak{G}$ рёбра этого пути. Дальше рассуждаем как в доказательстве предыдущей теоремы. □

Теоремы остаются справедливыми и для мультиграфов.

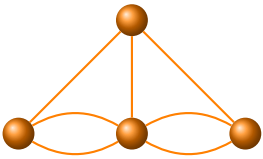


Рис. 34: Задача о кёнигсбергских мостах.

Пример 52. *Мультиграф, получаемый в «задаче о кёнигсбергских мостах», изображён на рис. 34.*

Этот мультиграф имеет четыре нечётные вершины, поэтому он не является полуэйлеровым и, тем более, эйлеровым.

Существует двойственное понятие.

Определение 88 (Гамильтонов цикл)**.** *Путь (цикл) в графе $\mathfrak{G}$ называют гамильтоновым, если он проходит по каждой из вершин в точности по одному разу (за исключением первой и последней вершины для цикла, они совпадают по определению).*

Граф в котором есть гамильтонов цикл, тоже называется гамильтоновым.

В отличие от эйлеровых путей и циклов, для проверки наличия которых мы привели простые критерии, аналогичных критериев для проверки гамильтоновых путей и циклов неизвестно, и предполагается, что их не существует. Иными словами, проверить, есть ли в графе гамильтонов путь или цикл, можно, только перебрав все возможные варианты и проверив каждый из них.

§ 10.4. Раскраска графов

Важным частным случаем разметки графов являются раскраски.

Определение 89 (Раскраска графа)**.** *Раскраской называют разметку вершин графа такую, что каждые две смежные вершины имеют различные метки.*

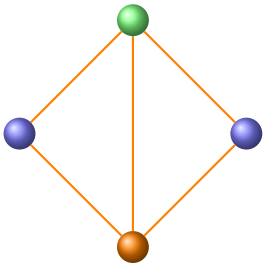


Рис. 35: Раскраска графа в три цвета.

В этом случае метки вершин часто называют цветами, откуда и пошло название. Тогда можно сказать, что при раскраске графа концы любого ребра должны иметь разные цвета (рис. 35).

В качестве тривиального следствия получаем, что граф с петлями нельзя раскрасить, так как у петли оба конца всегда получат один и тот же цвет.

Определение 90 (Хроматическое число)**.** *Наименьшее число цветов, в которое можно раскрасить граф, называется* *хроматическим числом* *графа.*

Задача о раскраске графа возникла в картографии. При создании карт требуется регионы, имеющие общую границу, раскрашивать в разные цвета. Эту задачу можно формализовать в терминах графов следующим образом. Будем считать регионы вершинами графа, а общую границу — ребром, которое эти вершины соединяет. Тогда карта превращается в граф (или мультиграф), причём, очевидно, в плоский, так как рёбра (границы регионов) пересекаться не могут.

Довольно давно картографами было замечено, что каждую карту можно раскрасить всего четырьмя цветами. В терминах графов этот факт формулируется так: любой плоский граф можно раскрасить в четыре цвета или, что то же самое, хроматическое число любого плоского графа не превосходит четырёх. Такое утверждение получило название «проблема четырёх красок». Впервые оно было явно сформулировано в середине XIX в. Довольно скоро после этого уда-

лось доказать, что каждый плоский граф раскрашивается в шесть, а затем — в пять цветов.

Мы докажем эти утверждения.

Теорема 73. *Каждый плоский граф без петель можно раскрасить в шесть цветов.*

Доказательство. Доказательство легко проводится индукцией по количеству вершин. Для одной вершины доказательство очевидно.

Пусть теперь вершин больше чем одна. Если граф $\mathfrak{G}$ несвязен, то разобьём его на компоненты связности $\mathfrak{G}_1$, ..., $\mathfrak{G}_m$, каждая из которых содержит меньше вершин, чем весь граф $\mathfrak{G}$. По индукционному предположению каждая из компонент раскрашивается в шесть цветов. Поскольку компоненты попарно не соединяются, то это и будет раскраской всего графа.

Пусть теперь граф $\mathfrak{G}$ связен. Согласно следствию 66 на стр. 205 в $\mathfrak{G}$ есть вершина, степень которой не превосходит 5. Пусть это — вершина v, а $u_1, \ldots, u_k$ — её соседи, $k \leqslant 5$. Построим граф $\mathfrak{G}'$, выбросив из $\mathfrak{G}$ вершину v и рёбра (v, u_1), ..., (v, u_k). Новый граф $\mathfrak{G}'$, естественно, будет плоским и содержит меньше вершин, чем $\mathfrak{G}$. По индукционному предположению $\mathfrak{G}'$ можно раскрасить в шесть цветов. Пусть $c_1, \ldots, c_k$ — цвета вершин $u_1, \ldots, u_k$ соответственно. Так как $k \leqslant 5$, а цветов шесть, то имеется цвет c_0, не совпадающий ни с одним из $c_1, \ldots, c_k$. Тогда вершину v в графе $\mathfrak{G}$ можно раскрасить в цвет c_0, оставив раскраску остальных вершин без изменений. □

Теперь модифицируем это доказательство, уменьшив число используемых цветов до пяти.

Теорема 74. *Каждый плоский граф без петель можно раскрасить в пять цветов.*

Доказательство. Изменения в доказательстве по сравнению с теоремой 73 коснутся только того места, где мы выбираем цвет для вершины v.

Первый вариант — если среди цветов $c_1, \ldots, c_k$ есть не больше четырёх различных. Тогда мы в качестве цвета v можно взять пятый, который среди них отсутствует.

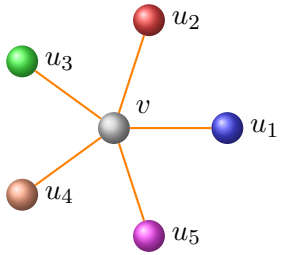


Рис. 36: Вершина степени 5.

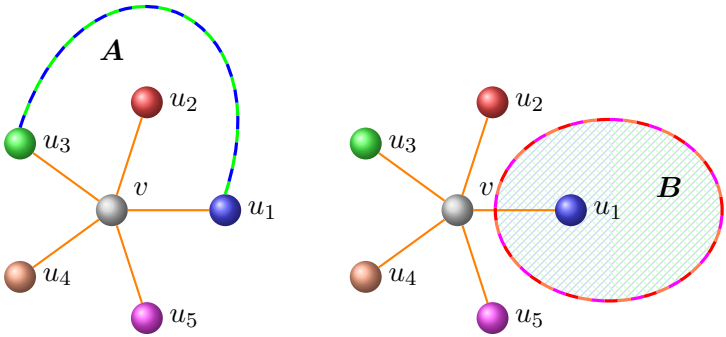


Рис. 37: Замена цветов.

Второй вариант — если все цвета $c_1, \ldots, c_5$ попарно различны. Тогда сразу выбрать цвет для v мы не можем, сначала придётся изменить раскраску графа $\mathfrak{G}'$. Будем считать, что на плоскости вершины $u_1, \ldots, u_5$ окружают вершину v именно в таком порядке против часовой стрелки (рис. 36).

Если из вершины u_1 в графе $\mathfrak{G}'$ есть путь в вершину u_3, проходящий только по вершинам цветов c_1 и c_3, то этот путь изолирует часть A графа $\mathfrak{G}'$, содержащую вершину u_2 (рис. 37, слева). Поменяем в части A цвета вершин c_2 и c_4 местами. Такое перекрашивание не приведёт к тому, что соседние вершины получат одинаковые цвета.

Но зато теперь вершина u_2 получит цвет c_4 и мы сведём всё к первому случаю.

Пусть теперь из вершины u_1 нет пути в вершину u_3, проходящего только по вершинам цветов c_1 и c_3. Тогда рассмотрим вершины графа, соединённые с вершиной u_1 такими путями (рис. 37 на предшествующей странице, справа). Обозначим эту часть графа B, она не содержит u_3, и окружена вершинами цветов c_2, c_4 и c_5. Перекрасим часть B графа, поменяв цвета c_1 и c_3 местами. Как и прежде после перекраски соседние вершины не получат одинаковых цветов. Вершина u_1 теперь имеет цвет c_3 и мы снова получили первый случай. □

Долгое время этот результат не удавалось улучшить, чтобы решить проблему четырёх красок. Наконец, К. Аппель и В. Хакен в 1977 г. смогли решить эту задачу. Основные принципы их доказательства аналогичны доказательству теоремы для пяти цветов, но количество рассматриваемых случаев намного больше и каждый раз рассматривается не одна вершина, а довольно большой подграф. Чтобы перебрать все эти случаи, ими была использована компьютерная программа. Таким образом, проблема четырёх красок стала первой значительной теоремой в математике, для доказательства которой была использована вычислительная техника. Впоследствии это доказательство было модифицировано и упрощено, однако количество вариантов остаётся всё ещё слишком большим, чтобы их можно было рассмотреть без использования компьютеров.

Существуют плоские графы, которые меньше чем в четыре цвета раскрасить нельзя (рис. 38 на следующей странице), поэтому ещё уменьшить число цветов невозможно.

Очевидно, что хроматическое число 1 имеют в точности графы без рёбер. Поэтому наименьшее хроматическое число нетривиальных графов — два. Эти графы довольно часто встречаются на практике и имеют особое название.

Определение 91 (Двудольный граф)**.** *Граф, который можно раскрасить в два цвета, называется* *двудольным* *(или* *бихроматическим)*.

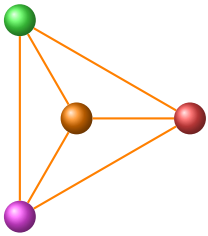


Рис. 38: Плоский граф с хроматическим числом 4.

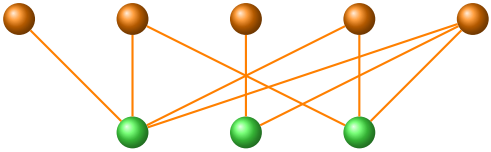


Рис. 39: Двудольный граф.

Название «двудольный» отражает следующую особенность структуры такого графа: все вершины делятся на две доли (соответствующие их раскраске в два цвета) и каждое ребро обязательно соединяет вершины разных долей (рис. 39).

С примером двудольного графа мы уже встречались — это граф $\mathfrak{G}_{3,3}$ из предложения 68 на стр. 205. При доказательстве утверждения о том, что он неплоский, мы использовали следующий факт: каждый цикл имеет чётную длину. Оказывается это условие является необходимым и достаточным для двудольности.

Теорема 75. *Граф является двудольным тогда и только тогда, когда в нём нет циклов нечётной длины.*

Доказательство. В прямую сторону доказательство очевидно, так как в любом пути вершины двух цветов обязательно должны будут чередоваться, поэтому цикл нечётной длины в два цвета раскрасить нельзя.

Пусть теперь в графе нет циклов нечётной длины. Прежде всего заметим, что если граф несвязен, то достаточно раскрасить в

два цвета каждую из компонент связности. Поэтому далее будем раскрашивать связный граф.

Пусть у нас есть два цвета c_0 и c_1. Зафиксируем произвольную вершину v. Тогда с каждой вершиной u можно связать некоторый путь из v в u: $(v_0 = v, v_1, \ldots, v_{n-1}, v_n = u)$. Раскрасим u в цвет c_0, если длина пути, то есть n, чётна, и в цвет c_1, если нечётна.

Прежде всего покажем, что такая раскраска однозначна, то есть не может быть двух путей из v в u, имеющих разную чётность. Допустим, что это не так и есть пути $(v_0 = v, v_1, \ldots, v_{n-1}, v_n = u)$ и $(w_0 = v, w_1, \ldots, w_{m-1}, w_m = u)$, причём n чётно, а m нечётно. Соединим их в такой путь:

$$(v_0 = v, v_1, \ldots, v_{n-1}, v_n = u = w_m, w_{m-1}, \ldots, w_1, w_0 = v).$$

Этот путь имеет длину $n+m$, то есть нечётную. Превратим этот путь в цикл. Для этого будем выполнять последовательное его сокращение: если встречается фрагмент (a, b, a), то оставляем только одно a. Таким образом, на каждом шаге длина уменьшается на два, поэтому остаётся нечётной. Когда таких фрагментов не останется, путь станет циклом нечётной длины, что невозможно.

Итак, мы показали, что сопоставление каждой вершине цвета задано корректно. Докажем, что это сопоставление даст раскраску графа. Снова будем рассуждать от противного. Допустим, что имеется ребро (u', u''), соединяющее вершины одного цвета. Это означает, что в графе существуют пути

$$(v_0 = v, v'_1, \ldots, v'_{n-1}, v'_n = u') \quad \text{и} \quad (v_0 = v, v''_1, \ldots, v''_{m-1}, v''_m = u''),$$

причём n и m имеют одинаковую чётность. Построим путь

$$(v_0 = v, v'_1, \ldots, v'_{n-1}, v'_n = u', u'' = v''_m, v''_{m-1}, \ldots, v''_1, v_0 = v),$$

который имеет длину $n + m + 1$, то есть нечётную. Далее такими же рассуждениями как и раньше превращаем этот путь в цикл нечётной длины и получаем противоречие. □

Задачи

157. Доказать, что сумма степеней всех вершин произвольного неориентированного графа чётна. ▼

158. Перечислить все неизоморфные неориентированные графы без петель, у которых не более четырёх вершин. ▼

159. Доказать, что неориентированный связный граф остаётся связным после удаления некоторого ребра (a, b) тогда и только тогда, когда ребро (a, b) принадлежит некоторому простому циклу. ▼

160. Доказать, что неориентированный связный граф с n вершинами

(а) содержит не менее $n - 1$ рёбер;

(б) если содержит больше $n - 1$ рёбер, то имеет хотя бы один цикл. ▼

161. Доказать, что во всякой группе V из шести человек есть трое попарно знакомых или трое попарно незнакомых. ▼

162. Доказать, что неориентированный граф $\mathfrak{G} = (V, E)$ связен тогда и только тогда, когда для каждого разбиения $V = V_1 \cup V_2$ с непустыми V_1 и V_2 существует ребро, соединяющее какую-то вершину из V_1 с какой-то вершиной из V_2. ▼

163. Доказать, что если в неориентированном графе имеется ровно две нечётные вершины, то они связаны путём. ▼

164. Доказать, что в связном неориентированном графе любые два простых пути максимальной длины имеют общую вершину. ▼

165. Пусть неориентированный граф без петель $\mathfrak{G} = (V, E)$ имеет k компонент связности. Доказать, что тогда ▼

$$|E| \leqslant \frac{(|V| - k)(|V| - k + 1)}{2}.$$

166. Доказать, что количество нечётных вершин графа всегда чётно. Определить, какое наименьшее количество рёбер нужно добавить к связному графу, чтобы он стал эйлеровым или полуэйлеровым. ▼

167. Для геодезических исследований территорию триангулируют: разбивают на треугольные части. В вершинах треугольников устанавливают специальные знаки — вышки. Определить, сколько всего вышек потребуется для триангуляции, если территория разделена на N треугольников, а на её границе располагается K вышек. ▼

168. Определить, сколько рёбер и вершин будет иметь многогранник, у которого f граней и все они треугольные. Для каких f такие многогранники могут существовать? ▼

169. И с к л ю ч е н и е м вершины называется такая операция с графом. Если есть вершина v степени 2 с инцидентными рёбрами (v, u) и (v, w), $u \neq w$, то мы удаляем вершину v и инцидентные рёбра, а вместо них добавляем ребро (u, w), рис. 40 на следующей странице. Доказать, что при исключении вершин планарность и

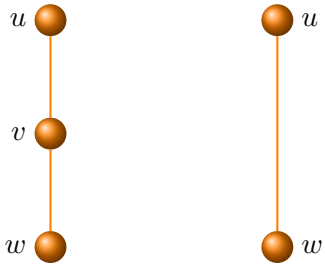


Рис. 40: Исключение вершины v.

эйлеровость графа не меняются, а хроматическое число и гамильтоновость могут измениться.

170. Для пяти правильных многогранников определить, каким будет граф, образованный их вершинами и рёбрами: эйлеровым, гамильтоновым, найти хроматическое число.

171. Показать, что наличие в графе эйлерова и гамильтонова циклов друг от друга не зависит.

172. Определить, является ли следующий граф $\mathfrak{G} = (V, E)$ эйлеровым (полуэйлеровым). Если $\mathfrak{G}$ не является эйлеровым, то удалить из $\mathfrak{G}$ минимальное число рёбер, чтобы он им стал. Построить в исходном или в получившемся графе эйлеров цикл. $V = \{a, b, c, e, f, g, h, k, m, n\}$, $E = \{(a, c), (a, h), (a, m), (a, k), (b, c), (b, k), (b, f), (b, m), (c, k), (c, m), (e, f), (e, g), (f, k), (f, n), (g, m), (g, h), (h, k), (h, m), (k, n)\}$.

173. Определить, является ли следующий граф $\mathfrak{G} = (V, E)$ двудольным. Если $\mathfrak{G}$ не двудольный, то удалить из $\mathfrak{G}$ наименьшее число рёбер, чтобы $\mathfrak{G}$ стал двудольным. $V = \{a, b, c, e, f, g, h, k, m, n\}$, $E = \{(a, h), (a, n), (a, k), (b, k), (b, f), (b, m), (c, k), (c, h), (e, f), (e, g), (f, a), (f, m), (g, m), (m, n)\}$.

Глава 11

Деревья

Краткое содержание: неориентированные и ориентированные деревья, эквивалентность разных определений деревьев, деревья и формулы, обходы деревьев.

Ключевые слова: неориентированное дерево, ориентированное дерево, корень, лист, ветвь, предок вершины, потомок вершины, высота дерева, глубина вершины, поддерево, лес, бинарное (двоичное) дерево, прямой (префиксный) обход дерева, обратный (суффиксный) обход дерева, инфиксный обход бинарного дерева.

§ 11.1. Неориентированные и ориентированные деревья

Деревья являются одним из интереснейших классов графов, используемых для представления разных иерахических структур.

Определение 92 (Неориентированное дерево)**.** *Неориентированный связный граф без циклов называется д е р е в о м.*

На рисунке 41 на следующей странице показан пример неориентированного дерева.

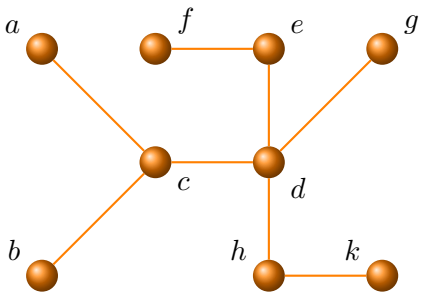


Рис. 41: Неориентированное дерево.

Неориентированные деревья имеют много эквивалентных определений.

Теорема 76. *Пусть $\mathfrak{G} = (V, E)$ — неориентированный граф. Тогда следующие условия эквивалентны:*

1) *$\mathfrak{G}$ является деревом;*
2) *для всяких двух вершин в $\mathfrak{G}$ имеется единственный соединяющий их простой путь;*
3) *$\mathfrak{G}$ связен, но при удалении из E любого ребра перестаёт быть связным;*
4) *$\mathfrak{G}$ связен и $|E| = |V| - 1$;*
5) *$\mathfrak{G}$ ациклический и $|E| = |V| - 1$;*
6) *$\mathfrak{G}$ ациклический, но добавление любого ребра к E порождает цикл.*

Доказательство. 1) $\Rightarrow$ 2) Если бы в $\mathfrak{G}$ некоторые две вершины соединялись двумя разными простыми путями, то, очевидно, в $\mathfrak{G}$ имелся бы цикл. Но это противоречит определению дерева.

2) $\Rightarrow$ 3) Если граф $\mathfrak{G}$ связен, но при удалении некоторого ребра $(u, v) \in E$ не теряет связности, то между u и v имеется путь, не содержащий это ребро. Но тогда в $\mathfrak{G}$ имеется не менее двух путей, соединяющих u и v, что противоречит условию 2).

3) $\Rightarrow$ 4) Предоставляется читателю (см. задачу 160 на стр. 217).

4) ⇒ 5) Если граф $\mathfrak{G}$ содержит цикл и является связным, то при удалении любого ребра из цикла связность не нарушится, а рёбер останется $|E| = |V| - 2$. Но в связном графе должно быть не меньше чем $|V| - 1$ рёбер (задача 160 на стр. 217, пункт (а)). Полученное противоречие показывает, что циклов в графе $\mathfrak{G}$ нет и выполнено условие 5).

5) ⇒ 6) Предположим, что добавление ребра (u, v) к E не привело к появлению цикла. Тогда в $\mathfrak{G}$ вершины u и v находятся в разных компонентах связности. Следовательно, количество компонент связности k больше одного. Если бы все эти компоненты (V_i, E_i), $i = 1, \ldots, k$, не имели циклов, то для них должно было бы выполняться $|E_i| = |V_i| - 1$ (задача 160 на стр. 217). Просуммировав по всем $i = 1, \ldots, k$, мы получим $|E| = |V| - k < |V| - 1$, что противоречит условию.

6) ⇒ 1) Если бы $\mathfrak{G}$ не был связным, то нашлись бы две вершины u и v из разных компонент связности. Тогда добавление ребра (u, v) к E не привело бы к появлению цикла, что противоречит 6). Следовательно, $\mathfrak{G}$ связен и является деревом. □

Определение 93 (Ориентированное дерево)**.** *Ориентированный граф $\mathfrak{G} = (V, E)$ называется (ориентированным) д е р е в о м, если*

1) *в нём есть ровно один исток $r \in E$, он называется к о р н е м дерева;*
2) *в каждую из остальных вершин входит ровно по одному ребру;*
3) *все вершины достижимы из корня.*

На рисунке 42 на следующей странице показано ориентированное дерево. Обратите внимание на то, что оно получено из неориентированного дерева на рис. 41 на противоположной странице с помощью выбора вершины c в качестве корня и ориентации всех рёбер в направлении «от корня».

Это не случайно. Имеет место следующее утверждение о связи между неориентированными и ориентированными деревьями.

Лемма 77. *Пусть в неориентированном дереве $\mathfrak{G} = (V, E)$ выбрана произвольная вершина $v \in V$. Для каждой вершины u обозначим*

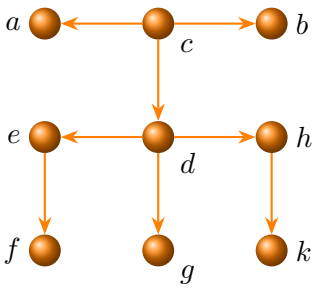


Рис. 42: Ориентированное дерево.

с помощью $n(u)$ минимальную длину пути из v в u. Для каждого неориентированного ребра (u, w) определим его направление от u к w, если $n(u) < n(w)$. Тогда полученный граф будет ориентированным деревом с корнем v.

Доказательство. См. задачу 175 на стр. 231. □

Для ориентированных деревьев часто удобно использовать следующее индуктивное определение.

Определение 94 (Ориентированное дерево, индуктивное определение)**.** *Ориентированный граф является деревом в следующих случаях.*

1) *Любой граф $\mathfrak{T}_0 = (V, E)$ с единственной вершиной $V = \{v\}$ и пустым множеством рёбер $E = \varnothing$ является деревом. Вершина v называется к о р н е м этого дерева.*

2) *Пусть графы $\mathfrak{T}_1 = (V_1, E_1)$, …, $\mathfrak{T}_k = (V_k, E_k)$ являются деревьями с корнями $r_1 \in V_1$, …, $r_k \in V_k$ соответственно, множества вершин V_i, $i = 1, \ldots, k$, попарно не пересекаются, r_0 — новая вершина: $r_0 \notin V_i$, $i = 1, \ldots, k$. Тогда следующий граф $\mathfrak{T} = (V, E)$ тоже будет деревом с корнем r_0:*

$$V = \{r_0\} \cup \bigcup_{i=1}^{k} V_i, \quad E = \{(r_0, r_i) : i = 1, \ldots, k\} \cup \bigcup_{i=1}^{k} E_i.$$

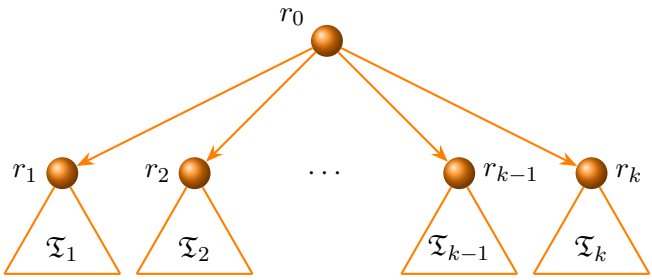


Рис. 43: Индуктивное построение ориентированного дерева.

Рисунок 43 иллюстрирует индукционный шаг этого определения.

Покажем, что оба определения ориентированных деревьев задают один и тот же класс графов.

Теорема 78. *Определения ориентированных деревьев 93 на стр. 221 и 94 на противоположной странице эквивалентны.*

Доказательство. Слева направо. Пусть граф $\mathfrak{G} = (V, E)$ удовлетворяет условиям определения 93. Покажем индукцией по числу вершин $|V|$, что $\mathfrak{G}$ удовлетворяет определению 94.

Базис индукции. Если $|V| = 1$, то единственная вершина $v \in V$ является по свойству 1) корнем дерева, то есть в этом графе рёбер нет: $E = \varnothing$. Тогда $\mathfrak{G} = \mathfrak{T}_0$.

Индукционный шаг. Предположим, что всякий граф с не более чем n вершинами, удовлетворяющий определению 93, будет удовлетворять и определению 94. Пусть граф $\mathfrak{G} = (V, E)$ с $(n+1)$-й вершиной удовлетворяет условиям определения 93. По условию 1) в нём имеется вершина-корень r_0. Пусть из r_0 выходит k рёбер и они ведут в вершины $r_1, \dots, r_k$, $k \geqslant 1$. Обозначим через $\mathfrak{G}_i = (V_i, E_i)$, $i = 1, \dots, k$, граф, состоящий из тех вершин, которые достижимы из r_i, а также — соединяющих их рёбер.

Легко понять, что $\mathfrak{G}_i$ удовлетворяет условиям определения 93. Действительно, в r_i не входят рёбра, то есть эта вершина — корень $\mathfrak{G}_i$. В каждую из остальных вершин из V_i входит по одному ребру как и в $\mathfrak{G}$. Если $v \in V_i$, то она достижима из r_i по определению графа

$\mathfrak{G}_i$. Так как $|V_i| \leqslant n$, то по индукционному предположению все $\mathfrak{G}_i$ удовлетворяют определению 94. Тогда граф $\mathfrak{G}$ получен по пункту 2) определения 94 из деревьев $\mathfrak{G}_1, \ldots, \mathfrak{G}_k$.

Справа налево. Если некоторый граф $\mathfrak{G} = (V, E)$ удовлетворяет условиям определения 94, то выполнение условий 1)–3) определения 93 для него легко установить индукцией по определению 94 (задача 179 на стр. 231). □

С ориентированными деревьями связана богатая терминология, пришедшая из двух источников: ботаники и области семейных отношений:

- К о р е н ь — единственная вершина, в которую не входят рёбра.
- Л и с т ь я — вершины, из которых не выходят рёбра.
- Путь из корня в лист называется в е т в ь ю дерева.
- В ы с о т а д е р е в а — это максимальная из длин его ветвей.
- Г л у б и н а в е р ш и н ы — это длина пути из корня в эту вершину.
- Для вершины $v \in V$ подграф дерева $\mathfrak{T} = (V, E)$, включающий все достижимые из v вершины и соединяющие их рёбра из E, образует п о д д е р е в о $\mathfrak{T}_v$ дерева $\mathfrak{T}$ с корнем v (см. задачу 180 на стр. 231).
- В ы с о т а в е р ш и н ы v — это высота дерева $\mathfrak{T}_v$.
- Граф, являющийся объединением нескольких непересекающихся деревьев, называется л е с о м.
- Если из вершины v ведёт ребро в вершину w, то v называется о т ц о м w, а w — с ы н о м v.
- Если из вершины v ведёт путь в вершину w, то v называется п р е д к о м w, а w — п о т о м к о м v.

Из определения дерева непосредственно следует, что у корня дерева отца нет, а у каждой из остальных вершин имеется единственный отец.

Выделим ещё один класс графов, обобщающий ориентированные деревья, — ориентированные ациклические графы. Два вида таких размеченных графов будут использованы далее для представления булевых функций. У этих графов может быть несколько корней — вершин, в которые не входят рёбра, и в каждую вершину может входить несколько рёбер, а не одно, как у деревьев.

§ 11.2. Деревья и формулы (выражения)

Напомним, что в параграфе 3.3 на стр. 66 было введено общее понятие формулы над системой функций $\mathcal{B}$ (определение 26 на стр. 66). Его можно применить для произвольных функций, а не только булевых. В языках программирования такие конструкции часто называются выражениями.

Определение 95 (Формула)**.** *Формула над множеством функций $\boldsymbol{F}$, множеством констант $\boldsymbol{C}$ и множеством переменных $\boldsymbol{V}$ определяется индуктивно по следующим правилам.*

1) *Каждая переменная из $\boldsymbol{V}$ есть формула.*
2) *Каждая константа из $\boldsymbol{C}$ есть формула.*
3) *Если $g_1, \ldots, g_k$ — формулы, а $f^{(k)}$ — k-местная функция из $\boldsymbol{F}$, то $f(g_1, \ldots, g_k)$ — это формула.*

Обозначим множество всех формул через $\mathcal{F}(\boldsymbol{F}, \boldsymbol{C}, \boldsymbol{V})$.

Для $\boldsymbol{F}$, $\boldsymbol{C}$ и $\boldsymbol{V}$ также определим специальный класс упорядоченных размеченных деревьев.

Определение 96 (Формульное дерево)**.** *Упорядоченное ориентированное дерево назовём формульным, если*

1) *его листья помечены элементами из $\boldsymbol{C} \cup \boldsymbol{V}$;*
2) *внутренние вершины помечены функциями из $\boldsymbol{F}$, причём, если вершина помечена символом k-местной функции из $\boldsymbol{V}$, то у неё имеется в точности k сыновей.*

Множество формульных деревьев обозначим с помощью $\mathcal{T}(\boldsymbol{F}, \boldsymbol{C}, \boldsymbol{V})$.

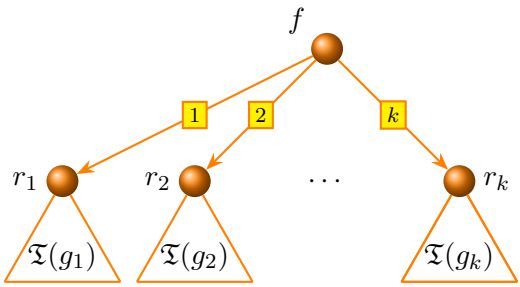


Рис. 44: Ориентированное дерево для формулы.

Предложение 79. *Между множеством формул $\mathcal{F}(\boldsymbol{F}, \boldsymbol{C}, \boldsymbol{V})$ и множеством деревьев $\mathcal{T}(\boldsymbol{F}, \boldsymbol{C}, \boldsymbol{V})$ имеется взаимно однозначное соответствие.*

Доказательство. Это соответствие легко устанавливается индукцией по определениям формул и деревьев.

Константе c (или переменной x) соответствует дерево $\mathfrak{T}(c)$ (или $\mathfrak{T}(x)$), которое состоит из одного корня, помеченного c (или соответственно x).

Формуле вида $f(g_1, \ldots, g_k)$ ставится в соответствие дерево, изображённое на рис. 44. По индукции мы считаем, что все деревья $\mathfrak{T}(g_i)$, $i = 1, \ldots, k$, уже построены и имеют корни $r_1, \ldots, r_k$ соответственно. □

Пример 53. *Рассмотрим, например, класс обычных арифметических формул над множеством функций $\boldsymbol{F} = \{+, -, \times, /\}$, целочисленных констант $\boldsymbol{C} = \{0, 1, 2, \ldots\}$ и переменных $\boldsymbol{V} = \{x, y, z, \ldots\}$. Рассмотрим формулу*

$$\Phi = +(\times(5, +(x, 7)), (/(y, +(x, 7))),$$

её обычное, инфиксное, представление имеет вид

$$\Phi = 5 \times (x + 7) + y/(x + 7).$$

Тогда в соответствии с предложением 79 эта формула представляется деревом $\mathfrak{T}_\Phi$, изображённом на рис. 45 на следующей странице.

Заметим, что у деревьев, представляющих арифметические или логические (булевы) формулы, внутренние вершины имеют не более

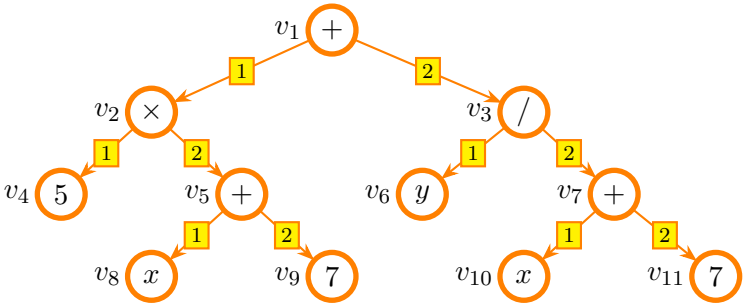


Рис. 45: Формульное дерево из примера 53.

двух сыновей. Такие деревья образуют важный подкласс ориентированных деревьев, называемых бинарными или двоичными.

Определение 97 (Бинарное дерево)**.** *Ориентированное дерево называется б и н а р н ы м (или д в о и ч н ы м), если у каждой его внутренней вершины есть не более двух сыновей, а рёбра, ведущие к ним, помечены двумя разными метками (обычно используются метки из пар: «левый»–«правый», «0»–«1», «+»–«−» и т. д.).*

Бинарное дерево называется п о л н ы м, если каждая внутренняя вершина имеет два сына и все ветви имеют одинаковую длину.

Ориентированные ациклические графы также используются для представления формул. Они получаются из соответствующих деревьев при «склеивании» вершин, представляющих одинаковые подформулы.

Пример 54. *В примере 53 на противоположной странице дерево содержит два одинаковых поддерева $\mathfrak{T}_{v_5}$ и $\mathfrak{T}_{v_7}$, представляющих подформулу $x + 7$. Их можно «склеить», как показано на рис. 46 на следующей странице.*

Часто номера рёбер не указывают, если они нумеруются у каждой вершины подряд слева направо. Поэтому в графе на рис. 45 номера рёбер можно опустить, а на рис. 46 на следующей странице — нельзя.

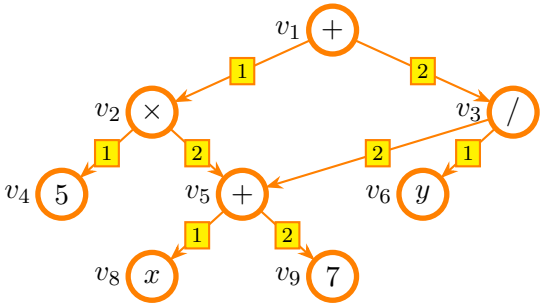


Рис. 46: Ациклический граф из примера 54.

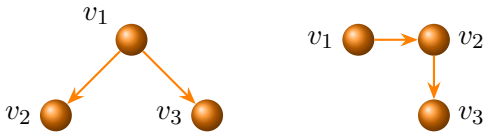


Рис. 47: Два дерева с одинаковым префиксным обходом.

§ 11.3. Обходы деревьев

Часто при обработке представленной в ориентированном упорядоченном дереве информации требуется обойти некоторым регулярным способом все его вершины. Имеется два естественных стандартных способа обхода деревьев. Каждый из них позволяет линейно упорядочить вершины дерева и тем самым представить его «двумерную структуру» в виде линейной последовательности вершин.

Префиксный (или прямой) обход дерева основан на принципе: «сначала родитель, затем дети». Сразу отметим, что если просто выписать вершины дерева в указанной последовательности, то однозначно восстановить исходное дерево невозможно. Например, последовательность вершин (v_1, v_2, v_3) соответствует двум разным деревьям, показанным на рис. 47. Один из способов преодолеть эту трудность, который мы и применим, — это указывать у каждой вершины количество её сыновей.

Определение 98 (Префиксный обход)**.** *Определим индукцией по построению дерева* $\mathfrak{T}$ *(в определение 94 на стр. 222) его прямое представление* ПРФ($\mathfrak{T}$)*:*

1) *если* $\mathfrak{T}_0 = (\{v\}, \varnothing)$*, то* ПРФ($\mathfrak{T}_0$) $= (v^{(0)})$.
2) *если* $\mathfrak{T}$ *получено из деревьев* $\mathfrak{T}_1, \ldots, \mathfrak{T}_k$ *и нового корня* r_0 *(пункт 2) определения 94), то*

$$\text{ПРФ}(\mathfrak{T}) = (r_0^{(k)}, \text{ПРФ}(\mathfrak{T}_1), \ldots, \text{ПРФ}(\mathfrak{T}_k)).$$

Суффиксный (или обратный) обход дерева основан на противоположном принципе: «сначала дети, затем родитель».

Определение 99 (Суффиксный обход)**.** *Определяем* СФФ($\mathfrak{T}$) *индукцией по построению дерева* $\mathfrak{T}$*:*

1) *если* $\mathfrak{T}_0 = (\{v\}, \varnothing)$*, то* СФФ($\mathfrak{T}_0$) $= (v^{(0)})$.
2) *если* $\mathfrak{T}$ *получено из деревьев* $\mathfrak{T}_1, \ldots, \mathfrak{T}_k$ *и нового корня* r_0*, то*

$$\text{СФФ}(\mathfrak{T}) = (\text{СФФ}(\mathfrak{T}_1), \ldots, \text{СФФ}(\mathfrak{T}_k), r_0^{(k)}).$$

По каждому из указанных обходов исходное дерево может быть однозначно восстановлено (см. задачу 185 на стр. 232).

В случае формульных деревьев, когда местность каждой функции заранее известна, известно и количество сыновей, которое должно быть у вершины с меткой f. Поэтому при обходах формульных деревьев указывать количество сыновей не обязательно. Следовательно, формульное дерево $\mathfrak{T}$ из класса $\mathcal{T}(\boldsymbol{F}, \boldsymbol{C}, \boldsymbol{V})$ однозначно восстанавливается по любому из обходов ПРФ($\mathfrak{T}$) или СФФ($\mathfrak{T}$).

Замечание 10. Для программирования особенно интересен обратный обход, называемый для формульных деревьев «обратной польской записью». По нему компилятор легко строит программу вычисления соответствующего выражения.

Для бинарных деревьев, внутренние вершины которых имеют не более двух сыновей, каждый из которых помечен как «левый» и «правый» (даже если он один!), можно естественно определить ещё один способ обхода — инфиксный (или внутренний) обход,

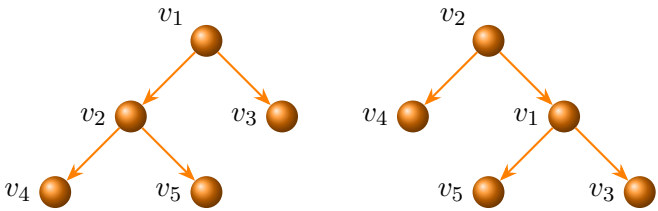


Рис. 48: Два дерева с одинаковым инфиксным обходом.

основанный на принципе: «сначала левый сын, затем родитель, а затем правый сын».

Определение 100 (Инфиксный обход)**.** *Определяем* ИНФ($\mathfrak{T}$) *индукцией по построению дерева* $\mathfrak{T}$*:*

1) *если* $\mathfrak{T}_0 = (\{v\}, \varnothing)$*, то* СФФ($\mathfrak{T}_0$) $= (v^{(0)})$.
2) *если* $\mathfrak{T}$ *получено из одного «левого» дерева* $\mathfrak{T}_1$ *и нового корня* r_0*, то* ИНФ($\mathfrak{T}$) $=$ (ИНФ($\mathfrak{T}_1$), $r_0^{(1)}$)*;*
3) *если* $\mathfrak{T}$ *получено из одного «правого» дерева* $\mathfrak{T}_2$ *и нового корня* r_0*, то* ИНФ($\mathfrak{T}$) $=$ ($r_0^{(1)}$, ИНФ($\mathfrak{T}_2$))*;*
4) *если* $\mathfrak{T}$ *получено из двух деревьев: «левого»* $\mathfrak{T}_1$*, «правого»* $\mathfrak{T}_2$*, а также нового корня* r_0*, то*

$$\text{ИНФ}(\mathfrak{T}) = (\text{ИНФ}(\mathfrak{T}_1), r_0^{(2)}, \text{ИНФ}(\mathfrak{T}_2)).$$

В отличие от префиксного и суффиксного обходов по инфиксному однозначно восстановить дерево нельзя. Например, оба дерева, изображённые на рис. 48, дают один и тот же порядок вершин при инфиксном обходе: $(v_4^{(0)}, v_2^{(2)}, v_5^{(0)}, v_1^{(2)}, v_3^{(0)})$.

Пример 55. *Построим в соответствии с этими определениями три разных обхода бинарного дерева* $\mathfrak{T}_\Phi$*, изображённого на рис. 45 на стр. 227.*

$$\text{ПРФ}(\mathfrak{T}_\Phi) = v_1(+), v_2(\times), v_4(5), v_5(+), v_8(x), v_9(7), v_3(:), v_6(y), v_7(+), v_{10}(x), v_{11}(7);$$

$$\text{СФФ}(\mathfrak{T}_\Phi) = v_4(5), v_8(x), v_9(7), v_5(+), v_2(\times), v_6(y), v_{10}(x), v_{11}(7), v_7(+), v_3(:), v_1(+);$$

$$\text{ИНФ}(\mathfrak{T}_\Phi) = v_4(5), v_2(\times), v_8(x), v_5(+), v_9(7), v_1(+), v_6(y), v_3(:), v_{10}(x), v_7(+), v_{11}(7).$$

В скобках после вершины указана её метка.

Задачи

174. Будет ли неориентированное дерево плоским, эйлеровым, полуэйлеровым, гамильтоновым графом? Найти хроматическое число. ▼

175. Доказать лемму 77 на стр. 221. ▼

176. Доказать, что в неориентированном дереве $\mathfrak{T} = (V, E)$, содержащем хотя бы одно ребро, количество висячих вершин равно $2 + \sum\limits_{v \in V_2} (\deg v - 2)$, где V_2 — множество невисячих вершин. ▼

177. Доказать, что если в связном неориентированном графе число вершин равно числу рёбер, то можно выбросить одно из рёбер так, что после этого граф станет деревом. ▼

178. Доказать, что в неориентированном дереве существует вершина, через которую проходят все максимальные простые пути. ▼

179. Доказать теорему 78 на стр. 223 в обратную сторону. ▼

180. Пусть $\mathfrak{T} = (V, E)$ — это ориентированное дерево с корнем $v_0 \in V$. Определим для каждой вершины $v \in V$ подграф $\mathfrak{T}_v = (V_v, E_v)$ следующим образом: V_v — это множество вершин, достижимых из v в $\mathfrak{T}$, а E_v — это множество рёбер из E, оба конца которых входят в V_v. Доказать, что

(а) $\mathfrak{T}_v$ является деревом с корнем v;

(б) если две разные вершины v и u имеют одинаковую глубину, то деревья $\mathfrak{T}_v$ и $\mathfrak{T}_u$ не пересекаются. ▼

181. Пусть $\leqslant$ — отношение частичного порядка на конечном множестве V, которое обладает следующими двумя свойствами:

(а) существует наименьший элемент r;

(б) если x и y несравнимы, $a \geqslant x$, $b \geqslant y$, то a и b тоже несравнимы.

Отношение $E(x, y)$ означает, что x — максимальный элемент, меньший y. Доказать, что (V, E) — ориентированное дерево с корнем r. ▼

182. Пусть $\mathfrak{T} = (V, E)$ — ориентированное дерево, а $x \leqslant y$ означает, что вершина y достижима из вершины x. Доказать, что $\leqslant$ — отношение нестрогого частичного порядка на V, удовлетворяющее свойствам (а) и (б) из предыдущей задачи. ▼

183. Пусть $\mathfrak{G} = (V, E)$ — ориентированный граф с не менее чем двумя вершинами. Доказать, что $\mathfrak{G}$ является (ориентированным) деревом тогда и только тогда, когда в $\mathfrak{G}$ нет циклов, имеется один исток r, а в каждую из остальных вершин $v \in V \setminus \{r\}$ входит ровно одно ребро. ▼

184. Пусть корень ориентированного дерева $\mathfrak{T}$ имеет 5 сыновей, а каждая из остальных внутренних вершин имеет 3 или 4 сына, при этом число вершин с тремя сыновьями вдвое больше числа вершин с четырьмя. Сколько всего вершин и рёбер в $\mathfrak{T}$, если известно, что число его листьев равно 26? ▼

185. Пусть $\mathfrak{F} = (\mathfrak{T}_1, \ldots, \mathfrak{T}_n)$ — лес деревьев. Доказать, что по последовательности $P = (\text{ПРФ}(\mathfrak{T}_1), \ldots, \text{ПРФ}(\mathfrak{T}_k))$ можно однозначно восстановить лес $\mathfrak{F}$. Доказать аналогичное утверждение для суффиксных обходов. ▼

186. Доказать по индукции, что в каждом бинарном дереве число n вершин с двумя сыновьями на единицу меньше числа ℓ листьев. ▼

187. Найти число листьев и вершин в полном бинарном дереве высоты h. ▼

188. Построить дерево, представляющее следующую логическую формулу

$$\Psi = ((x \vee \neg y) \wedge \neg(z \to (x \wedge y))) \vee (\neg z \oplus y).$$

Для полученного дерева определить прямой, обратный и инфиксный обходы (для отрицания единственного сына считать правым). ▼

189. Построить дерево и ациклический ориентированный граф, представляющие следующую арифметическую формулу

$$\Phi = (a + b)/(c + a \times d) + ((c + a \times d) - (a + b) \times (c - d)).$$

Сколько вершин удалось сократить? ▼

Глава 12

Три алгоритма на графах

Краткое содержание: построение минимального остовного дерева: алгоритм Крускала, задача о лабиринте и поиск в глубину на неориентированном графе, нахождение кратчайших путей из одного источника: алгоритм Дейкстры.

Ключевые слова: остовное дерево графа, минимальное остовное дерево, глубинное остовное дерево, прямое ребро, обратное ребро, мост, длина пути, кратчайший путь, дерево кратчайших путей из вершины графа.

§ 12.1. Минимальное остовное дерево

Для неориентированных связных графов одним из интересных классов подграфов являются деревья, сохраняющие связность вершин. Они называются остовными деревьями (а также — остовами, каркасами или скелетами) графа.

Определение 101 (Остовное дерево)**.** *Остовным деревом неориентированного связного графа* $\mathfrak{G} = (V, E)$ *называется его подграф* $\mathfrak{T} = (V, T)$*, являющийся деревом.*

Пусть задана функция $c : E \to \mathbb{R}$, приписывающая каждому ребру $e \in E$ некоторое действительное число $c(e) \in \mathbb{R}$ — его вес (стоимость, длину). Тогда вес $c(\mathfrak{T})$ дерева $\mathfrak{T}$ определяется как сумма весов всех его рёбер, то есть $c(\mathfrak{T}) = \sum\limits_{e \in T} c(e)$.

Определение 102 (Минимальное остовное дерево)**.** *Остовное дерево наименьшего веса называется минимальным остовным деревом.*

Таким образом, минимальное остовное дерево — это самая лёгкая (дешёвая, короткая) система путей, которые связывают все вершины графа $\mathfrak{G}$.

Опишем алгоритм построения минимального остовного дерева, предложенный Дж. Крускалом в 1956 г. (рис. 49 на следующей странице). Входными данными для алгоритма Крускала являются неориентрированный граф $\mathfrak{G} = (V, E)$ и функция разметки рёбер $c : E \to \mathbb{R}$ вещественными числами, выходными — множество рёбер T, образующих минимальное остовное дерево.

Докажем, что этот алгоритм корректен.

Теорема 80 (Корректность алгоритма Крускала)**.** *Результатом работы алгоритма* МинОД$(\mathfrak{G}, c)$ *является минимальное остовное дерево входного графа* $\mathfrak{G} = (V, E)$ *с функцией разметки* c*.*

Доказательство. С помощью T_i обозначим значение переменной T после выполнения i шагов цикла в строках 8–12.

Предположим, результатом работы МинОД$(\mathfrak{G}, c)$ является граф $\mathfrak{T} = (V, T)$. Отметим вначале, что этот граф $\mathfrak{T}$ является деревом. Действительно, отсутствие циклов непосредственно следует из алгоритма и определения множеств T_i. Предположим, что граф $\mathfrak{T}$ не является связным. Тогда должны существовать две вершины $u, v \in V$, которые не достижимы друг из друга в $\mathfrak{T}$. Но исходный граф $\mathfrak{G}$ связен, поэтому в нём есть путь из u в v. Тогда на этом пути обязательно имеется такое ребро $e_i = (a, b) \in (E \setminus T)$, у которого один конец a соединён путём с вершиной u в графе $\mathfrak{T}$, а второй конец b — не соединён. Получаем, что на шаге i ребро e_i должно попасть в T_{i+1}, так

Алгоритм МинОД($\mathfrak{G}$, c) # Минимальное остовное дерево
  Вход: $\mathfrak{G} = (V, E)$ — связный граф
  Вход: $c : E \to \mathbb{R}$ — веса рёбер
  Выход: $\mathfrak{T} = (V, T)$ — минимальное остовное дерево
  $m \leftarrow |E|$
  $T \leftarrow \varnothing$
  Упорядочить рёбра E по возрастанию веса
    # $c(e_1) \leqslant c(e_2) \leqslant \cdots \leqslant c(e_m)$
  Для всех $i \leftarrow 1, \ldots, m$ **выполнять**
    Если в графе $(V, T \cup \{e_i\})$ нет циклов **то**
      $T \leftarrow T \cup \{e_i\}$
    Конец Если
  Конец Для
Конец Алгоритм

Рис. 49: Алгоритм Крускала

как его добавление не образует цикла. Это противоречит $e_i \in (E \setminus T)$, следовательно, граф $\mathfrak{T}$ связен.

Докажем, что дерево $\mathfrak{T}$ имеет минимальный вес среди остовных. Пусть $T = \{d_1, \ldots, d_k, \ldots, d_{n-1}\}$ — порядок всех рёбер T, в котором они в T попали при выполнении алгоритма. В частности, они упорядочены по возрастанию веса. Покажем индукцией по k, что существует минимальное остовное дерево $\mathfrak{T}_k = (V, T^{(k)})$, содержащее рёбра $d_1, \ldots, d_k$.

Базис индукции, $k = 0$, очевиден.

Индукционный шаг. Предположим, что для k утверждение доказано. Допустим, $d_{k+1} \notin T^{(k)}$. Так как рёбра $T^{(k)}$ образуют дерево и $d_{k+1} \notin T^{(k)}$, то какие-то рёбра множества $T^{(k)} \cup \{d_{k+1}\}$ образуют простой цикл C, включающий ребро d_{k+1} (так как без него циклов не было).

Возможны два случая. Первый — все остальные рёбра в C имеют вес строго меньший веса d_{k+1}. Но тогда все рёбра из C должны были быть рассмотрены алгоритмом Крускала раньше d_{k+1}. Поскольку

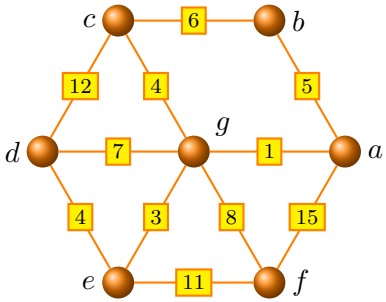


Рис. 50: Граф из примера 56.

они вместе с $d_1, \ldots, d_k$ цикла не образовывали, то T_k должно их все включать. Следовательно, ребро d_{k+1} не могло быть добавлено в T, противоречие.

Второй случай — в C есть ребро d', вес которого не меньше чем у ребра d_{k+1}. Рассмотрим граф $\mathfrak{T}' = (V, T')$, где

$$T' = T^{(k)} \cup \{d_{k+1}\} \setminus \{d'\}.$$

Так как d' является элементом простого цикла в связном графе $(V, T^{(k)} \cup \{d_{k+1}\})$, то при его удалении граф останется связным. Поскольку

$$|T'| = |T^{(k)} \cup \{d_{k+1}\} \setminus \{d'\}| = |T^{(k)}| = |V| - 1,$$

то $\mathfrak{T}'$ является деревом, вес которого не больше веса $\mathfrak{T}_k$. Так как вес $\mathfrak{T}_k$ наименьший, то $\mathfrak{T}'$ — минимальное остовное дерево, содержащее рёбра $d_1, \ldots, d_k, d_{k+1}$, что и требуется. □

Отметим, что в графе может быть несколько рёбер одинакового веса, их порядок может оказаться произвольным. В этом случае могут быть получены различные минимальные остовные деревья.

Пример 56. *Возьмём граф $\mathfrak{G}$, изображённый на рис. 50. Рассмотрим результат выполнения* МинОД$(\mathfrak{G}, c)$. *На первом этапе нам нужно упорядочить все рёбра, а на втором — для каждого из них определить соответствующее множество T_i. Рёбра, попавшие в T, будем по ходу вычисления*

e_i	$c(e_i)$	Включать?	T_i
(a,g)	1	+	$\{(a,g)\}$
(g,e)	3	+	$\{(a,g),(g,e)\}$
(g,c)	4	+	$\{(a,g),(g,e),(g,c)\}$
(e,d)	4	+	$\{(a,g),(g,e),(g,c),(e,d)\}$
(a,b)	5	+	$\{(a,g),(g,e),(g,c),(e,d),(a,b)\}$
(b,c)	6	−	не меняется
(d,g)	7	−	не меняется
(f,g)	8	+	$\{(a,g),(g,e),(g,c),(e,d),(a,b),(f,g)\}$
(e,f)	11	−	не меняется
(c,d)	12	−	не меняется
(a,f)	15	−	не меняется

Рис. 51: Выполнение алгоритма Крускала из примера 56.

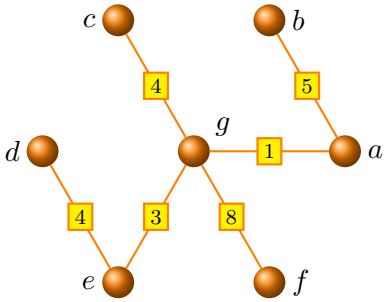


Рис. 52: Результат алгоритма Крускала, пример 56.

отмечать знаком «+», а не попавшие — знаком «−» (рис. 51). Таким образом, мы построили для $\mathfrak{G}$ *минимальное остовное дерево* $\mathfrak{T} = (V, T)$, *где*

$$T = T_8 = \{(a,g),(g,e),(g,c),(e,d),(a,b),(f,g)\}.$$

Оно показано на рис. 52, а его вес равен $c(\mathfrak{T}) = 25$.

Замечание 11. Так как дерево с n вершинами содержит в точности $n-1$ рёбер, то работу МинОД$(\mathfrak{G}, c)$ можно прекращать после такого шага i, на

котором в T_i окажется $|V| - 1$ рёбер. В нашем примере $|V| = 7$ и алгоритм мог остановиться после восьмого шага.

§ 12.2. Поиск в глубину и задача о лабиринте

Задача поиска выхода из лабиринта известна с древних времён. В терминах графов её можно формализовать следующим образом. Лабиринт — это неориентированный граф, вершины которого представляют «перекрёстки» лабиринта, а рёбра — дорожки между соседними перекрёстками. Одна или несколько вершин отмечены как выходы. Задача состоит в построении пути из некоторой исходной вершины в вершину-выход.

В этом разделе мы рассмотрим метод обхода всех вершин графа, называемый поиском в глубину (рис. 53 на следующей странице). Его идею кратко можно описать так: находясь в некоторой вершине v, идём из неё в произвольную ещё не посещённую смежную вершину w, если такой вершины нет, то возвращаемся в вершину, из которой мы пришли в v.

Теорема 81 (Корректность поиска в глубину)**.** *Алгоритм* ПОГ $(\mathfrak{G})$ *обходит (нумерует) все вершины графа* $\mathfrak{G} = (V, E)$*. Если* $\mathfrak{G}$ *— связный граф, то результатом является* $\mathfrak{T} = (V, T)$ *— остовное дерево для* $\mathfrak{G}$*. Если граф* $\mathfrak{G}$ *не является связным, то* $\mathfrak{T} = (V, T)$ *— это остовной лес для* $\mathfrak{G}$*, то есть объединение остовных деревьев для каждой из компонент связности* $\mathfrak{G}$*.*

Доказательство. Первое утверждение следует из того, что по окончании алгоритма ПОГ $(\mathfrak{G})$ выполнено $V = R$, то есть для каждой вершины $v \in V$ вызывалась процедура Поиск (v), которая в строке 18 присвоила номер v.

Заметим теперь, что если ребро (v, w) попадает в T, то вызов процедуры Поиск (w) происходит после вызова Поиск (v) и поэтому $Num[v] < Num[w]$. Существование цикла в $\mathfrak{T}$ означало бы, что для некоторого ребра из T это свойство нарушено (почему?). Следовательно, в $\mathfrak{T}$ циклов нет.

```
1: Алгоритм ПОГ(𝔊)                                  # Поиск в глубину
2:     Вход: 𝔊 = (V, E) — граф
3:     Вход: L_v — списки смежности вершин 𝔊, v ∈ V
4:     Выход: Num — порядок прохождения вершин
5:     Выход: T — рёбра остовных деревьев
6:     Создать массив Num[V]
7:     T ← ∅
8:     R ← ∅
9:     n ← 1
10:    Пока R ≠ V выполнять
11:        Выбрать v ∈ V \ R
12:        Поиск (v)
13:    Конец Пока
14: Конец Алгоритм

15: Алгоритм Поиск(v)                               # Процедура поиска
16:    Вход: v ∈ V \ R — вершина графа
17:    R ← R ∪ {v}
18:    Num[v] ← n
19:    n ← n + 1
20:    Для всех w ∈ L_v выполнять
21:        Если w ∉ R то
22:            T ← T ∪ {(v, w)}
23:            Поиск (w)
24:        Конец Если
25:    Конец Для
26: Конец Алгоритм
```

Рис. 53: Поиск в глубину

Пусть $\mathfrak{G}_1 = (V_1, E_1)$ — связная компонента графа $\mathfrak{G}$ и $v_1 \in V_1$ — первая её вершина, для которой вызывается процедура Поиск (v_1). Тогда для каждой вершины $w \in V_1$ внутри вызова Поиск (v_1) произойдёт вызов Поиск (w).

Это утверждение доказывается индукцией по расстоянию (длине кратчайшего пути) от v_1 до w.

Базис индукции. Если это расстояние равно 1, то $w \in L_{v_1}$ и рассматривается в вызове Поиск (v_1) в строке 20. Если в этот момент $w \in R$, то, значит, Поиск (w) уже вызывался. Если же $w \notin R$, то в строке 23 происходит вызов Поиск (w).

Индукционный шаг. Допустим, что Поиск (u) вызывается для всех вершин u, находящихся на расстоянии $k \geqslant 1$ от вершины v_1, и пусть вершина $w \in L_{v_1}$ находится на расстоянии $k+1$ от v_1. Тогда имеется путь длины $k+1$ от v_1 до w. Пусть u — это предпоследняя вершина на этом пути. Тогда расстояние от v_1 до u равно k и по нашему предположению в некоторый момент выполняется вызов Поиск (u). Так как $w \in L_u$, то в этом вызове вершина w в некоторый момент рассматривается в цикле в строке 20. Как и выше, если этот момент $w \in R$, то Поиск (w) уже вызывался, а в противном случае в строке 23 происходит вызов Поиск (w).

Также по индукции замечаем, что если вызов Поиск (w) произошёл внутри вызова Поиск (v), то в T имеется путь из v в w. Следовательно, граф $\mathfrak{T}_1 = (V_1, T_1)$, построенный в процессе вызова Поиск (v), является деревом с корнем v. □

Дерево $\mathfrak{T} = (V, T)$, которое строится для графа $\mathfrak{G}$ при вызове алгоритма ПОГ $(\mathfrak{G})$, называется глубинным остовным деревом графа $\mathfrak{G}$. Рёбра графа $\mathfrak{G}$, попавшие в множество T, называются прямыми, а не попавшие — обратными. Каждое обратное ребро (v, w) соединяет вершину v с её предком w в глубинном остовном дереве (см. задачу 194 на стр. 250). Поэтому в исходном графе $\mathfrak{G}$ оно соответствует такому циклу: от w к v по рёбрам дерева T, а затем обратно от v к w по ребру (v, w). Следовательно, с помощью ПОГ $(\mathfrak{G})$ можно проверять наличие циклов в графе $\mathfrak{G}$. Ребро (v, w) не добавляется к T, то есть является обратным, тогда и только тогда,

когда в строке 21 вызова Поиск (v) обнаруживается, что вершина w уже принадлежит R. Поэтому, добавив в процедуру Поиск (v) в конец условного оператора (строки 21–24) ветку

Иначе Печать (v, w),

мы получим процедуру, которая в дополнение к построению к глубинного остовного дерева графа будет распечатывать список всех обратных рёбер. Если этот список не пуст, то в графе имеются циклы, иначе — нет.

Поиск в глубину позволяет обнаруживать не только циклы, но и многие другие важные части графов.

Определение 103 (Мост)**.** *Ребро* (v, w) *неориентированного графа* $\mathfrak{G} = (V, E)$ *называется мостом* $\mathfrak{G}$*, если при его удалении из* E *число связных компонент графа увеличивается, то есть в графе* $\mathfrak{G}' = (V, E \setminus \{(v, w)\})$ *связных компонент больше, чем в* $\mathfrak{G}$*.*

Из этого определения, в частности, следует, что ребро является мостом тогда и только тогда, когда оно не входит ни в какой цикл (почему?). Поиск в глубину позволяет находить все мосты графа. Во-первых, заметим, что любой мост (v, w) является ребром глубинного остова, так как другого пути, связывающего v и w, нет. Во-вторых, если это ребро ориентировано от v к w, то в глубинном остове нет обратных рёбер, соединяющих w и его потомки с предками w. Это условие является и достаточным, так как если таких рёбер нет, то удаление (v, w) нарушит связь между v и w и они окажутся в разных компонентах связности. Обозначим через $Up[w]$ минимум из $Num[w]$ и наименьшего из номеров вершин, к которым ведут обратные рёбра от вершин поддерева $\mathfrak{T}_w$. Тогда, учитывая, что обратные рёбра соединяют потомков с предками и что номера предков меньше номеров потомков, предложенный критерий можно переформулировать в следующем виде.

Теорема 82 (О поиске в глубину и мостах)**.** *Ребро* (v, w) *глубинного остовного дерева* $\mathfrak{T} = (V, T)$ *в неориентированном графе* $\mathfrak{G} = (V, E)$

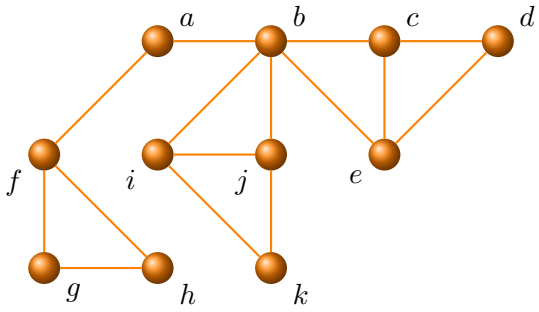


Рис. 54: Граф из примера 57.

является мостом тогда и только тогда, когда $Up[w] > Num[v]$ *или, что эквивалентно,* $Up[w] = Num[w]$.

Вычисление значения $Up[w]$ можно организовать в процессе обхода в глубину, используя следующее соотношение:

$$Up[w] = \min\big(\{Num[w]\} \cup \{\, Up[z] : (w,z) \text{ — прямое ребро}\} \cup \\ \cup \{Num[u] : (w,u) \text{ — обратное ребро}\}\big).$$

Для этого достаточно вначале создать ещё один массив $Up[V]$, добавить начальное присваивание $Up[v] \leftarrow Num[v]$ после строки 18, после вызова Поиск (w) в строке 23 вставить присваивание

$$Up[v] \leftarrow \min\{\, Up[v],\, Up[w]\},$$

учитывающее прямые рёбра, и вставить в условный оператор ветку:

$$\textbf{Иначе}\ Up[v] \leftarrow \min\{\, Up[v], Num[w]\}$$

для учёта обратных рёбер.

Зная значения $Up[w]$, нетрудно выявить все мосты, используя критерий из теоремы 82 на предыдущей странице.

Пример 57. *Рассмотрим вызов алгоритма* ПОГ $(\mathfrak{G})$ *для графа* $\mathfrak{G}$, *изображённого на рис. 54. Его представление в виде списков смежности имеет*

следующий вид:

$$\begin{array}{lll} L_a = (f, b); & L_e = (b, c, d); & L_i = (b, j, k); \\ L_b = (a, i, j, c, e); & L_f = (a, g, h); & L_j = (b, i, k); \\ L_c = (b, e, d); & L_g = (f, h); & L_k = (i, j). \\ L_d = (c, e); & L_h = (f, g); & \end{array}$$

Допустим, что при вызове ПОГ $(\mathfrak{G})$ *выбрана вершина* a *и запущено* Поиск (a). *Эта процедура рекурсивно вызовет* Поиск (f) *и т. д. Вот структура всех получающихся вызовов этой процедуры:*

$$\begin{array}{ccccccc} \text{Поиск}\,(a) & \Rightarrow & \text{Поиск}\,(f) & \Rightarrow & \text{Поиск}\,(g) & \Rightarrow & \text{Поиск}\,(h) \\ \Downarrow & & & & & & \\ \text{Поиск}\,(b) & \Rightarrow & \text{Поиск}\,(i) & \Rightarrow & \text{Поиск}\,(j) & \Rightarrow & \text{Поиск}\,(k) \\ \Downarrow & & & & & & \\ \text{Поиск}\,(c) & \Rightarrow & \text{Поиск}\,(e) & \Rightarrow & \text{Поиск}\,(d) & & \end{array}$$

Вначале идут «горизонтальные» вызовы, затем возвраты справа налево и вызовы «по вертикали». В результате вершины графа $\mathfrak{G}$ *получат следующие номера, отражающие порядок их прохождения:*

V:	a	b	c	d	e	f	g	h	i	j	k
Num:	1	5	9	11	10	2	3	4	6	7	8

Рёбра остова T, *построенные в процессе обхода графа, показаны на рисунке* 55 *на следующей странице. Стрелки указывают направление обхода. После двоеточия для каждой вершины указан её номер в массиве* Num, *то есть номер в порядке обхода алгоритмом.*

В процессе построения этого дерева были найдены следующие обратные рёбра: (h, f), (j, b), (k, i), (e, b) *и* (d, c). *Нетрудно проверить, что добавление любого из них к* T *приводит к образованию простого цикла.*

Используя расширенный вариант ПОГ *с вычислением функции* Up, *мы получим следующий результат:*

V:	a	b	c	d	e	f	g	h	i	j	k
Num:	1	5	9	11	10	2	3	4	6	7	8
Up:	1	5	5	9	5	2	2	2	5	5	6

Так как $Up[b] = Num[b] = 5$ *и* $Up[f] = Num[f] = 2$, *то по теореме* 82 *на стр. 241 мостами графа* $\mathfrak{G}$ *являются рёбра* (a, b) *и* (a, f) *и других мостов у него нет.*

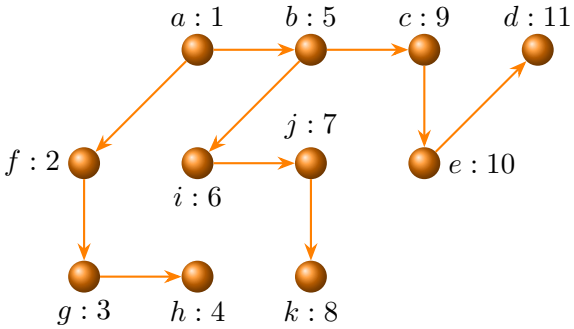


Рис. 55: Остовное «глубинное» дерево из примера 57.

Алгоритм поиска в глубину часто используется как основа для различных алгоритмов обработки графов. Вместо строки 18, в которой вершина v получает номер, можно вставить вызов любой процедуры, обрабатывающей информацию, связанную с этой вершиной (например, для задачи о лабиринте это может быть проверка того, что v является выходом из лабиринта). И тогда полученный вариант алгоритма обеспечит обработку всех вершин графа.

§ 12.3. Задача о кратчайших путях из одного источника

Пусть $\mathfrak{G} = (V, E)$ — размеченный ориентированный граф, для каждого ребра $e \in E$ которого указана его (неотрицательная) длина: $c(e) \geqslant 0$. Тогда длина пути $P = (v_1, v_2, \ldots, v_k)$ определяется как сумма длин рёбер, входящих в этот путь: $c(P) = \sum\limits_{i=1}^{k-1} c(v_i, v_{i+1})$. Если в $\mathfrak{G}$ имеется путь из вершины a в вершину b, то имеется и путь минимальной длины. Он называется кратчайшим путём из a в b. Конечно, в графе может оказаться несколько различных кратчайших путей из a в b.

Естественно спросить, как узнать длину кратчайшего пути из a в b и построить его? Лучшие известные на сегодняшний день алгоритмы,

отвечающие на этот вопрос, решают, на самом деле, даже более общую задачу построения всех кратчайших путей из одного источника: по вершине a найти длины кратчайших путей из a во все достижимые из неё вершины и построить для каждой из таких вершин некоторый кратчайший путь из a.

Если для каждой вершины $v \in V$, достижимой из a, зафиксировать один кратчайший путь из a в v, то получившийся граф будет представлять ориентированное дерево с корнем a (задача 203 на стр. 251). Это дерево называется д е р е в о м к р а т ч а й ш и х п у т е й и з a.

Мы рассмотрим алгоритм построения дерева кратчайших путей и определения их длин, предложенный в 1959 г. Э. Д е й к с т р о й (рис. 56 на следующей странице). Его идея следующая: перед каждым этапом множество R содержит вершины, для которых кратчайшие пути уже найдены. На очередном этапе к нему добавляется вершина w с самым коротким путём из a, проходящим по множеству R. После этого пересчитываются длины кратчайших путей из a в оставшиеся вершины из $V \setminus R$ с учётом новой вершины w. Длина текущего кратчайшего пути из a в v, проходящего по множеству R, заносится в ячейку $D[v]$ массива D. В конце работы в этом массиве отыскиваются длины соответствующих кратчайших путей. Для определения дерева кратчайших путей служит массив «отцов» F, его элемент $F[v]$ является отцом вершины v в дереве кратчайших путей.

Пример 58. *Рассмотрим работу этого алгоритма на нагруженном графе* $\mathfrak{G} = (V, E)$, $V = \{a, b, c, d, e, f\}$, *и выделенной вершине* $a \in V$. *Зададим длины рёбер матрицей* $C = (c_{uv})$, *где* $c_{uv} = c(u, v)$:

$$\begin{pmatrix} & a & b & c & d & e & f \\ a & 0 & 25 & 5 & 30 & \infty & 75 \\ b & 12 & 0 & \infty & \infty & 120 & 20 \\ c & \infty & 15 & 0 & 20 & 45 & 60 \\ d & \infty & \infty & \infty & 0 & 23 & 20 \\ e & \infty & \infty & 75 & 20 & 0 & 20 \\ f & 40 & 15 & 15 & 26 & \infty & 0 \end{pmatrix}$$

Сам граф изображён на рис. 57 *на стр. 247*.

```
1: Алгоритм КрПути(𝔊, c, a)          # Поиск кратчайших путей
2:     Вход: 𝔊 = (V, E) — граф
3:     Вход: c : V × V → ℝ — положительные веса рёбер
4:                                  # Если (v, u) ∉ E, то c(v, u) = ∞
5:     Вход: a ∈ V — исходная вершина
6:     Выход: F — массив отцов
7:     Выход: D — длины путей
8:     Создать массивы F[V], D[V]
9:     R ← {a}
10:    Для всех v ∈ V \ {a} выполнять
11:        D[v] ← c(a, v)
12:        Если c(a, v) < ∞ то
13:            F[v] ← a
14:        Иначе
15:            F[v] ← «–»
16:        Конец Если
17:    Конец Для
18:    Пока R ≠ V выполнять
19:        Выбрать вершину w ∈ V \ R с наименьшим D[w]
20:        R ← R ∪ {w}
21:        Для всех u ∈ V \ R выполнять
22:            Если D[w] + c(w, u) < D[u] то
23:                D[u] ← D[w] + c(w, u)
24:                F[u] ← w
25:            Конец Если
26:        Конец Для
27:    Конец Пока
28: Конец Алгоритм
```

Рис. 56: Алгоритм Дейкстры

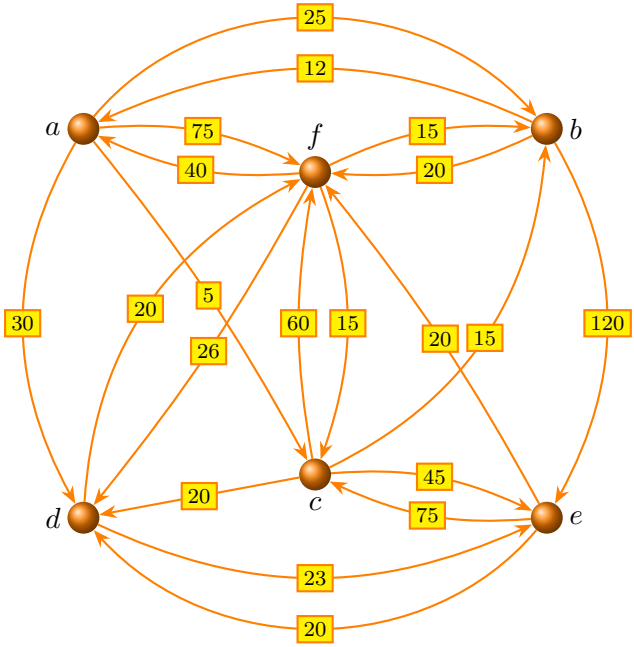


Рис. 57: Граф из примера 58.

Поэтапную работу алгоритма Дейкстры удобно представлять в виде таблицы (рис. 58 на следующей странице), строки которой соответствуют шагам выполнения основного цикла «Пока». Первый столбец — номер шага, второй показывает множество рассмотренных вершин R, третий — вершину w с минимальным расстоянием $D[w]$, добавляемую к R на следующем шаге, четвёртый — $D[w]$, затем идут столбцы со значениями элементов массивов D и F.

Дерево кратчайших путей из вершины a задаётся массивом F. Оно представлено на рис. 59 на следующей странице.

Теорема 83 (Корректность алгоритма Дейкстры)**.** *Алгоритм Дейкстры строит дерево кратчайших путей из вершины a во все достижимые из неё вершины и для каждой такой вершины v определяет длину $D[v]$ кратчайшего пути в неё из a.*

№	R	w	$D[w]$	D					F				
				b	c	d	e	f	b	c	d	e	f
1	a	c	5	25	5	30	∞	75	a	a	a	–	a
2	a, c	b	20	20	5	25	50	65	c	a	c	c	c
3	a, c, b	d	25	20	5	25	50	40	c	a	c	c	b
4	a, c, b, d	f	40	20	5	25	48	40	c	a	c	d	b
5	a, c, b, d, f	e	48	20	5	25	48	40	c	a	c	d	b

Рис. 58: Поэтапное построение кратчайших путей из примера 58.

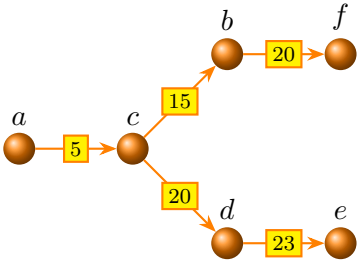


Рис. 59: Дерево кратчайших путей из примера 58.

Доказательство. Обозначим с помощью R_k, D_k и F_k значение R и содержимое массивов D и F соответственно после k шагов выполнения цикла «Пока», а w_k — вершину, выбранную на $(k+1)$-м шаге.

Докажем индукцией по k, что после каждого этапа алгоритма выполнены следующие условия:

(а) для каждой вершины $v \in R_k$ величина $D_k[v]$ равна длине кратчайшего пути из a в v;

(б) для каждой вершины $v \in V \setminus R_k$ величина $D_k[v]$ равна длине кратчайшего пути из a в v, который содержит только вершины из R_k (кроме, конечно, последней вершины v);

(в) для каждой вершины v дерева T_k, задаваемого массивом F_k, длина пути из корня a в v равна $D_k[v]$.

Базис индукции. После инициализации в строках 8–17, эти три условия очевидно выполняются.

Индукционный шаг. Предположим теперь, что они выполнены для k. По индукционному предположению $D_k[w_k]$ — длина кратчайшего пути из a в w_k, все вершины которого, кроме w_k, входят в R_k. Предположим, что есть другой более короткий путь P из a в w_k. Зафиксируем на этом пути первую вершину u, не входящую в R_k. По выбору P получаем $u \neq w_k$. Поэтому путь P разбивается на две непустые части: путь P_1 из a в u и путь P_2 из u в w_k. Но по выбору w_k мы имеем, что длина P_1 не меньше $D_k[u] \geqslant D_k[w_k]$. Так как длина P_2 неотрицательна, то длина P не меньше $D_k[w_k]$, то есть этот путь не короче пути, представленного в дереве T_k. Таким образом, $D_k[w_k]$ — это длина кратчайшего пути из a в w_k. Следовательно, условие (а) выполнено и для $k+1$.

Рассмотрим теперь произвольную вершину $u \in V \setminus (R_k \cup \{w_k\})$. Кратчайший путь P из a в u, проходящий по множеству $R_k \cup \{w_k\}$, либо не включает вершину w_k и в этом случае его длина равна $D_k[u]$ и он имеется в текущем дереве T_k, либо он проходит через w_k и составлен из кратчайшего пути из a в w_k через R_k, продолженного ребром (w_k, u). В последнем случае длина пути равна $D_k[w_k] + c(w_k, u)$. Но в строке 22 алгоритма эти величины сравниваются и, если путь через w_k короче, то его длина становится значением $D_{k+1}[u]$ (строка 23) и он фиксируется в дереве T_{k+1} (строка 24). Следовательно, условия (б) и (в) также выполнены для $k+1$.

Так как после завершения алгоритма $R = V$, то в завершающем дереве T представлены кратчайшие пути из a во все достижимые из неё вершины, а массив D содержит длины этих путей. Значение $D[u] = \infty$ указывает на то, что вершина u не достижима из a. □

Замечание 12 (О сложности алгоритма)**.** На каждом этапе (исполнении тела основного цикла в строках 18–27) одна вершина добавляется во множество R. Поэтому таких этапов не более $|V|$. Чтобы выбрать в массиве D вершину w с минимальным $D[w]$ (строка 19), требуется не более $|V|$

шагов. Обновление массива $D[u]$ для каждой из вершин $u \in V \setminus R$ требует константного числа операций, поэтому весь цикл в строках 21–26 потребует не более $O(|V|)$ шагов. Отсюда получаем, что время выполнения алгоритма Дейкстры не превышает $O(|V|^2)$. Поскольку размер любого представления исходного графа не меньше $|V|$, то алгоритм работает в квадратичное время от размера входа.

Задачи

190. Построить минимальное остовное дерево для неориентированного графа $\mathfrak{G} = (V, E)$ с множеством вершин $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$ и множеством рёбер $E = \{(v_1, v_2; 18), (v_1, v_3; 2), (v_3, v_2; 4), (v_3, v_4; 6), (v_3, v_5; 8), (v_4, v_6; 5), (v_5, v_4; 4), (v_6, v_1; 7), (v_6, v_8; 4), (v_6, v_7; 3), (v_7, v_5; 1), (v_7, v_8; 7), (v_8, v_1; 5), (v_8, v_9; 3), (v_9, v_1; 1)\}$. Третий параметр в скобках — вес ребра. ▼

191. Пусть $\mathfrak{T} = (V, T)$ — минимальное остовное дерево, построенное при вызове алгоритма МинОД$(\mathfrak{G})$ для нагруженного неориентированного графа $\mathfrak{G} = (V, E)$ с n вершинами. Пусть $c_1 \leqslant c_2 \leqslant \ldots \leqslant c_{n-1}$ — это последовательность длин рёбер из T, упорядоченных по возрастанию. Пусть $\mathfrak{T}'$ — произвольное остовное дерево для $\mathfrak{G}$ с длинами рёбер $d_1 \leqslant d_2 \leqslant \ldots \leqslant d_{n-1}$. Показать, что $c_i \leqslant d_i$ для всех $i = 1, \ldots, n-1$. ▼

192. Пусть e — ребро максимального веса в некотором цикле графа $\mathfrak{G} = (V, E)$. Доказать, что существует минимальное остовное дерево графа $\mathfrak{G}' = (V, E \setminus \{e\})$, который является также минимальным остовным деревом графа $\mathfrak{G}$. ▼

193. Недостатком алгоритма Крускала является необходимость проверки на каждом шаге наличия циклов, что само по себе является достаточно сложной задачей. Следующий алгоритм независимо предложен В. Ярником, Р. Примом и Э. Дейкстрой.

(а) Выбрать произвольным образом вершину a, она сама по себе является деревом (без рёбер).

(б) На каждом шаге добавлять к имеющемуся дереву ребро наименьшего веса, одна из вершин которого принадлежит дереву, а вторая — нет.

Доказать корректность этого алгоритма для связных графов по аналогии с теоремой 80 на стр. 234. ▼

194. Пусть $\mathfrak{T} = (V, T)$ — это глубинное остовное дерево, построенное алгоритмом обхода «в глубину» для графа $\mathfrak{G} = (V, E)$. Доказать, что для каждого обратного ребра $(u, v) \in E \setminus T$ или u является предком v в $\mathfrak{T}$, или v является предком u в $\mathfrak{T}$. ▼

195. Модифицировать алгоритм поиска в глубину так, чтобы он вычислял $Up[v]$ и распечатывал список всех мостов графа. ▼

196. Обойти (занумеровать) вершины заданного неориентированного графа $\mathfrak{G} = (V, E)$ с помощью алгоритма обхода «в глубину» и построить дерево этого обхода. $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$, $E = \{(v_1, v_2), (v_1, v_4), (v_1, v_8), (v_7, v_8), (v_2, v_9), (v_9, v_5), (v_3, v_8), (v_6, v_3), (v_3, v_7), (v_6, v_7), (v_{10}, v_9), (v_{10}, v_5)\}$. Какое обратное ребро $e \in E \setminus T$ и цикл в $\mathfrak{G}$ обнаружились в этом обходе первыми? Вычислить для каждой вершины v значение $Up[v]$ и определить все мосты графа $\mathfrak{G}$. Считать, что все списки смежности упорядочены по возрастанию номера вершины, а выбирается вершина с наименьшим номером. ▼

197. Изменить алгоритм обхода «в глубину» так, чтобы он позволил перечислить все связные компоненты неориентированного графа. ▼

198. Если в ориентированном графе $\mathfrak{G} = (V, E)$ существует вершина r, из которой достижимы все остальные, то для него тоже можно определить понятие остовного (теперь ориентированного) дерева: дерево $\mathfrak{T} = (V, T)$ с корнем r. Определить, какие из алгоритмов построения остовного дерева можно приспособить для построения остовного дерева в ориентированном графе: алгоритм Крускала, алгоритм Ярника-Прима-Дейкстры, алгоритм поиска в глубину. ▼

199. Определить для следующего нагруженного графа $\mathfrak{G} = (V, E)$ и вершины $a \in V$ длины кратчайших путей из a в остальные вершины $\mathfrak{G}$ и построить дерево этих путей. Здесь $V = \{a, b, c, d, e, f\}$, $E = \{(a, b; 154), (a, c; 17), (a, d; 214), (a, e; 63), (b, d; 25), (c, e; 33), (c, d; 192), (c, b; 123), (d, f; 5), (e, f; 140), (d, e; 10)\}$. ▼

200. Где в доказательстве правильности алгоритма Дейкстры используется неотрицательность весов рёбер? Привести пример графа (с отрицательными весами), для которого алгоритм Дейкстры даёт неверный ответ. ▼

201. Показать, что в доказательстве индукционного шага теоремы 83 на стр. 247 вершина w_k может быть только предпоследней в кратчайшем пути из a в $u \in V \setminus (R_k \cup \{w_k\})$. ▼

202. Сколько раз может меняться для одной вершины v значение $D[v]$ в ходе работы алгоритма Дейкстры для графа с шестью вершинами? Привести пример на каждый возможный случай. ▼

203. Пусть в графе $\mathfrak{G}$ выбрана вершина a и для каждой вершины $v \in V$, достижимой из a, существует единственный кратчайший путь из a в v. Доказать, что рёбра всех этих путей образуют ориентированное дерево с корнем a. ▼

Глава 13

Схемы из функциональных элементов

Краткое содержание: схемы из функциональных элементов и реализуемые ими функции, задачи синтеза и анализа схем, схемы и линейные программы, примеры схем: сложение по модулю два и двоичный сумматор.

Ключевые слова: схема из функциональных элементов, входная вершина, функциональный элемент, сложность схемы, глубина схемы, линейная (неветвящаяся) программа, сумматор порядка n.

В этой и следующей главах мы свяжем два основных предыдущих раздела нашего курса: булевы функции и графы. Мы рассматривали два основных представления булевых функций: табличное и с помощью формул общего вида или формул специального вида, в частности, дизъюнктивных или конъюнктивных нормальных форм и многочленов Жегалкина. К сожалению, эти способы не позволяют эффективно представлять функции от большого числа переменных: таблица для функции от n переменных всегда содержит 2^n строк,

многочлен Жегалкина может включать до 2^n слагаемых (и для большинства функций по порядку столько и включает). Такие представления нельзя реализовать на практике уже для n порядка нескольких десятков. Могло показаться, что сокращённые ДНФ, которые мы научились эффективно строить с помощью метода Блейка, и минимальные ДНФ, которые можно получить, удаляя из сокращённых «лишние» конъюнкции (впрочем, хороший алгоритм для такого удаления неизвестен), дают существенно более экономные представления булевых функций. Но в общем случае это не так. Для большинства булевых функций от n переменных минимальные ДНФ имеют экспоненциальный от n размер. В качестве примера конкретной простой функции с длинной ДНФ можно рассмотреть линейную функцию, определяющую нечётность суммы аргументов:

$$\mathrm{odd}(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$$

(см. задачу 204 на стр. 264).

Те два представления булевых функций, которые мы рассматриваем в этом и следующем разделах: схемы из функциональных элементов и упорядоченные бинарные диаграммы решений (УБДР), тоже для большинства функций имеют экспоненциальные размеры от числа переменных. Но во многих ситуациях они позволяют построить достаточно компактные представления естественно возникающих на практике булевых функций от сотен и даже тысяч аргументов.

§ 13.1. Схемы и булевы функции

Многие элементы в современной электронике являются устройствами, преобразующими некоторые входные сигналы (данные) в выходные. Схемы из функциональных элементов представляют собой математическую модель таких устройств, в которых временем выполнения преобразования входов в выходы можно пренебречь.

Чтобы не усложнять определение, зафиксируем конкретный базис $\mathcal{B}_0 = \{\wedge, \vee, \neg\}$ и определим схемы в этом базисе.

Определение 104 (Схема из функциональных элементов)**.** *Схемой из функциональных элементов (или говорят логической схемой) в базисе $\mathcal{B}_0$ называется размеченный ориентированный граф без циклов $\mathfrak{S} = (V, E)$, в котором*

1) *вершины, в которые не входят рёбра, называются входами схемы, и каждая из них помечена некоторой переменной (разным вершинам соответствуют разные переменные);*
2) *в каждую из остальных вершин входит одно или два ребра; вершины, в которые входит одно ребро, помечены функцией $\neg$, а вершины, в которые входят по два ребра, — одной из функций $\wedge$ или $\vee$. Такие вершины называются функциональными элементами.*

Как и для деревьев, для ориентированных графов без циклов можно естественным образом ввести понятие глубины.

Определение 105 (Глубина вершины, схемы)**.** *Глубина вершины $v \in V$ в схеме $\mathfrak{S} = (V, E)$ — это максимальная длина пути из входов $\mathfrak{S}$ в v.*

Глубиной $\operatorname{depth}(\mathfrak{S})$ *схемы $\mathfrak{S}$ назовём максимальную из глубин её вершин.*

С каждой вершиной $v \in V$ схемы S свяжем булеву функцию, реализуемую в этой вершине.

Определение 106 (Функция вершины)**.** *Пусть входы схемы $\mathfrak{S}$ помечены переменными $x_1, \ldots, x_n$. Определим f_v индукцией по глубине v.*

База индукции: вершина v имеет глубину ноль. Следовательно, v — это входная вершина, которая помечена некоторой переменной x_i. Положим $f_v(x_1, \ldots, x_n) = x_i$.

Шаг индукции: пусть всем вершинам w глубины не более k уже сопоставлены функции f_w, а v — произвольная вершина глубины $k + 1$. Тогда

1) *если v помечена $\neg$ и в неё входит ребро (w, v), то положим*

$$f_v(x_1, \ldots, x_n) = \neg f_w(x_1, \ldots, x_n);$$

2) *если v помечена $\wedge$ и в неё входят два ребра (w_1, v) и (w_2, v), то положим*

$$f_v(x_1, \ldots, x_n) = f_{w_1}(x_1, \ldots, x_n) \wedge f_{w_2}(x_1, \ldots, x_n);$$

3) *если v помечена $\vee$ и в неё входят два ребра (w_1, v) и (w_2, v), то положим*

$$f_v(x_1, \ldots, x_n) = f_{w_1}(x_1, \ldots, x_n) \vee f_{w_2}(x_1, \ldots, x_n).$$

Нетрудно понять, что шаг индукции в этом определении корректен, так как если в схеме S имеется ребро (w, v) и глубина вершины v равна $k + 1$, то глубина вершины w не превосходит k и для неё f_w уже определена по индукционному предположению.

Определение 107 (Реализация функции схемой)**.** *Схема $\mathfrak{S}$ реализует множество булевых функций $\{g_1, g_2, \ldots, g_m\}$, если для каждого $i = 1, \ldots, m$ в схеме существует такая вершина v_i, что $f_{v_i} = g_i$.*

Замечание 13. Определение схем естественным образом можно распространить и на другие базисы. При этом, однако, может оказаться, что вершина помечена функцией, значение которой зависит от порядка аргументов, например, импликацией. Для таких вершин нужно явно нумеровать входящие в них рёбра, указывая, каким аргументам они соответствуют.

Определение 108 (Сложность схемы, сложность булевой функции)**.** *Сложность схемы $\mathfrak{S}$ — это число функциональных элементов в $\mathfrak{S}$, обозначаем* $\mathrm{size}(\mathfrak{S})$. *Сложность булевой функции f — это наименьшая из сложностей схем, реализующих эту функцию, обозначаем* $\mathrm{size}(f)$.

Отношения между булевыми функциями и схемами естественно приводят к двум следующим основным проблемам:

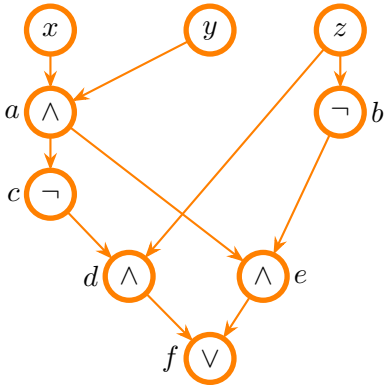


Рис. 60: Схема $\mathfrak{S}$ из примера 59.

Проблема анализа: по заданной схеме из функциональных элементов и выделенному подмножеству её выходных вершин определить булевы функции, реализуемые в этих вершинах.
Проблема синтеза: по некоторому описанию булевой функции построить схему из функциональных элементов, реализующую эту функцию. При решении проблемы синтеза часто стараются построить схему минимальной или почти минимальной сложности.

Пример 59. *Рассмотрим схему $\mathfrak{S}$ с тремя входными переменными x, y и z, изображённую на рис. 60 и решим для неё проблему анализа.*

В соответствии с данным выше определением вершины схемы $\mathfrak{S}$ реализуют следующие функции:

$$f_a(x,y,z) = x \wedge y; \quad f_b(x,y,z) = \neg z; \quad f_c(x,y,z) = \neg f_a(x,y,z) = \neg(x \wedge y);$$

$$f_d(x,y,z) = f_c(x,y,z) \wedge z = \neg(x \wedge y) \wedge z;$$

$$f_e(x,y,z) = f_a(x,y,z) \wedge f_b(x,y,z) = x \wedge y \wedge \neg z$$

и, наконец,

$$f_f(x,y,z) = f_d(x,y,z) \vee f_e(x,y,z) = (\neg(x \wedge y) \wedge z) \vee ((x \wedge y) \wedge \neg z).$$

Глубина этой схемы $\operatorname{depth}(\mathfrak{S}) = 4$, *а её сложность* $\operatorname{size}(\mathfrak{S}) = 6$. *В то же время формула для результирующей функции f_f содержит 7 функциональных знаков. За счёт чего достигнута экономия? За счёт того, что*

функция $(x \wedge y)$ в схеме $\mathfrak{S}$ вычисляется один раз в вершине a, а в формуле она встречается дважды.

В этом и состоит основное преимущество вычислений булевых функций схемами: каждую подформулу (подфункцию) достаточно вычислить один раз, а затем полученное значение можно использовать сколько угодно раз в качестве аргумента для других подфункций.

§ 13.2. Схемы и линейные программы

Указанное выше свойство характерно и для программ, в которых один раз вычисленное значение выражения можно использовать неоднократно. Рассмотрим один из простейших классов программ — линейные или неветвящиеся программы.

Определение 109 (Линейные программы)**.** *Пусть $\boldsymbol{V}$ — множество пропозициональных переменных, $\mathcal{B}$ — булевых функций (базис).*

Линейная (или неветвящаяся) программа — это последовательность слов вида $x \leftarrow f(x_1, \ldots, x_k)$, которые называют присваиваниями. Здесь $x, x_1, \ldots, x_k \in \boldsymbol{V}$ — переменные, $f^{(k)} \in \mathcal{B}$ — k-местная базисная функция.

В случае нашего базиса $\mathcal{B}_0 = \{\wedge, \vee, \neg\}$ линейная программа состоит из присваиваний вида: $z \leftarrow x \wedge y$, $z \leftarrow x \vee y$ и $z \leftarrow \neg x$.

Если в линейной программе Π выделить входные переменные $x_1, \ldots, x_n$, то для каждого набора их значений $\sigma_1, \ldots, \sigma_n$ порождается естественный процесс вычисления.

Определение 110 (Вычисление линейной программы)**.** *Для линейной программы $\Pi = (p_1, \ldots, p_n)$ и набора значений входных переменных $\sigma_1, \ldots, \sigma_n$ вычисление — это последовательность интерпретаций $(I_0, I_1, \ldots, I_n)$ такая, что*

1) *$I_0(x_i) = \sigma_i$ для входных переменных $x_1, \ldots, x_k$, $I(y) = 0$ для всех остальных переменных y;*
2) *если $p_i = u \leftarrow f_i(u_1, \ldots, u_k)$, то*

$$I_i(u) = f_i(I_{i-1}(u_1), \ldots, I_{i-1}(u_k))$$

и $I_i(y) = I_{i-1}(y)$ *для всех остальных переменных* y.

Таким образом, после выполнения каждого присваивания линейной программы значение соответствующей переменной меняется на значение правой формулы. После последнего из присваиваний каждая из переменных z программы Π получит заключительное значение, которое мы будем обозначать $\Pi_z(\sigma_1, \ldots, \sigma_n)$.

Определение 111 (Вычисление функции линейной программой)**.** *Скажем, что линейная программа* Π *с входными переменными* $x_1, \ldots, x_n$ *вычисляет функцию* $f(x_1, \ldots, x_n)$ *в переменной* z, *если для всякого набора значений входных переменных* $(\sigma_1, \ldots, \sigma_n)$ *выполнено* $\Pi_z(\sigma_1, \ldots, \sigma_n) = f(\sigma_1, \ldots, \sigma_n)$.

Вычисление значения любой формулы можно произвести с помощью линейной программы.

Предложение 84. *Для каждой формулы* Φ *в базисе* $\mathcal{B}$ *с переменными* $x_1, \ldots, x_n$ *существует линейная программа* Π_Φ, *вычисляющая значение этой формулы в какой-то переменной. При этом выполняются условия*

1) *значения переменных* $x_1, \ldots, x_n$ *не меняются;*
2) *каждая переменная* y, *кроме* $x_1, \ldots, x_n$, *встречается в* Π *первый раз в присваивании вида* $y \leftarrow e$, *где* e *не содержит* y.

Содержательно, второе требование означает, что мы не используем никаких переменных, кроме входных, до того, как явно присвоили им значение.

Доказательство. Используем индукцию по построению формулы. Если $\Phi = x_i$, то Π_Φ — пустая последовательность, так как в x_i изначально содержится значение Φ. Если $\Phi = c$ — константа (нульместная функция), то в качестве Π_Φ берём $y \leftarrow c$, тогда в y будет значение Φ. Пусть теперь $\Phi = f(\Psi_1, \ldots, \Psi_k)$ и для $\Psi_1, \ldots, \Psi_k$ уже построены программы $\Pi_{\Psi_1}, \ldots, \Pi_{\Psi_k}$, каждая Π_{Ψ_i} вычисляет значение Ψ_i в переменной y_i соответственно. Будем считать, что в программах $\Pi_{\Psi_1}, \ldots, \Pi_{\Psi_k}$ нет никаких общих переменных, кроме $x_1, \ldots, x_n$, этого можно добиться переименованием. Тогда для вычисления Φ можно

использовать программу $(\Pi_{\Psi_1}, \ldots, \Pi_{\Psi_k}, y \leftarrow f(y_1, \ldots, y_k))$, где y — новая переменная. Перед последним присваиванием по индукционному предположению переменные y_i содержат значения формул Ψ_i соответственно. Следовательно, в конце y будет содержать значение формулы Ψ. □

Обратное утверждение тоже верно.

Предложение 85. *Пусть* Π *— линейная программа с входными переменными* $x_1, \ldots, x_n$*, вычисляющая функцию* f *в переменной* y*, и никакая невходная переменная не используется до присваивания ей значения. Тогда существует формула* Φ_Π *с переменными* $x_1, \ldots, x_n$*, реализующая эту же функцию.*

Доказательство. Задача 205 на стр. 264. □

Между схемами и линейными программами имеется тесная связь.

Теорема 86. *Пусть* $\mathcal{B}$ *— базис.*

1) *По каждой схеме из функциональных элементов* $\mathfrak{S}$ *в базисе* $\mathcal{B}$ *со входами* $x_1, \ldots, x_n$ *и элементами* $v_1, \ldots, v_m$ *можно эффективно построить линейную программу* $\Pi_\mathfrak{S}$ *в базисе* $\mathcal{B}$ *со входными переменными* $x_1, \ldots, x_n$ *и рабочими переменными* $v_1, \ldots, v_m$*, которая в каждой переменной* $v_i, i = 1, \ldots, m$*, вычисляет функцию* $f_{v_i}(x_1, \ldots, x_n)$*.*

2) *Пусть базис* $\mathcal{B}$ *позволяет получить константу* 0*, а программа* Π *в базисе* $\mathcal{B}$ *кроме входных переменных* $x_1, \ldots, x_n$ *содержит* $z_1, \ldots, z_m$*, в каждой из которых вычисляется функция* $f_1, \ldots, f_m$ *соответственно. Тогда можно эффективно построить схему* $\mathfrak{S}_\Pi$ *в базисе* $\mathcal{B}$*, содержащую в том числе вершины* $v_1, \ldots, v_m$ *такие, что* $f_{v_i} = f_i$ *для* $i = 1, \ldots, m$*.*

Доказательство. Пусть $\mathfrak{S}$ — схема со входами $x_1, \ldots, x_n$ и функциональными элементами $v_1, \ldots, v_m$. Построим по ней линейную программу $\Pi_\mathfrak{S}$ со входными переменными $x_1, \ldots, x_n$ следующим образом. Упорядочим все функциональные вершины $\mathfrak{S}$ по глубине (вершины одной глубины располагаем в любом порядке): $v'_1, \ldots, v'_m$. Программа Π_S будет последовательностью m присваиваний.

(а) Пусть вершина v'_i помечена $\neg$ и в неё входит ребро из v'_j. Тогда в качестве i-й команды поместим в $\Pi_{\mathfrak{S}}$ присваивание $v'_i \leftarrow \neg v'_j$.

(б) Пусть вершина v'_i помечена $\circ \in \{\wedge, \vee\}$ и в неё входят рёбра из v'_j и v'_k. Тогда в качестве i-й команды поместим в $\Pi_{\mathfrak{S}}$ присваивание $v'_i = v'_j \circ v'_k$.

Упорядочение вершин по глубине гарантирует, что $j < i$ и $k < i$ в пунктах (а) и (б). Поэтому при вычислении v'_i значения аргументов уже получены и индукцией по глубине легко показать, что для каждого $i = 1, \ldots, m$ программа $\Pi_{\mathfrak{S}}$ вычисляет в переменной v'_i функцию $f_{v'_i}(x_1, \ldots, x_n)$.

Доказательство пункта 2) проведите самостоятельно (см. задачу 206 на стр. 264). □

Пример 60. *Применим конструкцию доказанной теоремы к схеме* $\mathfrak{S}$, *представленной на рис. 60 на стр. 256. Упорядочим её вершины по глубине так:* a, b, c, d, e, f. *Порождая присваивания по описанным в теореме правилам, получим такую линейную программу* $\Pi_{\mathfrak{S}}$:

$$\begin{aligned} a &\leftarrow x \wedge y \\ b &\leftarrow \neg z \\ c &\leftarrow \neg a \\ d &\leftarrow c \wedge z \\ e &\leftarrow a \wedge b \\ f &\leftarrow d \vee e \end{aligned}$$

Замечание 14. Количество присваиваний в линейной программе $\Pi_{\mathfrak{S}}$, то есть длина её вычисления, совпадает со сложностью $\mathrm{size}(\mathfrak{S})$ схемы $\mathfrak{S}$. Глубина схемы $\mathrm{depth}(\mathfrak{S})$ также имеет смысл с точки зрения времени вычисления. Именно, $\mathrm{depth}(\mathfrak{S})$ — это время выполнения $\Pi_{\mathfrak{S}}$ на многопроцессорной системе. Действительно, все команды, соответствующие вершинам одинаковой глубины, можно выполнять параллельно на разных процессорах, так как результаты любой из них не используются в качестве аргументов другой.

§ 13.3. Построение сумматора

Рассмотрим задачу сложения двух чисел в двоичной системе. Каждое из чисел представлено в виде набора двоичных разрядов, требуется получить двоичное представление их суммы.

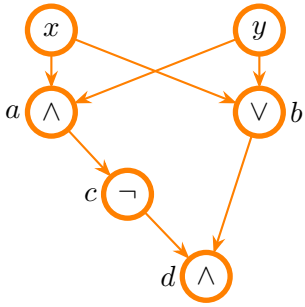


Рис. 61: Схема $\mathfrak{S}_\oplus$ для функции $x \oplus y$.

Сначала построим схему $\mathfrak{S}_\oplus$ для сложения по модулю два (рис. 61).

В соответствии с определением вершины этой схемы реализуют следующие функции:

$$f_a(x,y) = x \wedge y; \quad f_b(x,y) = x \vee y; \quad f_c(x,y) = \neg f_a(x,y) = \neg(x \wedge y);$$

$$f_d(x,y) = f_c(x,y) \wedge f_b(x,y) = \neg(x \wedge y) \wedge (x \vee y) = x \oplus y.$$

Таким образом, схема $\mathfrak{S}_\oplus$ реализует функцию $x \oplus y$ сложения по модулю два в вершине d. Попутно заметим, что в вершине a реализуется конъюнкция $x \wedge y$, мы в дальнейшем это тоже используем.

Из приведённого выше примера следует, что $\operatorname{size}(\mathfrak{S}_\oplus) = 4$, поэтому и $\operatorname{size}(\oplus) \leqslant 4$.

Используя схему $\mathfrak{S}_\oplus$, нетрудно построить схему $\mathfrak{S}_{\mathrm{odd}}$, показанную на рис. 62 на следующей странице, которая будет реализовывать линейную функцию — сумму n аргументов по модулю два:

$$\mathrm{odd}(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n.$$

Здесь прямоугольники $\mathfrak{S}_\oplus^{(1)}, \mathfrak{S}_\oplus^{(2)}, \ldots, \mathfrak{S}_\oplus^{(n)}$ содержат копии схемы $\mathfrak{S}_\oplus$. Входами $\mathfrak{S}_\oplus^{(1)}$ являются переменные x_1 и x_2, а входами $\mathfrak{S}_\oplus^{(i)}$ для $i = 2, \ldots, n-1$ являются выход схемы $\mathfrak{S}_\oplus^{(i-1)}$ и переменная x_{i+1}. По индукции легко показать, что вершина d в $\mathfrak{S}_\oplus^{(i)}$ реализует функцию $(x_1 \oplus x_2 \oplus \ldots \oplus x_{i+1})$. Таким образом, нами установлена

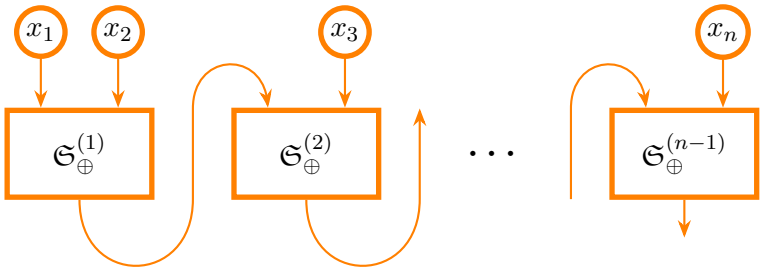


Рис. 62: Схема $\mathfrak{S}_{\mathrm{odd}}$.

Теорема 87. *Существует схема* $\mathfrak{S}_{\mathrm{odd}}$, *реализующая функцию*

$$\mathrm{odd}(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$$

со сложностью $\mathrm{size}(\mathfrak{S}_{\mathrm{odd}}) = 4(n-1)$.

Сумматором порядка n называют схему из функциональных элементов, вычисляющую результат сложения двух n-разрядных двоичных чисел $a = (a_{n-1} \ldots a_1 a_0)$ и $b = (b_{n-1} \ldots b_1 b_0)$. Пусть $c = a + b = (c_n c_{n-1} \ldots c_1 c_0)$. Здесь с помощью $a_i, b_i, c_i \in \{0, 1\}$ обозначены соответствующие двоичные разряды этих чисел):

$$\begin{array}{rc} & a_{n-1} \ldots a_1 \; a_0 \\ + & b_{n-1} \ldots b_1 \; b_0 \\ \hline & c_n \; c_{n-1} \ldots c_1 \; c_0 \end{array}$$

Поскольку результатом сложения двух n-разрядных чисел, вообще говоря, является $(n+1)$-разрядное число, то сумматор должен вычислять набор из $(n+1)$-й результирующей функции:

$$c_i(a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}), \quad i = 0, 1, \ldots, n,$$

задающих соответствующие разряды суммы c.

Обозначим через p_i бит переноса из $(i-1)$-го разряда в i-й. Тогда нетрудно видеть, что при $i = 0$

$$c_0 = a_0 \oplus b_0; \quad p_1 = a_0 \wedge b_0,$$

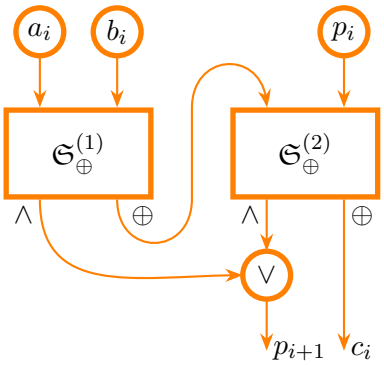


Рис. 63: Схема SUM_1.

а при $1 \leqslant i \leqslant n-1$

$$c_i = p_i \oplus a_i \oplus b_i; \quad p_{i+1} = (a_i \wedge b_i) \vee (p_i \wedge a_i) \vee (p_i \wedge b_i).$$

Старший разряд c совпадает с последним переносом: $c_n = p_n$.

Рассмотрим теперь построенную выше схему $\mathfrak{S}_{\oplus}$ как схему, вычисляющую набор из двух функций: $x \wedge y$ (в вершине a) и $x \oplus y$ (в вершине d). Нетрудно понять, что эта схема уже суммирует два одноразрядных числа: $c_0 = a \oplus b$, $c_1 = a \wedge b$.

Для того, чтобы учитывать бит переноса в старших разрядах, нужно построить схему суммирования с тремя входами. Это можно сделать, используя два экземпляра схемы $\mathfrak{S}_{\oplus}^{(1)}$ и $\mathfrak{S}_{\oplus}^{(2)}$ и дизъюнкцию. На рис. 63 показана схема SUM_1, которая имеет три входа a_i, b_i и p_i, $1 \leqslant i \leqslant n-1$, и вычисляет c_i и p_{i+1}.

Действительно, из построения следует, что в вершине p_{i+1} этой схемы вычисляется функция

$$f_{p_{i+1}} = (a_i \wedge b_i) \vee ((a_i \oplus b_i) \wedge p_i) = \\ = (a_i \wedge b_i) \vee (p_i \wedge a_i) \vee (p_i \wedge b_i) = p_{i+1}.$$

Из представленной схемы видно, что сложность этого сумматора $\mathrm{size}(\mathrm{SUM}_1) = 9$.

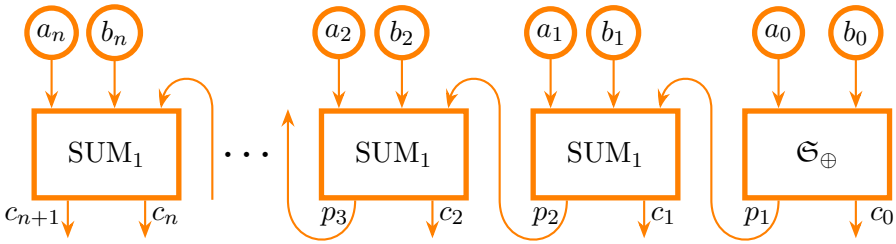


Рис. 64: Схема сумматора SUM_{n+1}.

Теперь из $\mathfrak{S}_\oplus$ и одноразрядных сумматоров SUM_1 соберём схему SUM_{n+1} для $(n+1)$-разрядного сумматора, показанную на рис. 64. Её правильность проверяется непосредственной индукцией по номерам разрядов результата.

Таким образом мы установили следующее утверждение.

Теорема 88. *Для каждого $n \geqslant 1$ существует схема SUM_n, реализующая операцию суммирования двух n-разрядных двоичных чисел и имеющая сложность* $\mathrm{size}(\mathrm{SUM}_n) = 9n - 5$.

Замечание 15. Логические схемы активно исследовались 50–70-х годах прошлого столетия. Например, К. Шеннон и О. Б. Лупанов нашли оценки сложности схем для булевых функций от n аргументов. Оказалось, что любую такую функцию можно реализовать со сложностью не большей (по порядку) $2^n/n$ и что «почти все» они имеют не меньшую сложность. При этом до сих пор не известна ни одна последовательность «конкретных» функций f_n, сложность которых по порядку превосходила бы линейную функцию.

Задачи

204. Доказать, что совершенная, сокращённая и минимальная ДНФ для функции $\mathrm{odd}(x_1, x_2, \ldots, x_n)$ совпадают и имеют 2^{n-1} элементарных конъюнкций длины n. ▼

205. Доказать предложение 85 на стр. 259. ▼

206. Доказать пункт 2) теоремы 86 на стр. 259. ▼

207. Доказать, что минимальная схема для сложения по модулю два имеет сложность $\mathrm{size}(\oplus) = 4$ в базисе $\mathcal{B}_0$.

208. Доказать, что в базисе $\mathcal{B}_0$ всякую булеву функцию f можно вычислить с помощью линейной программы, где присваивание выполняется не более чем трём переменным. Можно ли изменив базис уменьшить это количество до двух? ▼

209. Используя схему SUM_n, построить схему, реализующую операцию вычитания двух n-разрядных двоичных чисел: $d = a - b$ (при условии, что $a \geqslant b$). Оценить сложность полученной схемы. ▼

210. Определить глубину схем $\mathfrak{S}_{\oplus}$, $\mathfrak{S}_{\mathrm{odd}}$, SUM_1 и SUM_n. ▼

211. Два игрока независимо выбирают одно из четырёх чисел от 0 до 3. Первый игрок выигрывает, если выбранные числа совпадают. Построить схему, определяющую выигрыш первого игрока. Её входы x_1, x_2 представляют число, выбранное первым игроком, а y_1, y_2 — число, выбранное вторым игроком. Реализуемая функция $F(x_1, x_2, y_1, y_2)$ равна единице тогда и только тогда, когда $x_1 = y_1$ и $x_2 = y_2$. ▼

212. Построить схему для сравнения двух n-значных двоичных чисел. Схема должна иметь входы $a_{n-1}, \ldots, a_0$ и $b_{n-1}, \ldots, b_0$ для исходных чисел и три выхода результата: больше, меньше или равно. ▼

213. Построить схему для умножения двух двухзначных двоичных чисел. Схема должна иметь входы a_1, a_0 и b_1, b_0 для исходных чисел и четыре выхода для разрядов результата. ▼

214. Построить схему, определяющую результат голосования в комитете, состоящем из трёх членов и председателя. В случае равенства голосов, голос председателя является решающим. ▼

215. Пусть наборы аргументов булевой функции от трёх аргументов упорядочены лексикографически, а её значения задаются последовательностью из восьми нулей и единиц. Построить схемы, реализующие следующие функции:

(а) $f_1 = (1111\,1011)$;

(б) $f_2 = (1001\,1001)$;

(в) $f_3 = (0011\,1001)$. ▼

Глава 14

Упорядоченные бинарные диаграммы решений

Краткое содержание: бинарные деревья решений и их превращение в упорядоченные бинарные диаграммы решений (УБДР), сокращённые УБДР и их построение по произвольным УБДР, алгоритм сокращения УБДР, построение сокращённых УБДР по формулам.

Ключевые слова: бинарное дерево решений, упорядоченная бинарная диаграмма решений (УБДР), сложность УБДР, сток, внутренняя вершина, порядок переменных, сокращённая УБДР, правило сокращения, правило слияния.

Рассматриваемый в этой главе способ представления булевых функций с помощью специального подкласса ориентированных графов без циклов был предложен Р. Брайантом (R. Bryant) в 1986 г. Его английское название — «Ordered binary decision diagram», сокращённо — OBDD. Сейчас УБДР являются одним из основных средств реализации булевых функций от большого числа переменных в задачах искусственного интеллекта, проверки правильности электронных схем, программ, протоколов и т. д.

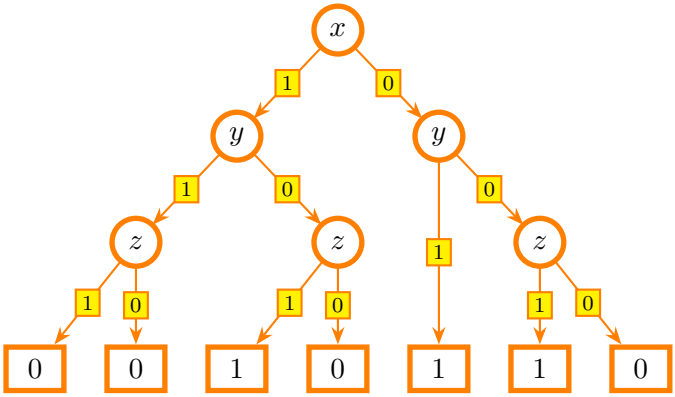


Рис. 65: Бинарное дерево решений $\mathfrak{T}_1$.

§ 14.1. Основные определения

Одним из предшественников УБДР являются бинарные деревья решений.

Определение 112 (Бинарное дерево решений)**.** *Пусть зафиксировано множество пропозициональных переменных* $\boldsymbol{V}$. *Бинарное дерево решений (БДР) — это размеченное бинарное дерево* $\mathfrak{T} = (V, E)$, *все внутренние вершины которого помечены переменными из* $\boldsymbol{V}$, *а листья — константами* 0 *или* 1. *Из каждой внутренней вершины* v *выходят в точности* 2 *ребра, одно помечено* 0, *другое —* 1; *вершина* w_0, *в которую ведёт ребро, помеченное* 0, *называется* 0-*сыном* v, *а вершина* w_1, *в которую ведёт ребро, помеченное* 1, *называется* 1-*сыном* v.

Пусть $\boldsymbol{V} = \{x_1, \ldots, x_n\}$. *Бинарное дерево решений* $\mathfrak{T}$ *реализует булеву функцию* $f(x_1, \ldots, x_n)$, *если для каждого набора значений* $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ *переменных ветвь в дереве, соответствующая этому набору (из помеченной* x_i *вершины идём к* σ_i-*сыну), завершается листом с меткой* $f(\sigma_1, \sigma_2, \ldots, \sigma_n)$.

Пример 61. *Рассмотрим БДР* $\mathfrak{T}_1$, *изображённое на рис.* 65.

x	y	z	$f_1(x,y,z)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Рис. 66: Функция $f_1(x,y,z)$, реализуемая БДР $\mathfrak{T}_1$.

По определению $\mathfrak{T}_1$ реализует функцию $f_1(x,y,z)$, которая представлена в виде таблицы на рис. 66. Чтобы, например, определить значение функции $f_1(1,1,0)$, мы рассматриваем такой путь в дереве: $x \overset{1}{\to} y \overset{1}{\to} z \overset{0}{\to} 0$.

Нетрудно построить ДНФ этой функции:

$$f_1(x,y,z) = (\neg x \wedge y) \vee (\neg y \wedge z).$$

Упорядоченные бинарные диаграммы решений являются модификацией БДР, в которой все листья с одной меткой «склеены» в одну вершину, в каждую вершину может входить несколько рёбер, и порядок следования переменных на ветвях может не совпадать с порядком аргументов функции.

Определение 113 (Упорядоченная бинарная диаграмма решений)**.** *Пусть зафиксирован некоторый порядок на множестве $\boldsymbol{V}$ из n пропозициональных переменных $\pi : (x_{\pi(1)}, \ldots, x_{\pi(n)})$.*

Упорядоченная бинарная диаграмма решений (УБДР) относительно порядка переменных π — это размеченный ориентированный мультиграф $\mathfrak{D}$ без циклов с одним истоком, в котором

1) *существуют лишь два стока, они помечены константами* 0 *и* 1*;*

2) *остальные (внутренние) вершины помечены переменными* $\boldsymbol{V}$*, из каждой из них выходят два ребра, одно из которых имеет метку* 0*, другое —* 1*;*

3) *порядок, в котором переменные встречаются на каждом пути, совместим с* π*, то есть если из вершины, помеченной* $x_{\pi(i)}$*, есть путь в вершину, помеченную* $x_{\pi(j)}$*, то* $i < j$*.*

Как и в случае БДР, УБДР реализует булеву функцию $f(x_1, \ldots, x_n)$*, если для каждого набора значений* $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ *переменных путь в диаграмме, начинающийся в истоке и соответствующий этому набору (из помеченной* x_i *вершины идём к* σ_i*-сыну), завершается стоком с меткой* $f(\sigma_1, \sigma_2, \ldots, \sigma_n)$*.*

Из этого определения непосредственно следует, что каждая внутренняя вершина v диаграммы $\mathfrak{D}$, помеченная переменной $x_{\pi(k)}$, является истоком поддиаграммы $\mathfrak{D}_v$, которая включает все вершины диаграммы $\mathfrak{D}$, достижимые из v, и реализует некоторую функцию $f_v(x_{\pi(k)}, x_{\pi(k+1)}, \ldots, x_{\pi(n)})$, зависящую от $n - k + 1$ переменной $x_{\pi(k)}, x_{\pi(k+1)}, \ldots, x_{\pi(n)}$. При этом её 0-сын w_0 является истоком поддиаграммы $\mathfrak{D}_{w_0}$, реализующей функцию

$$f_{w_0}(x_{\pi(k+1)}, \ldots, x_{\pi(n)}) = f_v(0, x_{\pi(k+1)}, \ldots, x_{\pi(n)}),$$

а 1-сын w_1 — исток поддиаграммы $\mathfrak{D}_{w_1}$, реализующей функцию

$$f_{w_1}(x_{\pi(k+1)}, \ldots, x_{\pi(n)}) = f_v(1, x_{\pi(k+1)}, \ldots, x_{\pi(n)}).$$

Пусть диаграмма реализует некоторую булеву функцию

$$f(x_1, \ldots, x_n) = f'(x_{\pi(1)}, \ldots, x_{\pi(n)})$$

и $\sigma_{\pi(1)}, \ldots, \sigma_{\pi(k-1)}$ — это набор значений переменных $x_{\pi(1)}, \ldots, x_{\pi(k-1)}$, который соответствует пути из истока в вершину v (таких наборов может быть несколько). Тогда

$$f_v(x_{\pi(k)}, \ldots, x_{\pi(n)}) = f'(\sigma_{\pi(1)}, \ldots, \sigma_{\pi(k-1)}, x_{\pi(k)}, \ldots, x_{\pi(n)}).$$

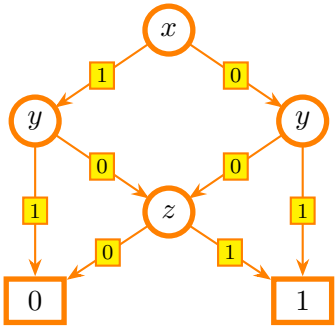


Рис. 67: УБДР $\mathfrak{D}_1$ для функции $f_1(x, y, z)$.

Пример 62. *Реализуем с помощью УБДР функцию $f_1(x, y, z)$, представленную выше в примере 61 на стр. 267, с помощью БДР $\mathfrak{T}_1$ и таблицы на рис. 66 на стр. 268.*

Вначале зафиксируем порядок переменных: $x < y < z$. Объединив листья с одинаковыми метками и две z-вершины с одинаковыми потомками, получим УБДР $\mathfrak{D}_1$, приведённую на рис. 67.

Ясно, что реализация функции $f_1(x, y, z)$ с помощью УБДР $\mathfrak{D}_1$ намного компактнее, чем с помощью БДР $\mathfrak{T}_1$.

Под с л о ж н о с т ь ю $\mathrm{size}(\mathfrak{D})$ УБДР $\mathfrak{D}$ будем понимать количество внутренних вершин $\mathfrak{D}$. Например, $\mathrm{size}(\mathfrak{D}_1) = 4$. Может ли сложность диаграммы для некоторой функции зависеть от порядка переменных? Да! Рассмотрим порядок переменных $y < x < z$. Как видно из рис. 68 на противоположной странице, относительно этого порядка функцию $f(x, y, z)$ можно реализовать УБДР $\mathfrak{D}_2$ со сложностью $\mathrm{size}(\mathfrak{D}_2) = 3$.

§ 14.2. Сокращённые УБДР

Если порядок переменных фиксирован, то по любой УБДР нетрудно построить минимальную УБДР, реализующую ту же функцию.

Определение 114 (Сокращённая УБДР)**.** *УБДР называется с о к р а щ ё н н о й если*

1) *у всякой внутренней вершины v её 0-сын и 1-сын не совпадают;*

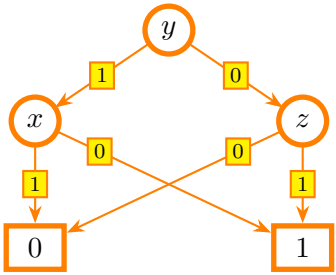


Рис. 68: УБДР $\mathfrak{D}_2$ для функции $f_1(x, y, z)$.

2) *нет такой пары внутренних вершин u и v, для которых поддиаграммы с истоками u и v являются изоморфными (то есть взаимно однозначно отображаются друг на друга с сохранением всех меток).*

Смысл этого определения понятен: если из вершины v оба ребра ведут в одну вершину, то такая вершина v не нужна, а если имеются две вершины с одинаковыми поддиаграммами, то их можно слить.

Определим два типа эквивалентных преобразований УБДР, соответствующих этому определению:

- Правило сокращения: если 0-сын и 1-сын вершины v совпадают и равны w, то удалить v, перенаправив все входящие в неё рёбра в вершину w (рис. 69 на следующей странице).
- Правило слияния: если вершины v и w помечены одной переменной и имеют одинаковых 0-сыновей и 1-сыновей, то удалить вершину v, перенаправив все входящие в неё рёбра в вершину w (рис. 70 на следующей странице).

Удаляемые вершины на обоих рисунках показаны тёмно-голубым цветом.

Теорема, которую мы собираемся доказать, скажет, что неприменимость этих двух правил и является критерием несокращаемости УБДР. Прежде докажем вспомогательное утверждение.

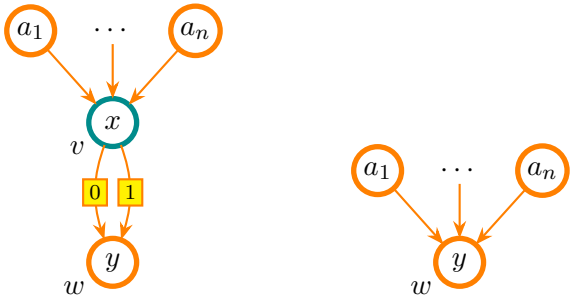


Рис. 69: Правило сокращения

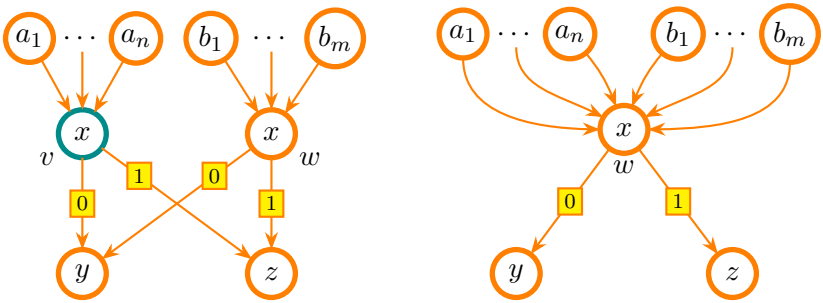


Рис. 70: Правило слияния

Лемма 89. *Если в УБДР $\mathfrak{D}$ есть такая пара вершин u и w, для которых поддиаграммы $\mathfrak{D}_v$ и $\mathfrak{D}_w$ с истоками v и w являются изоморфными, то в $\mathfrak{D}$ имеется и пара вершин v', w' с попарно одинаковыми* 0- *и* 1*-сыновьями и, следовательно, к $\mathfrak{D}$ применимо правило слияния.*

Доказательство. Так как поддиаграммы $\mathfrak{D}_v$ и $\mathfrak{D}_w$ изоморфны, то их высоты, то есть длины максимальных путей из истоков до стоков, одинаковы. Для доказательства леммы применим индукцию по высоте h этих поддиаграмм.

Базис индукции: $h = 1$. В этом случае 0- и 1-сыновьями вершин v и w являются одинаковые стоки.

Шаг индукции: допустим, утверждение верно для $h = k$. Пусть $\mathfrak{D}_v$ и $\mathfrak{D}_w$ — поддиаграммы высоты $h = k + 1$, v_0 и w_0 — это 0-

сыновья вершин v и w, а v_1 и w_1 — их 1-сыновья соответственно. Если $v_0 = w_0$ и $v_1 = w_1$, то в качестве v' и w' можно взять сами вершины v и w. Если же $v_i \neq w_i$ для некоторого $i \in \{0, 1\}$, то поддиаграммы $\mathfrak{D}_{v_i}$ и $\mathfrak{D}_{w_i}$ с истоками v_i и w_i являются изоморфными и имеют высоту k. Тогда утверждение леммы будет выполнено по индукционному предположению. □

Теорема 90. *УБДР $\mathfrak{D}$ является сокращённой, если и только если к ней не применимы правила ни слияния, ни сокращения.*

Доказательство. Слева направо. Если к $\mathfrak{D}$ применимо правило сокращения, то не выполнено условие 1) из определения сокращённой УБДР, а если к $\mathfrak{D}$ применимо правило слияния, то поддиаграммы с истоками v и w являются изоморфными и не выполнено условие 2).

Справа налево. Пусть к УБДР $\mathfrak{D}$ нельзя применить правило сокращения. Тогда в ней нет вершин с совпадающими 0- и 1-сыновьями и выполнено условие 1). Пусть к УБДР $\mathfrak{D}$ нельзя применить правило слияния. Тогда из леммы 89 на стр. 271 можно заключить, что в $\mathfrak{D}$ нет пары вершин, поддиаграммы которых являются изоморфными, и, следовательно, выполнено условие 2). □

Из теоремы 90 непосредственно следует, что, применяя к произвольной УБДР правила сокращения и слияния, мы, в конце концов, получим сокращённую УБДР. Чтобы эта процедура работала эффективно, нужно применять правила в порядке «снизу вверх». Будем считать, что в диаграмме принят «естественный» порядок переменных: $(x_1, \ldots, x_n)$, тогда построение сокращённой УБДР можно реализовать как показано на рис. 71 на следующей странице.

Пример 63. *Рассмотрим пример применения алгоритма* СокрУБДР, *показанный на рис. 72 на стр. 275. Здесь последовательно изображены стадии преобразования УБДР.*

На исходной УБДР слева все вершины уже занумерованы. При первом исполнении цикла 6–31 имеем $i = 3$, $W[3] = \{v_3\}$. Для вершины v_3 условие в строке 11 будет выполнено ($w_0 = w_1 = v_6$), поэтому применяется правило сокращения, вершина v_3 удаляется, а входящие в неё рёбра будут перенаправлены в вершину v_6.

При следующем исполнении цикла $i = 2$, $W[2] = \{v_2, v_4\}$. После цикла в строках 8–18 получаем $K[v_2] = (5, 6)$ и $K[v_4] = (5, 6)$. После сортировки

```
1: Алгоритм СокрУБДР(𝔇)                    # Сокращение УБДР 𝔇
2:     Вход: 𝔇 — УБДР для функции f(x_1, ..., x_n)
3:     Вход: V = {v_1, ..., v_m} — множество вершин 𝔇
4:     Выход: 𝔇 — сокращённая УБДР
5:     Создать массивы W[1, ..., n] и K[V]
6:     Для всех i ← n, ..., 1 выполнять
7:         W[i] ← {v : v ∈ V имеет метку x_i}
8:         Для всех v ∈ W[i] выполнять          # Сокращения
9:             w_0 ← 0-сын v
10:            w_1 ← 1-сын v
11:            Если w_0 = w_1 то
12:                W[i] ← W[i] \ {v}
13:                заменить в 𝔇 все рёбра вида (u, v) на (u, w_0)
14:                удалить v из 𝔇
15:            Иначе
16:                K[v] ← (w_0, w_1)
17:            Конец Если
18:        Конец Для
19:        отсортировать W[i] по значению K[v]
20:        k ← (∅, ∅)
21:        Для всех u ∈ W[i] выполнять          # Слияния
22:            Если k = K[u] то
23:                W[i] ← W[i] \ {u}
24:                заменить в 𝔇 все рёбра вида (t, u) на (t, w)
25:                удалить u из 𝔇
26:            Иначе
27:                w ← u
28:                k ← K[u]
29:            Конец Если
30:        Конец Для
31:    Конец Для
32: Конец Алгоритм
```

Рис. 71: Алгоритм сокращения УБДР.

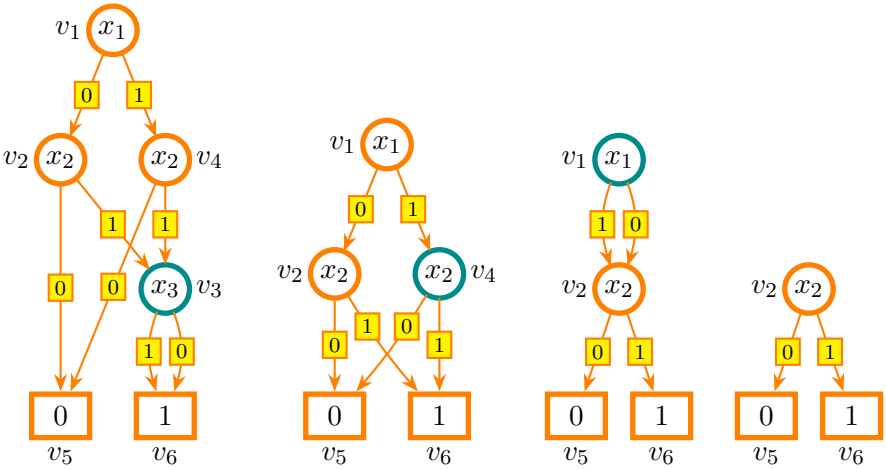


Рис. 72: Применение алгоритма сокращения УБДР.

$W[2] = (v_2, v_4)$. *В цикле в строках 21–30 для* v_4 *будет выполнено условие в строке 22. Поэтому применяется правило слияния и вершина* v_4 *удаляется, а её входы передаются вершине* v_2.

При третьем исполнении цикла $i = 1$, $W[1] = \{v_1\}$. *Для вершины* v_1 *условие в строке 11 выполнено* ($w_0 = w_1 = v_2$), *поэтому применяется правило сокращения и эта вершина удаляется.*

Оказывается, что построенная алгоритмом УБДР является единственной и минимальной для заданного порядка.

Теорема 91. *Пусть алгоритм* СокрУБДР *на входе* $\mathfrak{D}$ *возвращает УБДР* $\mathfrak{D}'$. *Тогда*

1) *УБДР* $\mathfrak{D}'$ *является сокращённой и эквивалентна* $\mathfrak{D}$;
2) *УБДР* $\mathfrak{D}'$ *является при данном порядке переменных единственной сокращённой (с точностью до изоморфизма) и минимальной.*

Доказательство. Справедливость первого пункта непосредственно вытекает из выполнения критерия теоремы 90 на стр. 273, так как к результирующей диаграмме никакое правило сокращения или слияния уже применить нельзя.

Второй пункт доказывается с помощью следующей леммы. □

Лемма 92. *После выполнения шага для i в полученной диаграмме для каждого $j = i-1, \ldots, n$ и для каждой подфункции*

$$f_{\sigma_1,\ldots,\sigma_j}(x_{j+1}, \ldots, x_n) = f(\sigma_1, \ldots, \sigma_j, x_{j+1}, \ldots, x_n),$$

где $\sigma_k \in \{0, 1\}$ при $k = 1, 2, \ldots, j$, имеется ровно одна вершина — исток поддиаграммы, реализующей эту подфункцию.

Доказательство этой леммы и вывод из неё утверждения 2) предыдущей теоремы оставляем в качестве задач 216 и 217 на стр. 278.

§ 14.3. Построение сокращённых УБДР по формулам

Алгоритм СокрУБДР позволяет построить сокращённую УБДР для функции f по любой другой её УБДР. Но как построить УБДР, если f задана, например, с помощью формулы? Можно, конечно, попытаться построить полное бинарное дерево решений, объединить в нём все листья с меткой 0 в один сток, а листья с меткой 1 — в другой. Затем применить к получившейся УБДР алгоритм СокрУБДР. Но этот подход годится только для функций от небольшого числа переменных, так как полное БДР для $f(x_1, \ldots, x_n)$ будет содержать 2^n листьев.

Другой подход связан с построением УБДР «сверху-вниз». Алгоритм такого построения является рекурсивным и предназначен для решения даже более общей задачи: построить одну УБДР, в которой реализуется сразу несколько заданных функций $f_1, \ldots, f_k$ с одними и теми же аргументами $x_1, \ldots, x_n$ (такая УБДР может иметь несколько истоков для реализации разных функций). Результатом алгоритма будет УБДР $\mathfrak{D}_n$ и список вершин $v_1, \ldots, v_k$, реализующих функции $f_1, \ldots, f_k$ соответственно. Будем предполагать, что зафиксирован естественный порядок переменных: $x_1 < x_2 < \ldots < x_n$.

Если $n = 0$, то все функции являются константами и УБДР состоит только из двух вершин: 0 и 1, которые являются одновременно и стоками, и истоками, реализующими функции-константы.

Пусть теперь $n > 0$. Для каждой функции f_i, $i = 1, \ldots, k$, построим две «остаточные» $(n-1)$-местные функции, получаемые из f_i фиксированием первого аргумента:

$$f_{i,0}(x_2, \ldots, x_n) = f_i(0, x_2, \ldots, x_n);$$
$$f_{i,1}(x_2, \ldots, x_n) = f_i(1, x_2, \ldots, x_n).$$

Удалим повторы одинаковых функций среди $\{f_{1,0}, f_{1,1}, \ldots, f_{k,0}, f_{k,1}\}$ и применим рекурсивно алгоритм к полученному множеству $(n-1)$-местных функций. Результатом будет УБДР $\mathfrak{D}_{n-1}$ и некоторые её вершины $v_{1,0}, v_{1,1}, \ldots, v_{k,0}, v_{k,1}$. Если $f_{i,0} = f_{i,1}$, то функция f_i реализуется в той же вершине УБДР, что и функции $f_{i,0} = f_{i,1}$, то есть $v_i = v_{i,0} = v_{i,1}$. Если $f_{i,0} \neq f_{i,1}$, то для реализации f_i добавляем в $\mathfrak{D}_{n-1}$ новую вершину v_i, 0- и 1-сыновьями v_i будут вершины $v_{i,0}$ и $v_{i,1}$, реализующие функции $f_{i,0}$ и $f_{i,1}$ соответственно.

Пример 64. *Рассмотрим функцию $f(x_1, x_2, x_3, x_4)$, заданную формулой*

$$(x_1 \wedge x_2 \wedge x_4) \vee (\neg x_1 \wedge x_2 \wedge \neg x_4) \vee (\neg x_2 \wedge x_3) \vee (\neg x_2 \wedge x_4),$$

и построим для неё УБДР относительно порядка $x_1 < x_2 < x_3 < x_4$, используя описанную выше процедуру.

Строим остаточные функции, получающиеся при $x_1 = 0$ и $x_1 = 1$:

$$f_0(x_2, x_3, x_4) = (x_2 \wedge \neg x_4) \vee (\neg x_2 \wedge x_3) \vee (\neg x_2 \wedge x_4),$$
$$f_1(x_2, x_3, x_4) = (x_2 \wedge x_4) \vee (\neg x_2 \wedge x_3) \vee (\neg x_2 \wedge x_4).$$

Они различны, поэтому для каждой из них строим остаточные функции при $x_2 = 0$ и $x_2 = 1$:

$$f_{00}(x_3, x_4) = (x_3 \vee x_4), \qquad f_{10}(x_3, x_4) = (x_3 \vee x_4),$$
$$f_{01}(x_3, x_4) = \neg x_4, \qquad f_{11}(x_3, x_4) = x_4.$$

Теперь $f_{00} = f_{10}$, поэтому далее строим остаточные функции только для f_{00}, f_{01} и f_{11} при $x_3 = 0$ и $x_3 = 1$:

$$f_{000}(x_4) = x_4, \qquad f_{010}(x_4) = \neg x_4, \qquad f_{110}(x_4) = x_4,$$
$$f_{001}(x_4) = 1, \qquad f_{011}(x_4) = \neg x_4, \qquad f_{111}(x_4) = x_4.$$

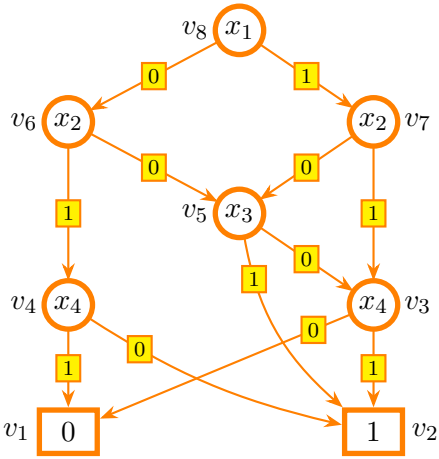


Рис. 73: УБДР $\mathfrak{D}_4$ для функции f из примера 64.

Осталось три различные функции: f_{000}, f_{001} *и* f_{010}. *На последнем шаге получатся только две константы:*

$$f_{0000} = f_{0101} = 0, \qquad f_{0001} = f_{0010} = f_{0011} = f_{0100} = 1.$$

На этом этапе мы создаём $\mathfrak{D}_0$ *из двух вершин:* v_1, *реализующей* $f_{0000} = f_{0101}$, *и* v_2, *реализующей* $f_{0001} = f_{0010} = f_{0011} = f_{0100}$.

Вершина v_2 *также реализует* f_{001}, *поскольку* $f_{0010} = f_{0011}$, *а для реализации* f_{000} *и* f_{010} *создадим новые вершины* v_3 *и* v_4 *соответственно. Поскольку* $f_{010} = f_{011}$ *и* $f_{110} = f_{111}$, *то вершина* v_4 *реализует и* f_{01}, *а вершина* v_3 — f_{11}. *Для реализации* f_{00} *нужно создать новую вершину* v_5. *Для* f_0 *и* f_1 *снова понадобятся новые вершины* v_6 *и* v_7. *Наконец, последним шагом будет добавлена вершина* v_8 *для реализации исходной функции* f. *В результате получается УБДР* $\mathfrak{D}_4$, *показанная на рис. 73.*

Задачи

216. Доказать лемму 92 на стр. 276 обратной индукцией по i. ▼

217. Используя лемму 92 на стр. 276, доказать утверждение 2) теоремы 91 на стр. 275. ▼

218. Доказать, что всякую булеву функцию от n переменных можно реализовать в виде УБДР с не более чем $2^{n/2+2} - 1$ внутренними вершинами. Указание: показать, как преобразовать полное БДР. ▼

219. Построить минимальные УБДР для двухместных функций: $x \wedge y$, $x \vee y$, $x \oplus y$, $x \to y$, $x \uparrow y$. ▼

220. Построить минимальные УБДР для функции

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = (x_1 \wedge x_2) \oplus (x_3 \wedge x_4) \oplus (x_5 \wedge x_6)$$

относительно двух упорядочений переменных:

(а) $x_1 < x_2 < x_3 < x_4 < x_5 < x_6$ и

(б) $x_1 < x_3 < x_5 < x_2 < x_4 < x_6$. ▼

221. Значение пороговой функции T_k^n от n переменных с порогом k равно 1 тогда и только тогда, когда во входном наборе имеется не менее k единиц:

$$T_k^n(x_1, x_2, \ldots, x_n) = 1 \iff [x_1 + x_2 + \ldots + x_n \geqslant k].$$

(а) Построить минимальные УБДР для пороговых функций T_2^3, T_2^4, T_3^5.

(б) Зависит ли сложность минимальной УБДР для пороговых функций от порядка переменных?

(в) Оценить сложность минимальной УБДР для пороговой функции T_k^n. ▼

222. Выбрать подходящий порядок переменных и построить для него минимальные УБДР, реализующие функции из задач (а) 211 на стр. 265 и (б) 214 на стр. 265. ▼

223. Как мы видели, схемы из функциональных элементов естественным образом реализуются в виде линейных программ. Наоборот, для деревьев решений и УБДР естественным программным представлением являются ветвящиеся программы, включающие лишь условные операторы вида **Если** v **то** Π_1 **Иначе** Π_2 и присваивания $y \leftarrow 0$ и $y \leftarrow 1$. Они соответствуют внутренним вершинам диаграмм и стокам соответственно. Здесь Π_1 и Π_2 — это снова ветвящиеся программы, а переменная y содержит результат.

Написать ветвящиеся программы, вычисляющие функции, представляемые УБДР $\mathfrak{D}_2$ на рис. 68 на стр. 271 и $\mathfrak{D}_f$ на рис. 73 на противоположной странице. ▼

Глава 15

Конечные автоматы

Краткое содержание: конечные преобразователи, работа конечного преобразователя, детерминированные конечные автоматы, распознавание автоматом языка, автоматные языки, доказательство корректности автомата, произведение автоматов, свойства замкнутости класса автоматных языков, недетерминированные конечные автоматы, детерминизация конечного автомата.

Ключевые слова: конечный преобразователь, входной алфавит, выходной алфавит, множество состояний, начальное состояние, программа, диаграмма автомата, конфигурация автомата, детерминированный конечный автомат, заключительное состояние, слово, конкатенация слов, язык, распознаваемый автоматом язык, автоматный язык, эквивалентные автоматы, произведение автоматов, недетерминированный конечный автомат, детерминизация.

§ 15.1. Конечные преобразователи

Конечные преобразователи являются математической моделью устройств, перерабатывающих дискретную входную информацию в

Рис. 74: Работа преобразователя.

режиме «реального времени», то есть в темпе её поступления. На такие устройства (рис. 74) в последовательные дискретные моменты времени $1, 2, 3, \ldots$ поступают входные сигналы $x_1, x_2, x_3, \ldots$ и в ответ на них преобразователь $\mathfrak{M}$ вырабатывает выходные сигналы $y_1, y_2, y_3, \ldots$

Конечные преобразователи характеризуются двумя следующими особенностями:

1) Отсутствие предвосхищения: последовательность выходных сигналов $y_1 \ldots y_s$, выданных к любому моменту времени t, зависит только от полученных к этому времени входных сигналов $x_1 \ldots x_t$, то есть преобразователь не может предвосхитить (узнать) будущие входы и заранее на них отреагировать. Таким образом, имеется некоторая функция выходов $\Psi(x_1 \ldots x_t)$, определяющая выдаваемую в момент t информацию по предшествующему входу.

2) Конечная память: в каждый момент t информация в преобразователе о полученном к этому моменту входе $x_1 \ldots x_t$ конечна. Это свойство удобно интерпретировать следующим образом: преобразователь имеет конечное множество состояний Q и в каждый момент находится в одном из этих состояний. При получении очередного входного сигнала состояние может измениться. Таким образом, состояние $q \in Q$, в котором находится преобразователь после получения входной последовательности $x_1 \ldots x_t$, представляет информацию об этой последовательности, используемую в дальнейшей работе преобразователя при определении следующего состояния и выхода.

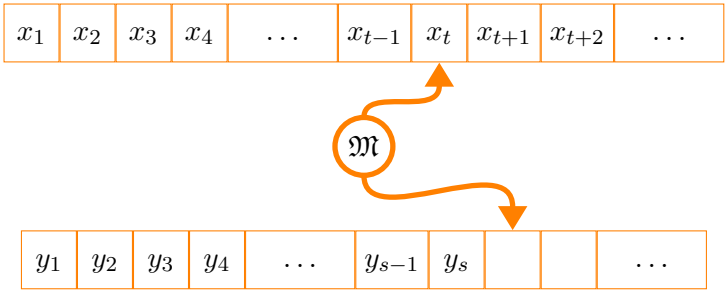


Рис. 75: Конечный преобразователь.

Удобно считать, что входные и выходные сигналы являются символами некоторых алфавитов, например, 0 и 1, если они имеют бинарный характер. Тогда конечная последовательность входных или выходных сигналов является словом в соответствующем алфавите. В этом случае преобразователь можно рассматривать как некое устройство, которое имеет две ленты (рис. 75). На одной из лент записана последовательность входных символов, на другую записываются выходные. По лентам последовательно движутся головки: на первой читающая, на второй — записывающая. Каждый раз при чтении или записи соответствующая головка сдвигается.

Наши рассуждения приводят к следующему определению конечного преобразователя.

Определение 115 (Конечный преобразователь)**.** *Мы называем конечным преобразователем* $\mathfrak{M}$ *пятёрку* $(Q, \Sigma, \Omega, P, q_0)$, *состоящую из следующих компонент:*

1) $Q = \{q_0, \ldots, q_{n-1}\}$, $n \geqslant 1$, — *конечное множество состояний;*
2) $\Sigma = \{a_1, \ldots, a_m\}$, $m \geqslant 1$, — *конечный входной алфавит;*
3) $\Omega = \{b_1, \ldots, b_r\}$, $r \geqslant 1$, — *конечный выходной алфавит;*
4) P — *программа преобразователя;*
5) $q_0 \in Q$ — *начальное состояние преобразователя.*

Программа преобразователя — это конечное множество команд, то есть слов вида «$q, a \to p, y$», где $q, p \in Q$ — состояния, $a \in \Sigma$ — символ входного алфавита, $y \in \Omega^*$ — слово в выходном алфавите, может быть, пустое. При этом для каждых $q \in Q$ и $a \in \Sigma$ в P должна быть в точности одна команда описанного вида.

Иногда программу P преобразователя называют функцией переходов.

Для конечных преобразователей существует несколько альтернативных способов описания их программы.

Один из них — табличный. Для описания преобразователя создают таблицу, каждая строка которой соответствует одному состоянию $q \in Q$, а каждый столбец — одному входному символу $a \in \Sigma$. Ячейка, находящаяся на пересечении строки q и столбца a, содержит правую часть команды: p и y.

Ещё один способ представления конечного преобразователя основан на использовании ориентированных размеченных графов.

Определение 116 (Диаграмма преобразователя)**.** *Диаграмма преобразователя $\mathfrak{M} = (Q, \Sigma, \Omega, P, q_0)$ — это размеченный ориентированный мультиграф*[1] *$\mathfrak{D}_\mathfrak{M} = (Q, E)$ такой, что:*

1) *вершины $\mathfrak{D}_\mathfrak{M}$ являются состояниями $\mathfrak{M}$;*
2) *специальным образом отмечено начальное состояние q_0;*
3) *ребро (q, p) с меткой «$a : y$» есть тогда и только тогда, когда в P есть команда вида $q, a \to p, y$.*

Таким образом, в диаграмме для каждой вершины $q \in Q$ и каждого символа $a \in \Sigma$ имеется в точности одно ребро с меткой вида $a : y$, выходящее из q.

Мы в дальнейшем, изображая диаграммы, будем отмечать начальные состояния зелёным цветом, как, например, на рис. 76 на стр. 287.

[1] Напомним, что в отличие от обычных графов в мультиграфе допускается наличие нескольких различных рёбер между одними и теми же вершинами. Эти рёбра могут иметь разные метки.

Работу конечного преобразователя $\mathfrak{M} = (Q, \Sigma, \Omega, P, q_0)$ неформально можно описать так. В начальный момент $\mathfrak{M}$ находится в состоянии q_0. Если в некоторый момент времени $\mathfrak{M}$ находится в состоянии q и на вход подаётся символ a, то в программе ищется команда вида $q, a \to p, y$ (напомним, что согласно определению она существует и единственна), $\mathfrak{M}$ переходит в состояние p, а на выход выдаётся последовательность символов слова y.

Придадим этому описанию точный характер.

Определение 117 (Конфигурация преобразователя)**.** *Для конечного преобразователя $\mathfrak{M} = (Q, \Sigma, \Omega, P, q_0)$ конфигурацией называется тройка вида (q, u, v), где $q \in Q$, $u \in \Sigma^*$, $v \in \Omega^*$.*

Напомним, что с помощью ε обозначается пустое слово. Конфигурация вида (q_0, u, ε) называется начальной, слово u в этом случае называется входным (или просто входом). Конфигурация вида (q, ε, v) для произвольного $q \in Q$ называется заключительной, а слово v в этом случае — выходным (или выходом).

Конфигурация служит средством описания ситуации в каждый момент времени. Если в какой-то момент конфигурация имеет вид (q, u, v), то это означает следующее: текущим состоянием $\mathfrak{M}$ является q, u — это ещё не прочитанная $\mathfrak{M}$ часть входа, а v — это выданная $\mathfrak{M}$ к текущему моменту часть выхода.

Определение 118 (Работа преобразователя)**.** *Пусть имеется конечный преобразователь $\mathfrak{M} = (Q, \Sigma, \Omega, P, q_0)$. Произвольная конфигурация вида (q, au, v) переходит за один шаг в конфигурацию (p, u, vy), если в программе P есть команда $q, a \to p, y$. Это записывается в виде $(q, au, v) \vdash_{\mathfrak{M}} (p, u, vy)$.*

Если σ и τ — конфигурации, то для произвольного $k \in \omega$ отношение $\sigma \vdash^k_{\mathfrak{M}} \tau$, переход за k шагов, определяется по индукции.

1) *$\sigma \vdash^0_{\mathfrak{M}} \tau$, если $\sigma = \tau$;*

2) *$\sigma \vdash^{k+1}_{\mathfrak{M}} \tau$, если $\sigma \vdash^k_{\mathfrak{M}} \rho$ и $\rho \vdash_{\mathfrak{M}} \tau$ для некоторый конфигурации ρ.*

Запись $\sigma \vdash^*_{\mathfrak{M}} \tau$ означает переход за произвольное количество шагов: $\sigma \vdash^k_{\mathfrak{M}} \tau$ для некоторого $k \in \omega$.

Итак, мы сказали, что за один шаг преобразователь читает один символ из входного слова (символ при этом из входного слова исчезает), переходит в новое состояние и записывает выходные символы в соответствии с программой.

Индукцией по k можно легко доказать такое утверждение.

Предложение 93. *Для конечного преобразователя $\mathfrak{M}$, его конфигурации σ и натурального числа k существует не более одной конфигурации τ такой, что $\sigma \vdash_{\mathfrak{M}}^{k} \tau$.*

Доказательство. Задача 228 на стр. 309. □

Это свойство называется д е т е р м и н и р о в а н н о с т ь.

Определение 119 (Вычисление)**.** *Пусть $\mathfrak{M}$ — конечный преобразователь, а w — входное слово. В ы ч и с л е н и е $\mathfrak{M}$ на w — это последовательность конфигураций (конечная или бесконечная)*

$$\sigma_0 \vdash_{\mathfrak{M}} \sigma_1 \vdash_{\mathfrak{M}} \cdots \vdash_{\mathfrak{M}} \sigma_t \vdash_{\mathfrak{M}} \sigma_{t+1} \vdash_{\mathfrak{M}} \cdots$$

такая, что σ_0 — начальная конфигурация со входом w.

Когда входные символы закончатся, записанные на выход образуют результат преобразования.

Определение 120 (Результат)**.** *Говорим, что $\mathfrak{M}$ п р е о б р а з у е т с л о в о u в с л о в о v, если вычисление $\mathfrak{M}$ на входе u заканчивается заключительной конфигурацией вида (q, ε, v), то есть с выходным словом v. Слово v тогда называется р е з у л ь т а т о м р а б о т ы $\mathfrak{M}$ н а u.*

Из детерминированности непосредственно вытекает, что существует не более одного результата для каждого входа. Следовательно, результат работы конечного преобразователя в точности один.

Бывает, что нужно знать, был ли прочитанный символ последним или будут ещё. В этом случае вводят некоторый специальный знак, которым обязательно заканчиваются все входные слова. Мы в необходимых случаях будем использовать для этой цели символ Λ.

Рассмотрим простейший пример.

Пример 65. *На вход подаётся двоичная запись некоторого натурального числа от младших разрядов к старшим. Требуется на выходе записать четверичную запись этого же числа.*

Входной алфавит преобразователя $\mathfrak{M}$ *будет состоять из трёх символов:* $\Sigma = \{0, 1, \Lambda\}$, *выходной — из четырёх:* $\Omega = \{0, 1, 2, 3\}$. *При работе каждые две подряд идущие двоичные цифры он должен преобразовать в одну четверичную. Поэтому* $\mathfrak{M}$ *должен иметь три состояния:*

1) q_0 — *начальное состояние, цифра не прочитана;*
2) q^0 — *прочитана цифра* 0;
3) q^1 — *прочитана цифра* 1.

Работать преобразователь будет так: в состоянии q_0 *читаем и запоминаем первую цифру, переходя в состояние* q^0 *или* q^1. *В каждом из этих состояний читаем вторую цифру и в зависимости от их сочетания выдаём четверичную. Если в состояниях* q^0 *или* q^1 *видим конец слова* Λ, *то считаем, что вторая цифра* 0. *Таким образом, программа* P *имеет вид:*

$$\begin{array}{lll} q_0, 0 \to q^0, \varepsilon & q^0, 0 \to q_0, 0 & q^1, 0 \to q_0, 1 \\ q_0, 1 \to q^1, \varepsilon & q^0, 1 \to q_0, 2 & q^1, 1 \to q_0, 3 \\ q_0, \Lambda \to q_0, \varepsilon & q^0, \Lambda \to q_0, \varepsilon & q^1, \Lambda \to q_0, 1 \end{array}$$

В виде таблицы она выглядит следующим образом:

$Q \setminus \Sigma$	0	1	Λ
q_0	q^0, ε	q^1, ε	q_0, ε
q^0	$q_0, 0$	$q_0, 2$	q_0, ε
q^1	$q_0, 1$	$q_0, 3$	$q_0, 1$

С помощью диаграммы эту же программу можно представить, как показано на рис. *76 на следующей странице*.

Рассмотрим, например, работу преобразователя на входе 10011:

$$(q_0, 10011\Lambda, \varepsilon) \vdash_{\mathfrak{M}} (q^1, 0011\Lambda, \varepsilon) \vdash_{\mathfrak{M}} (q_0, 011\Lambda, 1) \vdash_{\mathfrak{M}} (q^0, 11\Lambda, 1) \vdash_{\mathfrak{M}} \\ \vdash_{\mathfrak{M}} (q_0, 1\Lambda, 12) \vdash_{\mathfrak{M}} (q^1, \Lambda, 12) \vdash_{\mathfrak{M}} (q_0, \varepsilon, 121).$$

Следовательно, результатом будет слово 121. *Нетрудно проверить, что* 121 *действительно является четверичной записью числа* 25, *двоичная запись которого* 11001.

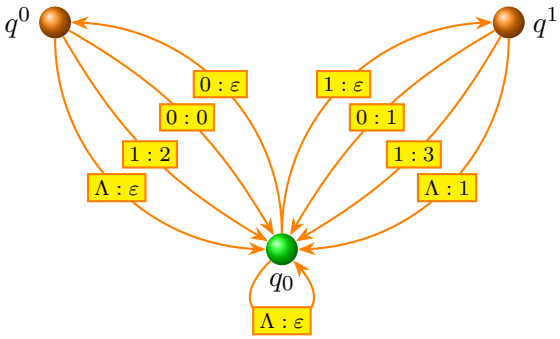


Рис. 76: Преобразование двоичной системы в четверичную

Существуют преобразователи с несколькими входами. В этом случае можно считать, что входной символ состоит из нескольких частей, которые часто называют «этажами», а сами символы — «многоэтажными». Тогда команды преобразователя приобретают вид:

$$q, \begin{bmatrix[1.5] a_1 \\ \vdots \\ a_n \end{bmatrix[1.5]} \to p, y.$$

Такая команда «срабатывает», если на первом входе видим a_1, …, на n-м входе — a_n.

Пример 66. *Рассмотрим задачу сложения двух чисел, записанных в двоичной системе. Как и в предыдущем примере мы полагаем, что числа написаны, начиная с младшего разряда к старшему, причём одинаковые разряды идут друг под другом: на верхнем этаже — разряд первого числа, на нижем — тот же разряд второго. Иными словами, вход у преобразователя такой же, как при сложении чисел «столбиком», только записан в обратном направлении.*

Теперь входной алфавит имеет вид

$$\Sigma = \left\{ \begin{bmatrix[1.5] 0 \\ 0 \end{bmatrix[1.5]}, \begin{bmatrix[1.5] 0 \\ 1 \end{bmatrix[1.5]}, \begin{bmatrix[1.5] 1 \\ 0 \end{bmatrix[1.5]}, \begin{bmatrix[1.5] 1 \\ 1 \end{bmatrix[1.5]}, \Lambda \right\},$$

выходной: $\Omega = \{0, 1\}$. *При работе нам нужно будет помнить, был ли перенос из предыдущего разряда в текущий, поэтому состояний будет два:* q_0

означает, что переноса не было, q_1 — перенос был. Начальным состоянием будет q_0, так как в начале вычисления переносов нет.

Программу преобразователя представим таблицей:

$Q \setminus \Sigma$	$\left[\!\left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]\!\right]$	$\left[\!\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]\!\right]$	$\left[\!\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]\!\right]$	$\left[\!\left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\!\right]$	Λ
q_0	$q_0, 0$	$q_0, 1$	$q_0, 1$	$q_1, 0$	q_0, ε
q_1	$q_0, 1$	$q_1, 0$	$q_1, 0$	$q_1, 1$	$q_0, 1$

Рассмотрим, как будет происходить сложение чисел 6 *и* 3. *В обратном виде их двоичная запись имеет вид* 011 *и* 110 *соответственно. Тогда входным словом будет*

$$\left[\!\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]\!\right]\left[\!\left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\!\right]\left[\!\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]\!\right]\Lambda,$$

и мы получим такую последовательность конфигураций:

$$\left(q_0, \left[\!\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]\!\right]\left[\!\left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\!\right]\left[\!\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]\!\right]\Lambda, \varepsilon\right) \vdash \left(q_0, \left[\!\left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]\!\right]\left[\!\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]\!\right]\Lambda, 1\right) \vdash \left(q_1, \left[\!\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]\!\right]\Lambda, 10\right) \vdash$$
$$\vdash (q_1, \Lambda, 100) \vdash (q_0, \varepsilon, 1001).$$

§ 15.2. Детерминированные конечные автоматы

Очень часто перед преобразователем $\mathfrak{M}$ ставится такая задача: определить, обладает ли входное слово некоторым свойством S? В качестве результата $\mathfrak{M}$ должен выдать 1 в случае положительного ответа или 0 в случае отрицательного. Рассмотрим, как можно такой преобразователь упростить.

Предложение 94. *Если единственными возможными выходными словами конечного преобразователя $\mathfrak{M} = (Q, \Sigma, \Omega, P, q_0)$ являются* 0 *и* 1, *то существует эквивалентный $\mathfrak{M}$ преобразователь $\mathfrak{M}'$, который записывает выход только при чтении заключительного входного символа Λ.*

Доказательство. Построим $\mathfrak{M}'$ из $\mathfrak{M}$, добавив к Q два новых состояния q^0 и q^1, всевозможные команды вида $q^x, a \to q^x, \varepsilon$ для $x \in \{0, 1\}$, $a \in \Sigma \setminus \{\Lambda\}$, и две команды $q^0, \Lambda \to q^0, 0$ и $q^1, \Lambda \to q^1, 1$.

Теперь заменим все команды вида $q, a \to p, x$, где $a \in \Sigma \setminus \{\Lambda\}$, $x \in \{0, 1\}$, на $q, a \to q^x, \varepsilon$.

Легко видеть, что новый преобразователь $\mathfrak{M}'$ будет работать точно так же, как $\mathfrak{M}$, за исключением случая, когда $\mathfrak{M}$ выдаёт ответ, прочитав только часть входного слова. В этой ситуации $\mathfrak{M}'$ вместо выдачи ответа запомнит его, перейдя в соответствующее состояние q^x, и выдаст ответ только при прочтении заключительного символа Λ. Таким образом, ответ $\mathfrak{M}'$ будет таким же как и $\mathfrak{M}$, но выдан он будет только при прочтении Λ. □

Теперь мы можем считать, что ответ преобразователем $\mathfrak{M}$ всегда выдаётся при чтении заключительного символа Λ. Но это означает, что перед его прочтением (то есть сразу после прочтения всех «настоящих» входных символов) $\mathfrak{M}$ уже обладал информацией о том, удовлетворяет ли входное слово свойству S. Поскольку вся информация, которой «обладает» конечный преобразователь $\mathfrak{M}$, заключена в его состоянии, то мы можем сделать такой вывод: состояние $\mathfrak{M}$ перед прочтением Λ однозначно определяет свойство S прочитанного слова. Но тогда выходные слова становятся ненужными, достаточно просто определить, какие из состояний являются «хорошими», соответствующими наличию свойства S.

Это приводит нас к определению детерминированного конечного автомата.

Определение 121 (Детерминированный конечный автомат)**.** *Детерминированный конечный автомат (ДКА)* $\mathfrak{M}$ *— это пятёрка вида* (Q, Σ, P, q_0, F)*, в которой:*

1) Q *— конечное множество состояний;*
2) Σ *— конечный входной алфавит;*
3) P *— программа автомата;*
4) $q_0 \in Q$ *— начальное состояние;*
5) $F \subseteq Q$ *— множество заключительных (принимающих) состояний.*

Программа детерминированного автомата — это конечное множество команд, то есть слов вида «$q, a \to p$», где $q, p \in Q$ — состояния, $a \in \Sigma$ — символ входного алфавита. При этом для каждых $q \in Q$ и $a \in \Sigma$ в P должна быть в точности одна команда описанного вида.

Как и преобразователи, автоматы можно задавать таблицей (отличие только в отсутствии выхода) и диаграммой. В последнем случае вводятся несколько новых терминов.

Определение 122 (Диаграмма конечного автомата)**.** *Пусть дан конечный автомат $\mathfrak{M} = (Q, \Sigma, P, q_0, F)$. Диаграмма автомата $\mathfrak{M}$ — это ориентированный мультиграф $\mathfrak{D}_\mathfrak{M} = (Q, E)$ такой, что:*

1) *вершины $\mathfrak{D}_\mathfrak{M}$ являются состояниями $\mathfrak{M}$;*
2) *отмечено начальное состояние q_0;*
3) *отмечены все заключительные состояния из F;*
4) *если в P есть команда вида $q, a \to p$, то ребро (q, p) имеет метку a, других рёбер нет.*

Если $R = (q_1, q_2, \ldots, q_{k-1}, q_k)$ — путь в $\mathfrak{D}_\mathfrak{M}$, последовательно проходящий по рёбрам $e_1, \ldots, e_{k-1}$, то метки этих рёбер составляют некоторое слово w. В этом случае говорят, что путь R несёт слово w, а слово w переводит q_1 в q_k.

Как мы уже сказали, начальное состояние мы всегда будем выделять на диаграммах автоматов зелёным цветом. Заключительные состояния мы будем выделять голубым кольцом, как это показано для состояний q_0 и q_1 в диаграмме на рис. 77 на противоположной странице. Ещё одно замечание заключается в следующем. В графах любой путь $(q_1, q_2, \ldots, q_{k-1}, q_k)$ однозначно определяет, по каким рёбрам он проходит. В мультиграфе это уже не так, так как q_1 и q_2, к примеру, могут соединяться несколькими рёбрами. Поэтому в тех случаях, когда может возникнуть такая неоднозначность, мы будем указывать метки соответствующих рёбер: $(q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \ldots \xrightarrow{a_{k-1}} q_k)$. Например, на рис. 77 на следующей странице есть четыре пути вида $(q_0, q_1, q_0, q_1, q_2)$:

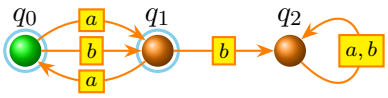


Рис. 77: Диаграмма автомата.

они несут слова $aaab$, $baab$, $aabb$ и $babb$. Чтобы указать именно путь, несущий второе слово, пишем $(q_0 \xrightarrow{b} q_1 \xrightarrow{a} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2)$.

Определение конфигураций и их переходов для конечных автоматов упрощается в связи с отсутствием выхода.

Определение 123 (Конфигурация автомата)**.** *Для конечного автомата $\mathfrak{M} = (Q, \Sigma, P, q_0, F)$ конфигурацией называется пара вида (q, u), где $q \in Q$, $u \in \Sigma^*$. Конфигурация вида (q_0, u) называется начальной, конфигурация вида (q, ε) — заключительной, если $q \in F$.*

Конфигурация вида (q, au) переходит за один шаг в конфигурацию (p, u) (записываем $(q, au) \vdash_{\mathfrak{M}} (p, u)$), если в программе P есть команда $q, a \to p$.

Определение перехода за k шагов ($\sigma \vdash^k_{\mathfrak{M}} \tau$) и за произвольное число шагов ($\sigma \vdash^*_{\mathfrak{M}} \tau$) не меняется.

Говорим, что $\mathfrak{M}$ принимает (или распознаёт) слово u, если $(q_0, u) \vdash^*_{\mathfrak{M}} (q, \varepsilon)$ для некоторого $q \in F$, то есть начальная конфигурация с входным словом u за какое-то число шагов переходит в заключительную.

Как видим, отличие этих определений от конечного преобразователя заключаются в отсутствии выходных слов и в требовании $q \in F$ для заключительных конфигураций.

Как следует из названия «детерминированный автомат», выполнено предложение аналогичное 93 на стр. 285.

Предложение 95. *Для ДКА $\mathfrak{M}$, его конфигурации σ и натурального k существует не более одной конфигурации τ такой, что $\sigma \vdash^k_{\mathfrak{M}} \tau$.*

Доказательство. Аналогично предложению 93 на стр. 285, индукцией по k. □

Напомним, что любое множество слов заданного алфавита называется языком.

Определение 124 (Автоматный язык)**.** *Язык, распознаваемый конечным автоматом $\mathfrak{M}$ (обозначаем $L(\mathfrak{M})$), — это множество принимаемых $\mathfrak{M}$ слов:*

$$L(\mathfrak{M}) = \{w \in \Sigma^* : \mathfrak{M} \text{ принимает } w\}.$$

Язык называется автоматным, если он распознаётся некоторым ДКА.

Из этого определения, в частности, следует, что $\varepsilon \in L(\mathfrak{M})$ тогда и только тогда, когда $q_0 \in F$.

Разные автоматы могут распознавать один и тот же язык.

Определение 125 (Эквивалентность автоматов)**.** *Автоматы $\mathfrak{M}$ и $\mathfrak{N}$ называются эквивалентными, если совпадают распознаваемые ими языки, то есть $L(\mathfrak{M}) = L(\mathfrak{N})$.*

Определение распознавания слова и языка можно легко перевести на язык диаграмм.

Лемма 96. *Автомат $\mathfrak{M}$ принимает слово w тогда и только тогда, когда в диаграмме $\mathfrak{D}_\mathfrak{M}$ есть несущий w путь из начального состояния q_0 в некоторое заключительное $q \in F$, то есть w переводит q_0 в заключительное состояние q.*

Доказательство. Можно провести индукцией по длине слова w (см. задачу 229 на стр. 309). □

Таким образом, язык $L(\mathfrak{M})$, распознаваемый автоматом $\mathfrak{M}$, состоит из всех слов, которые переводят в его диаграмме $\mathfrak{D}_\mathfrak{M}$ начальное состояние q_0 в заключительные состояния из F.

Наша цель теперь состоит в изучении класса автоматных языков.

Во многих случаях удаётся доказать, что язык L является автоматным, непосредственно построив распознающий его автомат. Для этого нужно постараться разбить множество всех входных слов на

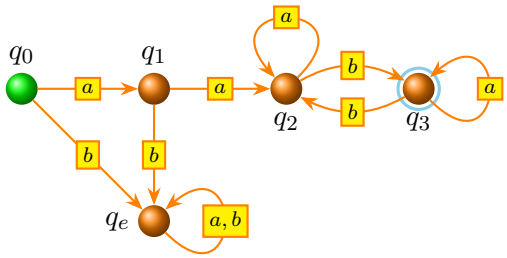


Рис. 78: Диаграмма автомата из примера 67.

конечное число классов «однородных», «эквивалентных» слов, то есть слов, получение которых на входе одинаково влияет на возможность их продолжения до слов распознаваемого языка. Затем для каждого такого класса создать состояние автомата и определить переходы между этими состояниями. Часто полезно бывает выделить одно состояние для представления «ошибочных» слов, для которых ни они сами, ни любые их продолжения не входят в язык.

Пример 67. *Пусть язык L состоит из всех слов в алфавите $\Sigma = \{a, b\}$, начинающихся с aa и содержащих нечётное количество символов b.*

Для выделения слов, начинающихся на aa, создадим начальное состояние q_0, которое первый символ a будет переводить в состояние q_1, а второй символ a будет переводить q_1 в состояние q_2. Ясно, что все слова, которые начинаются на ab, ba, bb, сами не входят в язык L и все их продолжения также ошибочны. Заведём для них «ошибочное» состояние q_e. Остальные слова естественно разбиваются на два класса: те, в которых чётное число символов b, и те, в которых число таких символов нечётно (они и принадлежат L). Так как после прочтения первых букв aa число b чётно, то для представления слов первого класса будем использовать состояние q_2, а для представления слов второго создадим ещё одно состояние q_3, которое и будет заключительным. В результате получаем автомат, диаграмма которого представлена на рисунке 78.

Рассмотрим работу этого автомата на входном слове $w = aababa$. При его чтении порождается следующая последовательность конфигураций:

$$(q_0, aaababa) \vdash_{\mathfrak{M}} (q_1, aababa) \vdash_{\mathfrak{M}} (q_2, ababa) \vdash_{\mathfrak{M}}$$
$$\vdash_{\mathfrak{M}} (q_2, baba) \vdash_{\mathfrak{M}} (q_3, aba) \vdash_{\mathfrak{M}} (q_3, ba) \vdash_{\mathfrak{M}} (q_2, a) \vdash_{\mathfrak{M}} (q_2, \varepsilon).$$

Последнее состояние этого вычисления q_2 не является заключительным, поэтому $w \notin L(\mathfrak{M})$. Если же мы рассмотрим в качестве входа слово $w_1 = wb = aaababab$, то, продолжив на один шаг приведённое выше вычисление, получим, что $(q_0, w_1) \vdash^*_{\mathfrak{M}} (q_3, \varepsilon)$. Следовательно, $w_1 \in L(\mathfrak{M})$.

Мы проверили, что на двух входах автомат $\mathfrak{M}$ работает верно. Это, конечно, не гарантирует, что он будет правильно работать и на других входных словах. Как установить, что автомат построен корректно, то есть верно работает на всех входных словах и распознаёт требуемый язык L? Типичная схема доказательства правильности конечного автомата такова:

1) определить (описать) для каждого состояния $q \in Q$ язык $L(q)$, состоящий из слов, переводящих начальное состояние q_0 в q;
2) доказать, что это определение правильное, используя индукцию по длине входного слова;
3) показать, что $L = \bigcup\limits_{q \in F} L(q)$.

Применим эту схему к доказательству правильности, построенного выше автомата $\mathfrak{M}$ из предыдущего примера.

Пример 68. *Языки, связанные с состояниями этого автомата, фактически, уже были определены при его построении. Уточним их:*

- $L(q_0) = \{\varepsilon\}$,
- $L(q_1) = \{a\}$,
- $L(q_2) = \{w : w$ *начинается на* aa *и содержит чётное число букв* $b\}$,
- $L(q_3) = L$,
- $L(q_e) = \{w : w$ *не начинается на* $aa\}$.

Правильность определения языков $L(q_0), L(q_1)$ и $L(q_e)$ следует непосредственно из построения $\mathfrak{M}$.

Для $L(q_2)$ и $L(q_3)$ используем индукцию по длине входного слова, чтобы доказать такое утверждение:

$$\begin{array}{l} \text{слово } w \text{ переводит начальное состояние } q_0 \text{ в } q_i, \text{ где} \\ i \in \{2, 3\}, \text{ тогда и только тогда, когда } w \in L(q_i). \end{array} \qquad (*)$$

Базис индукции. Самое короткое слово, переводящее q_0 в q_2 — aa, и оно принадлежит $L(q_2)$. Аналогично, самое короткое слово, переводящее q_0 в q_3 — aab, и оно принадлежит $L(q_3)$.

Индукционный шаг. Предположим теперь, что для каждого слова w длины не больше n утверждение $(*)$ доказано. Покажем, что оно будет выполнено и для всех слов длины $n+1$.

Пусть $|w| = n+1$. Тогда $w = w'x$, где $x \in \{a, b\}$. Так как $|w'| = n$, то для w' выполнено условие $(*)$. Тогда возможны четыре случая.

1) *$x = a$, $w \in L(q_2)$ и w содержит чётное число b. Тогда $w'a$ тоже содержит чётное число b и $w'a \in L(q_2)$ из-за команды $q_2, a \to q_2$.*
2) *$x = a$, $w \in L(q_3)$ и w содержит нечётное число b. В этом случае $w'a$ тоже содержит нечётное число b и $w'a \in L(q_3)$ из-за команды $q_3, a \to q_3$.*
3) *$x = b$, $w \in L(q_2)$ и w содержит чётное число b. Здесь $w'b$ содержит нечётное число b и $w'b \in L(q_3)$ из-за команды $q_2, b \to q_3$.*
4) *$x = b$, $w \in L(q_3)$ и w содержит нечётное число b. Тогда $w'b$ содержит чётное число b и $w'b \in L(q_2)$ из-за команды $q_3, b \to q_2$.*

Для завершения доказательства осталось только заметить, что единственным заключительным состоянием автомата $\mathfrak{M}$ является q_3 и поэтому $L(\mathfrak{M}) = L(q_3) = L$.

Рассмотрим одну важную конструкцию конечного автомата по двум другим, называемую произведением автоматов, которая позволяет строить новые автоматы из уже имеющихся.

Определение 126 (Произведение автоматов)**.** *Пусть даны два конечных автомата $\mathfrak{M}_1 = (Q_1, \Sigma, P_1, q_0^1, F_1)$ и $\mathfrak{M}_2 = (Q_2, \Sigma, P_2, q_0^2, F_2)$ с общим входным алфавитом Σ. Произведением автоматов $\mathfrak{M}_1$ и $\mathfrak{M}_2$ называется всякий автомат $\mathfrak{N}$ следующего вида:*

$$\mathfrak{N} = (Q_1 \times Q_2, \Sigma, P, (q_0^1, q_0^2), F),$$

в котором программа P содержит команду $(q_1, q_2), a \to (p_1, p_2)$, если $q_1, a \to p_1 \in P_1$ и $q_2, a \to p_2 \in P_2$.

Поскольку мы не наложили никаких условий на множество заключительных состояний F автомата $\mathfrak{N}$, то произведение не является

однозначно определённой операцией, в зависимости от F будут получаться разные автоматы.

Главное свойство этой конструкции определяет следующая лемма.

Лемма 97. *Если $\mathfrak{N}$ — произведение $\mathfrak{M}_1$ и $\mathfrak{M}_2$, то для любых двух состояний (q_1, q_2) и (p_1, p_2) автомата $\mathfrak{N}$ и любого входного слова w выполнено следующее:*

> *w переводит (q_1, q_2) в (p_1, p_2) в автомате $\mathfrak{N}$ в том и только том случае, когда w одновременно переводит q_1 в p_1 в автомате $\mathfrak{M}_1$ и q_2 в p_2 в автомате $\mathfrak{M}_2$.*

Другими словами, первая компонента состояний $\mathfrak{M}$ моделирует работу $\mathfrak{M}_1$, а вторая — $\mathfrak{M}_2$ на общем входе.

Доказательство. Проводится индукцией по длине слова w (задача 232 на стр. 309). □

Указанная конструкция даёт один из способов доказательства замкнутости множества автоматных языков относительно теоретико-множественных операций: объединения, пересечения, разности и дополнения. Для каждой из этих операций строится произведение автоматов и определяется соответствующее множество заключительных состояний F этого произведения.

Теорема 98. *Пусть $\mathfrak{N}$ — произведение автоматов $\mathfrak{M}_1$ и $\mathfrak{M}_2$ с входным алфавитом Σ, F, F_1 и F_2 — множества заключительных состояний этих автоматов соответственно. Тогда*

1) *если $F = \{(q, p) : q \in F_1$ или $p \in F_2\}$, то $L(\mathfrak{N}) = L(\mathfrak{M}_1) \cup L(\mathfrak{M}_2)$;*
2) *если $F = \{(q, p) : q \in F_1$ и $p \in F_2\}$, то $L(\mathfrak{N}) = L(\mathfrak{M}_1) \cap L(\mathfrak{M}_2)$;*
3) *если $F = \{(q, p) : q \in F_1$ и $p \notin F_2\}$, то $L(\mathfrak{N}) = L(\mathfrak{M}_1) \setminus L(\mathfrak{M}_2)$.*

Доказательство. Следует из предыдущей леммы. □

Следствие 99. *Класс автоматных языков в алфавите Σ замкнут относительно теоретико-множественных операций объединения, пересечения, разности и дополнения.*

Доказательство. Замкнутость для первых трёх операций непосредственно получается из теоремы. Дополнение языка L является разностью $\Sigma^* \setminus L$. Язык Σ^* является автоматным, поэтому если L автоматен, то и его дополнение $\Sigma^* \setminus L$ автоматно. □

Указанное доказательство замкнутости не единственно, в следующей главе будет приведено другое.

§ 15.3. Недетерминированные конечные автоматы

Недетерминированные конечные автоматы, рассматриваемые в этом параграфе, являются обобщениями детерминированных. Они при чтении очередного символа на входе могут «самостоятельно» выбрать в качестве следующего одно из нескольких состояний, а кроме того, могут изменить состояние без чтения входного символа. В ряде случаев такая модель автоматов является более удобной: мы будем именно ими пользоваться в следующей части при изучении регулярных выражений.

Определение 127 (Недетерминированный конечный автомат)**.** *Недетерминированный конечный автомат (НКА) — это пятёрка вида (Q, Σ, P, q_0, F), в которой:*

1) *Q — конечное множество состояний;*
2) *Σ — конечный входной алфавит;*
3) *P — программа автомата;*
4) *$q_0 \in Q$ — начальное состояние;*
5) *$F \subseteq Q$ — множество заключительных (принимающих) состояний.*

Программа недетерминированного автомата — это конечное множество команд, то есть слов вида «$q, a \to p$» или «$q, \varepsilon \to p$», где $q, p \in Q$ — состояния, $a \in \Sigma$ — символ входного алфавита.

Команды вида «$q, \varepsilon \to p$» называют пустыми или ε-переходами.

Как видим, отличие от детерминированного автомата — это отсутствие ограничений на количество команд с заданной левой частью и допустимость команд вида «$q, \varepsilon \to p$».

Как и для ДКА, программу можно представить с помощью множества команд, в виде таблицы или диаграммой.

В отличие от ДКА в таблице будет новый столбец — для ε-переходов. Также в таблице ячейка, соответствующая состоянию q и символу a, может содержать несколько состояний $p_1, \ldots, p_k$, соответствующих нескольким командам вида $q, a \to p_i$ для $i = 1, \ldots, k$. Ячейка может быть и пустой, если команд вида $q, a \to p$ в программе P нет.

Диаграммы НКА в отличие от ДКА могут иметь несколько рёбер с одинаковой меткой, исходящих из одной и той же вершины.

Пример 69. *Рассмотрим НКА $\mathfrak{N} = (Q, \Sigma, P, q_0, F)$, в котором $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$, $F = \{q_3\}$, а программа P состоит из следующих команд:*

$$\begin{array}{ll} q_0, a \to q_0; & q_2, a \to q_3; \\ q_0, a \to q_1; & q_2, \varepsilon \to q_0; \\ q_0, b \to q_0; & q_3, \varepsilon \to q_1; \\ q_1, \varepsilon \to q_4; & q_4, b \to q_2. \end{array}$$

Представление программы P в виде таблицы и диаграмма автомата $\mathfrak{D}_{\mathfrak{N}}$ показаны на рис. 79 на следующей странице.

Конкатенация пустого слова ε с любым другим словом w даёт снова слово w, поэтому в определении слова, которое несёт путь, метки ε не учитываются. Например, путь $(q_0 \xrightarrow{a} q_1 \xrightarrow{\varepsilon} q_4 \xrightarrow{b} q_2 \xrightarrow{\varepsilon} q_0 \xrightarrow{b} q_0)$ в диаграмме на рис. 79 на противоположной странице несёт слово abb. Следовательно, для недетерминированных автоматов длина слова и несущего его пути уже не обязательно совпадают.

Понятие конфигурации для недетерминированных автоматов такое же как для детерминированных, а в определении перехода за один шаг требуется добавить случай ε-переходов.

$Q \setminus \Sigma$	a	b	ε
q_0	q_0, q_1	q_0	
q_1			q_4
q_2	q_3		q_0
q_3			q_1
q_4		q_2	

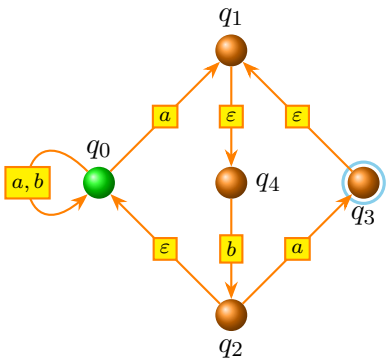


Рис. 79: Программа в виде таблицы и диаграмма автомата $\mathfrak{N}$ из примера 69.

Определение 128 (Переход для недетерминированного автомата)**.** *Для НКА $\mathfrak{N} = (Q, \Sigma, P, q_0, F)$ конфигурация вида (q, au) переходит за один шаг в конфигурацию (p, u) (записываем $(q, au) \vdash_{\mathfrak{N}} (p, u)$), если в программе P есть команда $q, a \to p$. Также конфигурация вида (q, u) переходит за один шаг в конфигурацию (p, u), если в программе P есть команда $q, \varepsilon \to p$.*

Таким образом, при «выполнении» команды $q, \varepsilon \to p$ символ входного слова не читается, а изменяется только состояние автомата. Определения перехода за k шагов ($\sigma \vdash^k_{\mathfrak{M}} \tau$) и за произвольное число шагов ($\sigma \vdash^*_{\mathfrak{M}} \tau$) остаются теми же самыми.

Недетерминированные автоматы в отличие от детерминированных за один шаг могут перейти в несколько разных конфигураций. Скажем, автомат $\mathfrak{N}$ из примера 69 на предыдущей странице из конфигурации (q_2, ab) может перейти в конфигурацию (q_0, ab) (с помощью команды $q_2, \varepsilon \to q_0$), а может — в конфигурацию (q_3, b) (с помощью команды $q_2, a \to q_3$). Возможна и ситуация, когда автомат не может перейти ни в какую конфигурацию, даже если входное слово непусто: для того же автомата конфигурация (q_2, b) не имеет следующей, так как команды вида $q_2, b \to \ldots$ в его программе нет.

Не меняются и другие определения. Автомат $\mathfrak{N} = (Q, \Sigma, P, q_0, F)$ принимает слово u, если существует $q \in F$, для которого $(q_0, u) \vdash^*_{\mathfrak{N}} \vdash^*_{\mathfrak{N}} (q, \varepsilon)$. Следует иметь в виду, что для НКА могут существовать много разных состояний q, для которых выполняется условие $(q_0, u) \vdash^*_{\mathfrak{N}} \vdash^*_{\mathfrak{N}} (q, \varepsilon)$. Автомат принимает слово u, если среди этих состояний есть хотя бы одно заключительное. Для диаграммы это означает, что есть хотя бы один несущий u путь из начального состояния в какое-то заключительное.

Таким же остаётся и определение языка, который распознаёт автомат.

Пример 70. *Рассмотрим работу автомата $\mathfrak{N}$ из примера 69 на стр. 298 на слове aababa:*

$$(q_0, aababa) \vdash_{\mathfrak{N}} (q_0, ababa) \vdash_{\mathfrak{N}} (q_1, baba) \vdash_{\mathfrak{N}} (q_4, baba) \vdash_{\mathfrak{N}} (q_2, aba) \vdash_{\mathfrak{N}} \\ \vdash_{\mathfrak{N}} (q_0, aba) \vdash_{\mathfrak{N}} (q_1, ba) \vdash_{\mathfrak{N}} (q_4, ba) \vdash_{\mathfrak{N}} (q_2, a) \vdash_{\mathfrak{N}} (q_3, \varepsilon).$$

Эта последовательность соответствует пути

$$(q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_1 \xrightarrow{\varepsilon} q_4 \xrightarrow{b} q_2 \xrightarrow{\varepsilon} q_0 \xrightarrow{a} q_1 \xrightarrow{\varepsilon} q_4 \xrightarrow{b} q_2 \xrightarrow{a} q_3),$$

несущему слово aababa. Так как q_3 — это заключительное состояние автомата $\mathfrak{N}$, то $aababa \in L(\mathfrak{N})$.

Заметим, что у автомата $\mathfrak{N}$ имеются и другие способы работы на этом слове. Например, следующий путь тоже несёт слово aababa и ведёт к заключительному состоянию, но иначе:

$$(q_0 \xrightarrow{a} q_0 \xrightarrow{a} q_1 \xrightarrow{\varepsilon} q_4 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \xrightarrow{\varepsilon} q_1 \xrightarrow{\varepsilon} q_4 \xrightarrow{b} q_2 \xrightarrow{a} q_3).$$

Есть пути, не ведущие к заключительному состоянию. Например, автомат $\mathfrak{N}$ может после чтения каждого символа оставаться в состоянии q_0. Имеются и пути, которые заводят в «тупик», например, если начать путь так: $q_0 \xrightarrow{a} q_1 \xrightarrow{\varepsilon} q_4$, то следующей конфигурации нет, так как в программе отсутствует команда вида $q_4, a \to \ldots$.

Но чтобы слово допускалось, достаточно существования хотя бы одного «хорошего» способа, который мы и привели.

§ 15.4. Детерминизация конечных автоматов

Очевидно, что детерминированные конечные автоматы являются частными случаями недетерминированных. Естественно спросить, распознают ли недетерминированные конечные автоматы бóльший класс языков, чем детерминированные? Следующая теорема показывает, что классы языков, распознаваемых НКА и ДКА, совпадают. Следовательно, для конечных автоматов недетерминированность несущественна, она всегда устранима.

Теорема 100 (О детерминизации НКА)**.** *Для каждого НКА* $\mathfrak{N}$ *можно эффективно построить такой ДКА* $\mathfrak{M}$*, что* $L(\mathfrak{M}) = L(\mathfrak{N})$.

Доказательство. Пусть $\mathfrak{N} = (Q, \Sigma, P, q_0, F)$ — НКА. Процедура построения по нему эквивалентного ДКА состоит из двух этапов: на первом по $\mathfrak{N}$ строится эквивалентный ему НКА $\mathfrak{N}_1$, в программе которого отсутствуют ε-переходы, а на втором этапе по $\mathfrak{N}_1$ строится эквивалентный ДКА $\mathfrak{M}$.

Этап 1, устранение пустых переходов. Рассмотрим поддиаграмму автомата $\mathfrak{N}$, в которой оставлены лишь ε-переходы: $\mathfrak{D}_\varepsilon = (Q, E_\varepsilon)$, где

$$E_\varepsilon = \{(q, p) : P \text{ содержит команду } q, \varepsilon \to p\}.$$

Пусть $\mathfrak{D}^*_\varepsilon = (Q, E^*_\varepsilon)$ — это граф достижимости (транзитивного замыкания) для $\mathfrak{D}_\varepsilon$. Тогда

$$E^*_\varepsilon = \{(q, p) : \text{в } \mathfrak{D}_\varepsilon \text{ имеется путь из } q \text{ в } p\}.$$

Определим НКА $\mathfrak{N}_1 = (Q_1, \Sigma, P_1, q_0, F_1)$ следующим образом. Множество состояний:

$$Q_1 = \{q_0\} \cup \{q \in Q : \text{существуют } p \in Q,\ a \in \Sigma \text{ и команда } p, a \to q \in P\},$$

то есть кроме начального остаются лишь те состояния, в которые входят «непустые» рёбра. Заключительные состояния:

$$F_1 = \{q \in Q_1 : \text{существует } p \in F \text{ и } (q, p) \in E^*_\varepsilon\},$$

то есть к заключительным состояниям $\mathfrak{N}$ добавляются состояния, из которых можно было попасть в заключительные по путям из ε-рёбер. Для каждой пары $q' \in Q_1$ и $a \in \Sigma$ строим множество команд:

$$P_{qa} = \{q, a \to p : \text{существует } r \in Q,\ (q, r) \in E_\varepsilon^* \text{ и } r, a \to p \in P\},$$

то есть $\mathfrak{N}_1$ есть a-переход из q в p, если в $\mathfrak{N}$ можно было по ε-переходам попасть из q в какое-то состояние r, а из него по символу a — в p. Заметим, что это определение автоматически гарантирует, что $p \in Q_1$. Программа P_1 автомата $\mathfrak{N}_1$ получается объединением всех P_{qa}:

$$P_1 = \bigcup_{q \in Q_1} \bigcup_{a \in \Sigma} P_{qa}.$$

Из способа построения $\mathfrak{N}_1$ непосредственно вытекает, что в НКА $\mathfrak{N}_1$ нет ε-переходов. Для завершения первого этапа осталось установить эквивалентность $\mathfrak{N}$ и $\mathfrak{N}_1$. Доказательство этого факта вынесено в отдельную лемму 101 на следующей странице.

Э т а п 2, детерминизация. Идея детерминизации состоит в том, что состояниями ДКА объявляются всевозможные подмножества состояний НКА. Тогда для каждого такого подмножества T и входного символа a однозначно определено множество состояний S, в которые НКА может попасть из состояний T при чтении a.

Определим по НКА $\mathfrak{N}_1 = (Q_1, \Sigma, P_1, q_0, F_1)$, который мы построили на первом этапе, искомый ДКА $\mathfrak{M} = (Q^{\mathfrak{M}}, \Sigma, P^{\mathfrak{M}}, q_0^{\mathfrak{M}}, F^{\mathfrak{M}})$ следующим образом:

$$\begin{aligned}
&Q^{\mathfrak{M}} = \mathsf{P}(Q_1); \quad q_0^{\mathfrak{M}} = \{q_0\};\\
&F^{\mathfrak{M}} = \{T \in Q^{\mathfrak{M}} : T \cap F_1 \neq \varnothing\};\\
&P^{\mathfrak{M}} = \big\{T, a \to S : T \in Q^{\mathfrak{M}} \text{ и}\\
&\qquad\qquad S = \{q \in Q_1 : \text{существуют } p \in T \text{ и } p, a \to q \in P_1\}\big\}.
\end{aligned}$$

Поскольку в определении $P^{\mathfrak{M}}$ множество S однозначно определено по T, то построенный автомат $\mathfrak{M}$ — детерминированный. Чтобы

продемонстрировать эквивалентность $\mathfrak{N}_1$ и $\mathfrak{M}$ мы докажем следующее вспомогательное утверждение (лемма 102 на следующей странице):

$$(T, w) \vdash^*_{\mathfrak{M}} (S, \varepsilon) \iff$$
$$\iff S = \{p \in Q_1 : (q, w) \vdash^*_{\mathfrak{N}_1} (p, \varepsilon) \text{ для некоторого } q \in T\}$$

для любых $T, S \in Q^{\mathfrak{M}}$ и слова $w \in \Sigma^*$.

Для завершения доказательства теоремы покажем с помощью леммы 102 на следующей странице, что $L(\mathfrak{M}) = L(\mathfrak{N}_1)$. Действительно, если слово w переводит начальное состояние q_0 в некоторое заключительное $q_f \in F_1$ в автомате $\mathfrak{N}_1$, то, положив в лемме $T = \{q_0\}$, получим, что $q_f \in S$ для состояния S, такого, что $(T, w) \vdash^*_{\mathfrak{M}} (S, \varepsilon)$. Но тогда $S \in F^{\mathfrak{M}}$ и $w \in L(\mathfrak{M})$.

Обратно, если $w \in L(\mathfrak{M})$, то для некоторого $S \in F^{\mathfrak{M}}$ имеем $(\{q_0\}, w) \vdash^*_{\mathfrak{M}} (S, \varepsilon)$. Тогда в S имеется некоторое состояние $q_f \in F_1$ и по лемме 102 на следующей странице для автомата $\mathfrak{N}_1$ будет выполнено $(q_0, w) \vdash^*_{\mathfrak{N}_1} (q_f, \varepsilon)$, то есть $w \in L(\mathfrak{N}_1)$. □

Лемма 101. $L(\mathfrak{N}_1) = L(\mathfrak{N})$.

Доказательство. Пусть $w = a_1 a_2 \dots a_t$ — произвольное входное слово, $a_i \in \Sigma$ для $i = 1, \dots, t$.

Предположим, что $w \in L(\mathfrak{N}_1)$. Это означает, что в диаграмме $\mathfrak{D}_{\mathfrak{N}_1}$ имеется путь $R = (q_0 = q_{i_0}, \dots, q_{i_t})$, несущий слово w, то есть ребро $(q_{i_{j-1}}, q_{i_j})$ помечено символом a_j, а кроме того, состояние $q_{i_t} \in F_1$ является заключительным в $\mathfrak{N}_1$. Из определения P_1 непосредственно следует, что ребро $(q_{i_{j-1}}, q_{i_j})$ с меткой a_j присутствует тогда и только тогда, когда в диаграмме $\mathfrak{D}_{\mathfrak{N}}$ имеется путь R_j из $q_{i_{j-1}}$ в q_{i_j}, в котором последнее ребро помечено символом a_j, а все предыдущие — пустые. Объединив все пути R_j в один, мы получим в диаграмме $\mathfrak{D}_{\mathfrak{M}}$ путь R' из q_0 в q_{i_t}, несущий слово w. Так как $q_{i_t} \in F_1$, то в диаграмме $\mathfrak{D}_{\mathfrak{N}}$ из q_{i_t} достижимо некоторое состояние $q_f \in F$ по ε-переходам, то есть имеется путь R'' из q_{i_t} в q_f, несущий пустое слово. Но тогда при объединении R' и R'' мы получим путь из q_0 в q_f, несущий слово w. Это означает, что автомат $\mathfrak{N}$ принимает w.

Обратно, пусть $w \in L(\mathfrak{N})$. Тогда в $\mathfrak{D}_{\mathfrak{N}}$ есть несущий слово w путь R из q_0 в какое-то заключительное состояние q_f. Выделим в нём все непустые переходы и пустые промежутки между ними:

$$(\underbrace{q_0, \ldots, q_{i_1-1}}_{R_1}, \underbrace{q_{i_1}, \ldots, q_{i_2-1}}_{R_2}, \ldots, \underbrace{q_{i_{t-1}}, \ldots, q_{i_t-1}}_{R_t}, \underbrace{q_{i_t}, \ldots, q_f}_{R'}).$$

Здесь каждое ребро (q_{i_j-1}, q_{i_j}) помечено символом a_j, $j = 1, \ldots, t$, а внутри фрагментов $R_1, R_2, \ldots, R_t, R'$ встречаются только пустые переходы. Тогда по определению программы P_1 получаем, что она будет содержать команды $q_0, a_1 \to q_{i_1}$, $q_{i_1}, a_2 \to q_{i_2}$, $\ldots$, $q_{i_{t-1}}, a_t \to q_{i_t}$. Поскольку в диаграмме $\mathfrak{D}_{\mathfrak{N}}$ из q_{i_t} имеется пустой путь R' в заключительное состояние $q_f \in F$, то $q_{i_t} \in F_1$. Другими словами, при чтении слова w автомат $\mathfrak{N}_1$ из состояния q_0 перейдёт в $q_{i_t} \in F_1$, то есть примет слово. □

Лемма 102. *Для всех состояний $T, S \in Q^{\mathfrak{M}}$ и всех слов $w \in \Sigma^*$*

$$(T, w) \vdash^*_{\mathfrak{M}} (S, \varepsilon) \iff$$
$$\iff S = \{p \in Q_1 : (q, w) \vdash^*_{\mathfrak{N}_1} (p, \varepsilon) \text{ для некоторого } q \in T\}.$$

Доказательство. Применим индукцию по длине слова w.

Базис индукции. Пусть $|w| = 0$, тогда $w = \varepsilon$, $T = S$ и утверждение выполнено.

Шаг индукции. Предположим, что $|w| = k + 1$ и лемма справедлива для всех слов длины k и меньше. Выделим в w первый символ: $w = aw'$. Пусть T' — это такое состояние, что $(T, a) \vdash_{\mathfrak{M}} (T', \varepsilon)$. Из определения $P^{\mathfrak{M}}$ получаем

$$T' = \{r : \text{существует такое } q \in T, \text{ что } q, a \to r \in P_1\}. \qquad (23)$$

Если $(T, w) \vdash^*_{\mathfrak{M}} (S, \varepsilon)$, то $(T', w') \vdash^*_{\mathfrak{M}} (S, \varepsilon)$. Так как $|w'| = k$, то по индукционному предположению это эквивалентно следующему:

$$S = \{p \in Q_1 : (r, w') \vdash^*_{\mathfrak{N}_1} (p, \varepsilon) \text{ для некоторого } r \in T'\}. \qquad (24)$$

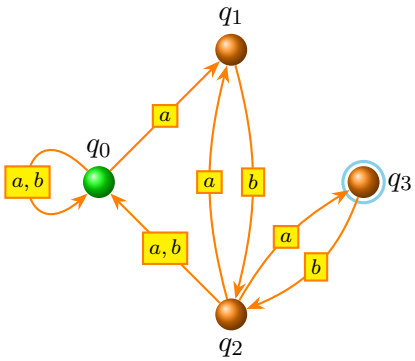


Рис. 80: Диаграмма автомата $\mathfrak{N}_1$ из примера 71.

Объединив (23) и (24), получаем:

$$S = \{p : \text{существует такое } q \in T, \text{ что } (q, w) \vdash^*_{\mathfrak{N}_1} (p, \varepsilon)\}. \tag{25}$$

Если теперь, наоборот, выполнено (25), то, учитывая (23), получаем (24), по индукционному предположению $(T', w') \vdash^*_{\mathfrak{M}} (S, \varepsilon)$, что вместе с $(T, a) \vdash_{\mathfrak{M}} (T', \varepsilon)$ даёт $(T, w) \vdash^*_{\mathfrak{M}} (S, \varepsilon)$. □

Пример 71. *Применим процедуру из теоремы о детерминизации к НКА* $\mathfrak{N}$ *из примера 69 на стр. 298.*

На первом этапе получаем $E^*_\varepsilon = \{(q_2, q_0), (q_1, q_4), (q_3, q_1), (q_3, q_4)\}$ *и НКА* $\mathfrak{N}_1$ *без пустых переходов, представленный на рис. 80.*

Заметим, что состояние q_4 *исчезло, так как в автомате* $\mathfrak{N}$ *в него можно было попасть только по* ε*-переходу. Между оставшимися состояниями сохраняются переходы по символам* a, b *и появляются новые, индуцируемые* ε*-путями. Например, переход* $q_3, b \to q_2$ *соответствует пути* (q_3, q_1, q_4, q_2) *в автомате* $\mathfrak{N}$*.*

На втором этапе детерминизируем $\mathfrak{N}_1$*. ДКА* $\mathfrak{M}$ *будет иметь 16 состояний:*

$$Q^{\mathfrak{M}} = \mathsf{P}(Q_1) = \big\{\varnothing, \{q_0\}, \{q_1\}, \{q_2\}, \{q_3\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_0, q_3\}, \{q_1, q_2\}, \{q_1, q_3\}, \{q_2, q_3\}, \{q_0, q_1, q_2\}, \{q_0, q_1, q_3\}, \{q_0, q_2, q_3\}, \{q_1, q_2, q_3\}, \{q_0, q_1, q_2, q_3\}\big\}.$$

$Q^{\mathfrak{M}} \setminus \Sigma$	a	b
$\varnothing$	$\varnothing$	$\varnothing$
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\varnothing$	$\{q_2\}$
$\{q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_3\}$	$\varnothing$	$\{q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0\}$
$\{q_0, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

$Q^{\mathfrak{M}} \setminus \Sigma$	a	b
$\{q_1, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_2\}$
$\{q_1, q_3\}$	$\varnothing$	$\{q_2\}$
$\{q_2, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_2\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_2\}$
$\{q_1, q_2, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_2\}$
$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_1, q_3\}$	$\{q_0, q_2\}$

Рис. 81: Команды автомата $\mathfrak{M}$ из примера 71

В множество заключительных состояний войдут состояния, содержащие заключительное состояние q_3 автомата $\mathfrak{N}_1$:

$$F^{\mathfrak{M}} = \{\{q_3\}, \{q_0, q_3\}, \{q_1, q_3\}, \{q_2, q_3\}, \{q_0, q_1, q_3\}, \{q_0, q_2, q_3\}, \{q_1, q_2, q_3\}, \{q_0, q_1, q_2, q_3\}\}.$$

Команды автомата $\mathfrak{M}$ определены таблицей (рис. 81).

На самом деле нас интересуют лишь те состояния, в которые можно попасть из начального состояния $\{q_0\}$. Несложный анализ показывает, что кроме $\{q_0\}$ таких состояний только три: $\{q_0, q_1\}$, $\{q_0, q_2\}$ и $\{q_0, q_1, q_3\}$. Остальные состояния не достижимы из $\{q_0\}$ и, следовательно, не влияют на работу автомата $\mathfrak{M}$. Их можно отбросить. Таким образом, в диаграмме автомата остаются четыре состояния, показанные на рис. 82 на противоположной странице.

Замечание 16. Доказанная теорема показывает, что с точки зрения выразительных возможностей недетерминированность автомата значения не имеет. Однако она может быть очень существенна в «количественном» смысле: сколько требуется состояний, чтобы построить автомат, распознающий данный язык?

Поскольку в качестве множества состояний ДКА берётся $\mathsf{P}(Q)$, то, учитывая равенство $|\mathsf{P}(A)| = 2^{|A|}$, можно сделать вывод, что количество состояний при детерминизации может сильно увеличиваться. В рассмотренном примере у построенного ДКА $\mathfrak{M}$ оказалось столько же состояний,

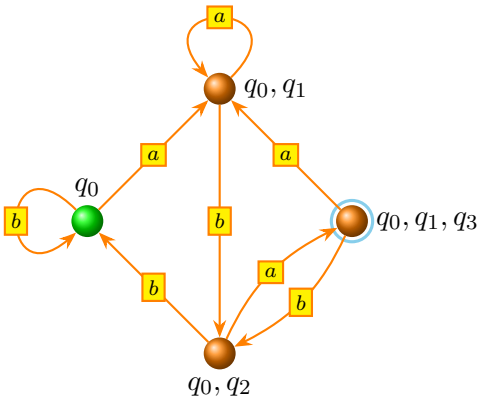


Рис. 82: Диаграмма автомата $\mathfrak{M}$ из примера 71.

как и у исходного НКА $\mathfrak{N}$. К сожалению, это не всегда так. Существуют примеры НКА с n состояниями, для которых эквивалентные ДКА содержат не менее 2^n состояний.

Из-за сильно возрастающего размера автомата вместо построения детерминированного автомата $\mathfrak{M}$ на практике прибегают к другому способу: после удаления переходов по пустым словам, то есть построения $\mathfrak{N}_1$, моделируют работу детерминированного автомата $\mathfrak{M}$ без явного его построения. В самом деле, команды автомата $\mathfrak{M}$ для каждого состояния T можно легко получить из команд $\mathfrak{N}_1$. Следовательно, можно реализовать алгоритм работы $\mathfrak{M}$, показанный на рис. 83 на следующей странице.

Нетрудно заметить, что после выполнения k итераций внешнего цикла (строки 5–13) T будет содержать в точности состояние детерминизированного автомата $\mathfrak{M}$, в которое он попадёт после прочтения первых k символов $a_1 \dots a_k$ входного слова w. Следовательно, по завершении этого цикла мы получим состояние автомата $\mathfrak{M}$, в котором тот окажется после прочтения всего входного слова w. Единственное, что остаётся, — это проверить, является ли полученное состояние T заключительным.

```
1: Алгоритм НедАвт(𝔑, w)          # Моделирование работы НКА
2:     Вход: 𝔑 = (Q, Σ, P, q_0, F) — НКА без ε-переходов
3:     Вход: w ∈ Σ* — входное слово w = a_1 ... a_k
4:     T ← {q_0}
5:     Для всех i ← 1, ..., k выполнять
6:         S ← ∅                  # Построение следующего состояния
7:         Для всех q ∈ T выполнять
8:             Если q, a_i → p ∈ P то
9:                 S ← S ∪ {p}
10:            Конец Если
11:        Конец Для
12:        T ← S
13:    Конец Для
14:    Если T ∩ F ≠ ∅ то
15:        Вернуть «да»
16:    Иначе
17:        Вернуть «нет»
18:    Конец Если
19: Конец Алгоритм
```

Рис. 83: Алгоритм моделирования НКА.

Задачи

224. Автомат по продаже кофе имеет щель для получения монет, кнопку, нажатие которой после уплаты достаточной суммы приводит к получению кофе, и накопитель, через который он выдаёт сдачу покупателю. Автомат принимает монеты достоинством в 1, 2 и 5 рублей. Чашка кофе стоит 8 руб. Пока полученная сумма недостаточна, горит красная лампочка. Если сумма, полученная автоматом, становится больше или равна 8 рублям, то зажигается зелёная лампочка и после нажатия кнопки автомат наливает кофе и, если требуется, выдаёт сдачу наименьшим числом монет. Если автомат получает монету, когда горит зелёная лампочка, то он немедленно её возвращает.

Построить конечный преобразователь, моделирующий работу этого автомата. Определить входной и выходной алфавиты и построить его программу. ▼

225. Электронные часы имеют табло с указанием часов, минут и секунд и две управляющие кнопки. Одна кнопка переводит часы из нормального режима в

режим настройки времени — вначале в настройку часов, затем — минут, затем — секунд, а затем возвращает в нормальный режим. Другая кнопка в нормальном режиме ничего не меняет, а в режиме настройки нажатие на неё увеличивает на единицу число настраиваемых часов, минут или секунд.

Построить конечный преобразователь, моделирующий работу часов. На вход он принимает сигналы нажатия от двух кнопок, а на выходе выдаёт сигналы изменения режима и увеличения соответствующего числа. ▼

226. Построить конечный преобразователь, умножающий число в двоичной записи на 3. Цифры записаны в порядке возрастания разрядов. ▼

227. Построить конечный преобразователь, нацело делящий число в десятичной записи на 3. Цифры записаны в порядке убывания разрядов. ▼

228. Доказать предложение 93 на стр. 285. ▼

229. Индукцией по длине слова w доказать такое утверждение: $(q, wx) \vdash^*_{\mathfrak{M}} (p, x)$ тогда и только тогда, когда в $\mathfrak{D}_{\mathfrak{M}}$ есть путь из q в p, несущий w. Вывести из него лемму 96 на стр. 292. ▼

230. Построить детерминированные конечные автоматы, которые распознают следующие языки в алфавите $\Sigma = \{a, b\}$:

(а) $L = \{w : \text{длина } w \text{ делится на } 5\}$;

(б) $L = \{w : w \text{ не содержит подслов «}aab\text{» и «}bba\text{»}\}$;

(в) $L = \{w : w \text{ содержит чётное число букв } a \text{ и нечётное число букв } b\}$;

(г) $L = \{w : \text{число букв } a \text{ в слове } w \text{ делится на 3, а число букв } b \text{ на 2}\}$. ▼

231. Выше в примере 66 на стр. 287 был построен преобразователь, выполняющий сложение двух двоичных чисел. Построить детерминированный конечный автомат, который проверяет правильность сложения. На вход поступают «трёхэтажные» символы, в которых на верхнем этаже записан разряд первого слагаемого, на среднем — второго, на нижнем — предполагаемой суммы. Автомат должен принять слово, если записанный на нижнем этаже результат сложения верен. ▼

232. Доказать лемму 97 на стр. 296. ▼

233. Доказать, что приведённый на рис. 82 на стр. 307 автомат $\mathfrak{M}$ распознаёт язык, состоящий из всех слов, заканчивающихся на «aba». ▼

234. Используя процедуру детерминизации недетерминированных автоматов из теоремы 100 на стр. 301, построить ДКА, эквивалентный заданному НКА $\mathfrak{N}$:

(а) $\mathfrak{N} = (\{q_0, q_1, q_2\}, \{a, b\}, P, q_0, \{q_2\})$ с программой $P = \{q_0, a \to q_1; q_0, \varepsilon \to q_1; q_1, b \to q_2; q_1, \varepsilon \to q_2; q_2, b \to q_2\}$;

(б) $\mathfrak{N} = (\{q_0, q_1, q_2\}, \{a, b\}, P, q_0, \{q_2\})$ с программой $P = \{q_0, a \to q_1; q_0, a \to q_2; q_1, b \to q_2; q_1, \varepsilon \to q_2; q_2, b \to q_0\}$. ▼

Глава 16

Регулярные выражения

Краткое содержание: операции конкатенации и итерации языков, регулярные выражения и языки, примеры регулярных выражений и языков, построение конечного автомата по регулярному выражению, построение регулярного выражения по конечному автомату.

Ключевые слова: конкатенация языков, итерация языка, регулярное выражение, регулярный язык.

§ 16.1. Регулярные выражения и языки

Регулярные выражения являются достаточно удобным средством для построения «алгебраических» описаний языков. Фактически они являются формулами, которые строятся из элементарных выражений $\varnothing$, ε и символов алфавита Σ с помощью операций объединения $(+)$, конкатенации $(\&)$ и итерации $(*)$. Каждое такое выражение r задаёт некоторый язык $L(r)$ над алфавитом Σ. Смысл операции объединения языков мы знаем. Определим операции конкатенации и итерации (иногда последнюю называют замыканием Клини в честь С. Клини).

Определение 129 (Конкатенация языков)**.** *Пусть L_1 и L_2 — языки в алфавите Σ. Тогда конкатенацией языков L_1 и L_2*

(обозначаем $L_1 \,\&\, L_2$) называют язык L, состоящий из всевозможных из конкатенаций слов первого языка со словами второго:

$$L = L_1 \,\&\, L_2 = \{w_1 w_2 : w_1 \in L_1, w_2 \in L_2\}.$$

Используя равенства $x \,\&\, \varepsilon = \varepsilon \,\&\, x = x$, сразу из определения можно сделать такой вывод.

Следствие 103. *Если $\varepsilon \in L_1$, то $L_2 \subseteq L_1 \,\&\, L_2$, а если $\varepsilon \in L_2$, то $L_1 \subseteq L_1 \,\&\, L_2$.*

Как и для слов, очень часто знак конкатенации языков пропускают и записывают конкатенацию просто как $L_1 L_2$.

Если производить многократную конкатенацию языка L с собой же, можно прийти к понятию «степени» языка L:

$$L^0 = \{\varepsilon\}; \quad L^{i+1} = L^i \,\&\, L, \ i \in \omega.$$

В частности, $L^1 = L^0 \,\&\, L = \{\varepsilon\} \,\&\, L = L$. Таким образом, язык L^i содержит всевозможные слова, которые можно разбить на i подряд идущих слов, принадлежащих L.

Определение 130 (Итерация)**.** *Итерация языка L — это язык L^*, состоящий из всех слов, которые можно разбить на несколько подряд идущих слов из L:*

$$L^* = \{w_1 \dots w_k : k \in \omega, w_1, \dots, w_k \in L\}.$$

В частности, при $k = 0$ получим $\varepsilon \in L^*$.

Итерацию можно определить как объединение всех степеней:

$$L^* = \bigcup_{i \in \omega} L^i.$$

Часто удобно рассматривать «усечённую» итерацию языка L, обозначаемую L^+, которая не содержит пустого слова, если его нет в языке L:

$$L^+ = \bigcup_{i > 0} L^i.$$

Это не новая операция, а просто удобное сокращение, потому что $L^+ = L^* \,\&\, L$.

Отметим также, что алфавит $\Sigma = \{a_1, \ldots, a_m\}$ одновременно является и конечным языком, состоящим в точности из всех однобуквенных слов. Тогда введённое ранее обозначение Σ^* для множества всех слов в алфавите Σ, включая и пустое, совпадает с определением итерации Σ^* этого языка.

Регулярные выражения над алфавитом Σ можно определить как формулы над множеством констант, содержащим ε, $\varnothing$ и все буквы из Σ, и операций $+$, $\&$, $*$.

Определение 131 (Регулярное выражение)**.** *Зафиксируем некоторый алфавит Σ. Регулярным выражением над алфавитом Σ называется слово, индуктивно построенное из символов Σ, а также $\varnothing$, ε, $+$, $\&$, $*$ и скобок следующим образом:*

1) *ε — регулярное выражение;*
2) *$\varnothing$ — регулярное выражение;*
3) *каждый символ $a \in \Sigma$ — регулярное выражение.*

Первые три пункта образуют базис индукционного определения. Пусть теперь r_1 и r_2 — регулярные выражения. Тогда регулярными выражениями будут строки следующих трёх видов:

4) $(r_1 + r_2)$; 5) $(r_1 \,\&\, r_2)$; 6) r_1^*.

Как мы уже упомянули, знак конкатенации $\&$ часто опускают и пишут просто $(r_1 r_2)$ вместо $(r_1 \,\&\, r_2)$. Чтобы сократить количество используемых скобок, считают, что операции в регулярных выражениях имеют следующий приоритет в порядке убывания: итерация $*$, конкатенация $\&$, объединение $+$. Последние две операции ассоциативны (см. предложение 104 на противоположной странице), поэтому при наличии подряд идущих одинаковых действий скобки также не нужны.

С каждым регулярным выражениям r связывается язык $L(r)$, который оно задаёт.

Определение 132 (Язык регулярного выражения). *Язык $L(r)$, задаваемый регулярным выражением r (также говорят, что r определяет/представляет/описывает язык $L(r)$), определяется по индукции в соответствии с построением регулярного выражения:*

1) $L(\varepsilon) = \{\varepsilon\}$ — *язык, состоящий из одного пустого слова;*
2) $L(\varnothing) = \varnothing$ — *пустой язык;*
3) $L(a) = \{a\}$ — *язык состоящий из одного слова a;*
4) $L(r_1 + r_2) = L(r_1) \cup L(r_2)$ — *объединение языков;*
5) $L(r_1 \,\&\, r_2) = L(r_1) \,\&\, L(r_2)$ — *конкатенация языков;*
6) $L(r_1^*) = (L(r_1))^*$ — *итерация языка.*

Язык, задаваемый каким-либо регулярным выражением, называется регулярным.

Из определения сразу получаем, что если r — регулярное выражение над алфавитом Σ, то $L(r)$ будет языком в алфавите Σ, так как его слова никаких других символов содержать не могут.

Определение 133 (Эквивалентность регулярных выражений). *Регулярные выражения r_1 и r_2 называются эквивалентными, если совпадают задаваемые ими языки: $L(r_1) = L(r_2)$. В этом случае пишем $r_1 \equiv r_2$.*

Нетрудно проверить, например, такие эквивалентности.

Предложение 104. *Имеют место следующие эквивалентности для любых регулярных выражений r, q, p:*

1) $r + p \equiv p + r$ *(коммутативность объединения),*
2) $(r + p) + q \equiv r + (p + q)$ *(ассоциативность объединения),*
3) $(rp)q \equiv r(pq)$ *(ассоциативность конкатенации),*
4) $(r^*)^* \equiv r^*$ *(идемпотентность итерации),*
5) $(r + p)q \equiv rq + pq$ *(дистрибутивность).*

Доказательство. Задача 237 на стр. 329. □

Докажем в качестве примера не столь очевидное утверждение.

Пример 72. *Продемонстрируем эквивалентность* $(r+p)^* \equiv (r^*p^*)^*$.

Пусть $L_1 = L((r+p)^*)$ *— язык, представляемый левым регулярным выражением, а* $L_2 = L((r^*p^*)^*)$ *— правым. С помощью* L_r *и* L_p *обозначим языки, задаваемые регулярными выражениями* r *и* p *соответственно. Пустое слово* ε *принадлежит обоим языкам.*

Если взять непустое слово $w \in L_1$*, то по определению итерации оно представимо в виде конкатенации подслов, принадлежащих языку* $L_r \cup L_p$*. Но этот язык является подмножеством языка* $L' = L_r^* L_p^*$ *(почему?). Поэтому* $w \in L_2 = (L')^*$.

Обратно, пусть $w \in L_2$*, тогда* w *представимо как конкатенация подслов, принадлежащих языку* L'*. Каждое из таких подслов* v *представимо в виде* $v = v_1^1 \dots v_k^1 v_1^2 \dots v_\ell^2$*, где для всех* $i = 1, \dots, k$ *выполняется* $v_i^1 \in L_r$*, а для всех* $j = 1, \dots, \ell$ *выполнено* $v_j^2 \in L_p$ *(возможно, что* k *или* ℓ *равно нулю). Но это значит, что* w *является конкатенацией подслов, каждое из которых принадлежит* $L_r \cup L_p$ *и, следовательно,* $w \in L_1$.

Рассмотрим несколько примеров регулярных выражений и представляемых ими языков.

Пример 73. *Регулярное выражение* $(0+1)^*$ *представляет множество всех слов в алфавите* $\{0,1\}$.

Пример 74. *Регулярное выражение* $11(0+1)^*001$ *представляет язык, состоящий из всех слов в алфавите* $\{0,1\}$*, которые начинаются на «11», а заканчиваются на «001».*

Пример 75. *Регулярное выражение* $(1+01+001)^*(\varepsilon+0+00)$ *представляет язык, состоящий из всех слов в алфавите* $\{0,1\}$*, которые не содержат подслово «000» (см. задачу 238 на стр. 329).*

Пример 76. *Регулярное выражение* $1^*(01^*01^*)^*$ *представляет язык* $L_{0\text{ч}}$*, состоящий из всех слов в алфавите* $\{0,1\}$*, в которых число нулей чётно.*

Действительно, каждое слово из $L_{0\text{ч}}$ *либо вообще не содержит нулей, то есть входит в язык, представляемый выражением* 1^**, либо может быть разбито на блоки вида* 01^i01^j*,* $i, j \geqslant 0$*, которым, быть может, предшествует блок единиц. Выражение* (01^*01^*)*, очевидно задаёт один такой блок, а его итерация — произвольную последовательность таких блоков.*

Пример 77. *Построим теперь регулярное выражение, представляющее язык* $L_{0\text{ч}1\text{ч}}$*, который состоит из всех слов в алфавите* $\{0,1\}$*, содержащих чётное число нулей и чётное число единиц.*

Пусть $w = a_1 a_2 \dots a_n$ *— произвольное слово языка* $L_{0\text{ч}1\text{ч}}$*, где* $a_i \in \{0,1\}$*,* $i = 1, \dots, n$*. Тогда, разумеется,* n *— чётно, пусть* $n = 2k$*. Разобьём слово*

w на пары соседних букв: $w = p_1 \dots p_k$, где $p_i = a_{2i-1}a_{2i}$, $i = 1, 2, \dots, k$. Каждая пара p_i может иметь один из четырёх видов: 00, 11, 01 *или* 10. *Пар вида* 00 *и* 11 *может быть сколько угодно, а пар вида* 01 *и* 10 *обязательно чётное количество. Поэтому w разбивается на блоки, каждый из которых начинается одной из пар* 01 *или* 10 *и содержит ещё одну такую пару. Каждый такой блок описывается выражением*

$$(01 + 10)(00 + 11)^*(01 + 10)(00 + 11)^*.$$

Перед первым блоком может быть префикс, состоящий только из пар вида 00 *и* 11. *Множество слов, состоящих из пар* 00 *и* 11, *задаётся выражением* $(00 + 11)^*$. *Отсюда получаем выражение $r_{0ч1ч}$, задающее язык $L_{0ч1ч}$:*

$$r_{0ч1ч} = (00 + 11)^*((01 + 10)(00 + 11)^*(01 + 10)(00 + 11)^*)^*.$$

§ 16.2. Автоматность регулярных языков

Покажем, что каждый регулярный язык можно распознать конечным автоматом.

Сначала докажем несколько вспомогательных утверждений, из них можно получить ещё одно доказательство свойств замкнутости автоматных языков.

Предложение 105. *Для каждого конечного автомата $\mathfrak{M}$ можно построить эквивалентный НКА $\mathfrak{N}$, имеющий одно заключительное состояние.*

Доказательство. Если множество заключительных состояний F автомата $\mathfrak{M}$ не содержит ни одного или содержит больше одного элемента, то поступим следующим образом. Добавим новое состояние q_f и команды вида $q, \varepsilon \to q_f$ для всех $q \in F$. Объявим в новом автомате q_f единственным заключительным состоянием (рис. 84 на следующей странице).

Если $\mathfrak{M}$ принимал слово w, то оно переводило q_0 в некоторое состояние $q \in F$ в автомате $\mathfrak{M}$. В новом автомате можно будет сделать ε-переход и попасть в q_f.

Обратно, если в новом автомате слово w переводит q_0 в q_f, то последним переходом на этом пути является ε-переход вида $q, \varepsilon \to q_f$,

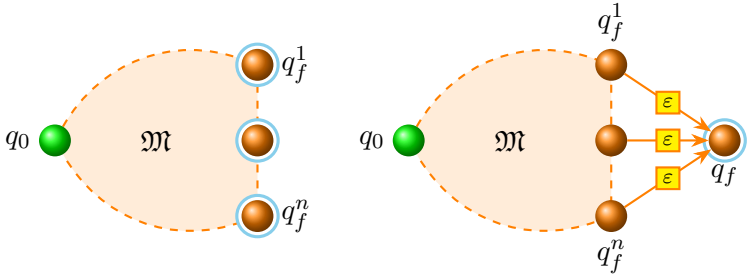


Рис. 84: Построение автомата с одним заключительным состоянием.

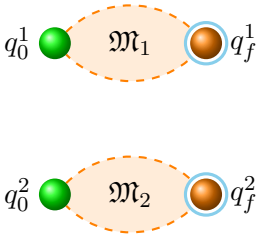


Рис. 85: Исходные автоматы.

так как других способов попасть в q_f просто нет. Но тогда в автомате $\mathfrak{M}$ состояние q было заключительным и слово w переводит $\mathfrak{M}$ из q_0 в q. Следовательно, $\mathfrak{M}$ принимает w. □

Предложение 106. *Для конечных автоматов $\mathfrak{M}_1$ и $\mathfrak{M}_2$ существует НКА $\mathfrak{N}$ такой, что $L(\mathfrak{N}) = L(\mathfrak{M}_1) \cup L(\mathfrak{M}_2)$.*

Доказательство. Без ограничения общности считаем, что исходные автоматы $\mathfrak{M}_1$ и $\mathfrak{M}_2$ не имеют общих состояний. Согласно предыдущему предложению можно считать, что $\mathfrak{M}_1$ и $\mathfrak{M}_2$ имеют по одному заключительному состоянию каждый. Пусть q_0^1 и q_0^2 являются начальными состояниями этих автоматов, а q_f^1 и q_f^2 заключительными соответственно (рис. 85).

Тогда автомат $\mathfrak{N}$ строим так: объединяем автоматы $\mathfrak{M}_1$ и $\mathfrak{M}_2$, добавляем новое начальное состояние q_0 и две команды $q_0, \varepsilon \to q_0^1$

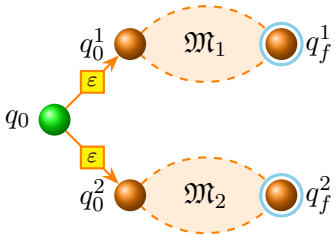


Рис. 86: Предложение 106, построение автомата для объединения.

и $q_0, \varepsilon \to q_0^2$. Заключительными в $\mathfrak{N}$ будут оба состояния q_f^1 и q_f^2 (рис. 86).

Нетрудно доказать, что $L(\mathfrak{N}) = L(\mathfrak{M}_1) \cup L(\mathfrak{M}_2)$. Если w принимается автоматом $\mathfrak{N}$, то q_0 переводится этим словом в одно из заключительных состояний q_f^i, $i \in \{1, 2\}$. Но любой такой путь должен начинаться с ε-перехода $q_0, \varepsilon \to q_0^i$. Следовательно, в автомате $\mathfrak{M}_i$ слово w переводит q_0^i в q_f^i, что означает, $\mathfrak{M}_i$ принимает w. Следовательно, $w \in L(\mathfrak{M}_1) \cup L(\mathfrak{M}_2)$.

Наоборот, если $w \in L(\mathfrak{M}_1) \cup L(\mathfrak{M}_2)$, то $w \in L(\mathfrak{M}_i)$, $i \in \{1, 2\}$. Следовательно, в автомате $\mathfrak{M}_i$ слово w переводит q_0^i в q_f^i. Но тогда в автомате $\mathfrak{M}_i$ слово w переводит q_0 в q_f^i: сначала ε-переход $q_0, \varepsilon \to q_0^i$, а затем тот же путь, что и в $\mathfrak{M}_i$. Поэтому w принимается автоматом $\mathfrak{N}$ и $w \in L(\mathfrak{N})$. □

Доказанное предложение ещё одним способом демонстрирует, что объединение автоматных языков снова будет автоматным (сравните с теоремой 98 на стр. 296). Используя его, можно новым способом доказать автоматность пересечения и разности автоматных языков. Сначала докажем автоматность дополнения.

Предложение 107. *Дополнение автоматного языка L в алфавите Σ снова является автоматным языком.*

Доказательство. В силу теоремы о детерминизации предполагаем, что язык L распознаётся детерминированным конечным автоматом $\mathfrak{M} = (Q, \Sigma, P, q_0, F)$. Тогда дополнение языка будет распознаваться автоматом $\mathfrak{N} = (Q, \Sigma, P, q_0, Q \setminus F)$. В самом деле, $w \in L$, если

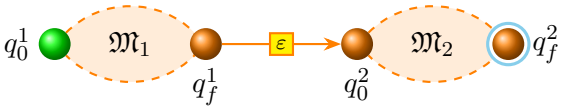


Рис. 87: Предложение 109, построение автомата для конкатенации.

и только если автомат $\mathfrak{M}$ распознаёт w, если и только если (лемма 96 на стр. 292) в диаграмме $\mathfrak{D}_{\mathfrak{M}}$ есть несущий слово w путь из q_0 в какое-то состояние $q_w \in F$. В силу детерминированности для каждого слова w такое состояние q_w существует и единственно (хотя, конечно, не для каждого слова w будет выполнено $q_w \in F$). Но тогда получаем, что $w \in \bar{L}$, то есть $w \notin L$, тогда и только тогда, когда $q_w \notin F$, то есть $q_w \in Q \setminus F$. Это и означает, что автомат $\mathfrak{N}$ распознаёт $\bar{L}$. □

Следствие 108. *Пересечение и разность автоматных языков снова будут автоматными языками.*

Доказательство. $L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$, $L_1 \setminus L_2 = L_1 \cap \bar{L}_2$. □

Аналогичную технику соединения автоматов можно применить для доказательства замкнутости класса автоматных языков относительно конкатенации.

Предложение 109. *Для конечных автоматов $\mathfrak{M}_1$ и $\mathfrak{M}_2$ существует НКА $\mathfrak{N}$ такой, что $L(\mathfrak{N}) = L(\mathfrak{M}_1)\ \&\ L(\mathfrak{M}_2)$.*

Доказательство. Как и в предложении 106 на стр. 316 считаем, что у автоматов $\mathfrak{M}_1$ и $\mathfrak{M}_2$ нет общих состояний, начальными являются q_0^1 и q_0^2, а заключительное состояние в каждом из них только одно: q_f^1 и q_f^2 соответственно (рис. 85 на стр. 316).

Автомат $\mathfrak{N}$ получается следующим образом: объединяем автоматы $\mathfrak{M}_1$ и $\mathfrak{M}_2$, добавляем команду $q_f^1, \varepsilon \to q_0^2$, начальным состоянием $\mathfrak{N}$ будет q_0^1, заключительным — q_f^2 (рис. 87).

Очевидно, $L(\mathfrak{N}) = L(\mathfrak{M}_1)\ \&\ L(\mathfrak{M}_2)$. Если w принимается автоматом $\mathfrak{N}$, то q_0 переводится в q_f^2. Единственный способ это сделать — пройти по ε-переходу $q_f^1, \varepsilon \to q_0^2$. Следовательно, путь, несущий w, имеет вид $(q_0^1, \ldots, q_f^1, q_0^2, \ldots, q_f^2)$. Пусть w_1 — слово, которое несёт

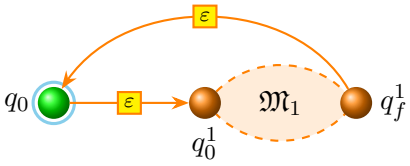


Рис. 88: Предложение 110, построение автомата для итерации.

путь $(q_0^1, \ldots, q_f^1)$, а w_2 — слово, которое несёт путь $(q_0^2, \ldots, q_f^2)$. Тогда $w = w_1 w_2$. Слово w_1 переводит q_0^1 в q_f^1, поскольку q_f^1 является заключительным в $\mathfrak{M}_1$, то w_1 принимается автоматом $\mathfrak{M}_1$. Слово w_2 переводит q_0^2 в q_f^2, так как q_f^2 — заключительное состояние $\mathfrak{M}_2$, то w_2 принимается автоматом $\mathfrak{M}_2$. Итак, мы получили, что $w = w_1 w_2$, где $w_1 \in L(\mathfrak{M}_1)$ и $w_2 \in L(\mathfrak{M}_2)$, то есть $w \in L(\mathfrak{M}_1)\ \&\ L(\mathfrak{M}_2)$.

Обратно: пусть $w \in L(\mathfrak{M}_1)\ \&\ L(\mathfrak{M}_2)$, это означает, что $w = w_1 w_2$ для некоторых $w_1 \in L(\mathfrak{M}_1)$ и $w_2 \in L(\mathfrak{M}_2)$. Если w_1 принимается автоматом $\mathfrak{M}_1$, то q_0^1 переводится словом w_1 в q_f^1. Так как w_2 принимается автоматом $\mathfrak{M}_2$, то q_0^2 переводится словом w_2 в q_f^2. Но тогда w может перевести $\mathfrak{N}$ из q_0^1 в q_f^2, то есть $w \in L(\mathfrak{N})$. □

Похожим образом можно продемонстрировать, что класс автоматных языков замкнут относительно итерации.

Предложение 110. *Для конечного автомата $\mathfrak{M}_1$ существует НКА $\mathfrak{N}$ такой, что $L(\mathfrak{N}) = (L(\mathfrak{M}_1))^*$.*

Доказательство. Как и раньше считаем, что автомат $\mathfrak{M}_1$ имеет начальное состояние q_0^1 и одно заключительное состояние q_f^1 (рис. 85 на стр. 316).

Новый автомат $\mathfrak{N}$ строим, добавив новое начальное состояние q_0 и две команды: $q_0, \varepsilon \to q_0^1$ и $q_f^1, \varepsilon \to q_0$, единственным заключительным состоянием будет q_0 (рис. 88).

Докажем, что $L(\mathfrak{N}) = (L(\mathfrak{M}_1))^*$. Пусть $w \in L(\mathfrak{N})$, то есть слово w переводит q_0 в q_0 в автомате $\mathfrak{N}$: есть путь $R = (q_0, \ldots, q_0)$, несущий w. Используем индукцию по длине пути.

Рис. 89: Автоматы для регулярных выражений $\varnothing$, ε и a.

Базис индукции: если длина равна нулю, то путь состоит из одной вершины q_0 и несёт пустое слово ε, которое принадлежит итерации любого языка.

Индукционный шаг: пусть длина пути R больше нуля, а для всех путей, имеющих меньшую длину, утверждение уже доказано. Выберем в R предпоследнее вхождение q_0 и разобьём R на две части:

$$z = (\underbrace{q_0, \ldots,}_{R_1} \overbrace{q_0, \ldots, q_0}^{R_2}).$$

где R_2 уже не содержит вершин q_0 кроме крайних. По индукционному предположению путь R_1, который короче R, несёт слово $w_1 \in (L(\mathfrak{M}_1))^*$. Так как фрагмент R_2 содержит хотя бы одно ребро, а единственное ребро, исходящее из q_0 — это ε-переход в q_0^1, то именно этот переход и является первым ребром в R_2. С другой стороны, единственное ребро, которое входит в q_0 — это ε-переход из q_f^1. Следовательно, R_2 имеет вид $(q_0, q_0^1, \ldots, q_f^1, q_0)$, где путь $(q_0^1, \ldots, q_f^1)$ несёт слово w_2. Это означает $w_2 \in L(\mathfrak{M}_1)$. Тогда из $w_1 \in (L(\mathfrak{M}_1))^*$ и $w_2 \in L(\mathfrak{M}_1)$ заключаем, что $w = w_1 w_2 \in (L(\mathfrak{M}_1))^*$.

Доказательство в обратную сторону использует индукцию по m, где m — это количество слов $L(\mathfrak{M}_1)$ из которых составлено $w \in (L(\mathfrak{M}_1))^*$: $w = w_1 \ldots w_m$ (задача 246 на стр. 330). □

Теорема 111. *Для каждого регулярного выражения r можно эффективно построить НКА $\mathfrak{N}$, распознающий язык $L(r)$.*

Доказательство. Построение НКА $\mathfrak{N}$ по регулярному выражению r проведём индукцией по построению r.

Базис индукции. Диаграммы автоматов для выражений: $\varnothing$, ε и $a \in \Sigma$ показаны на рис. 89.

Рис. 90: Автомат для распознавания слова $a_1 \dots a_n$.

Индукционный шаг. Если для регулярных выражений r_1 и r_2 уже построены автоматы $\mathfrak{M}_1$ и $\mathfrak{M}_2$ такие, что $L(\mathfrak{M}_1) = L(r_1)$ и $L(\mathfrak{M}_2) = L(r_2)$, то для выражений $(r_1 + r_2)$, $(r_1 \,\&\, r_2)$ и r_1^* используем предложения 106 на стр. 316, 109 на стр. 318 и 110 на стр. 319 соответственно. □

Из теорем 100 на стр. 301 и 111 на предыдущей странице непосредственно получаем

Следствие 112. *Для каждого регулярного выражения можно эффективно построить детерминированный конечный автомат, который распознаёт язык, представляемый этим выражением.*

Это утверждение — один из примеров теорем синтеза: по описанию задания (языка как регулярного выражения) эффективно строится программа (ДКА), его выполняющая.

Автомат, который строится в доказательстве предыдущей теоремы по регулярному выражению r, не всегда является самым простым.

Например, для распознавания выражения-слова $a_1 a_2 \dots a_n$, где $a_i \in \Sigma$, $i = 1, 2, \dots, n$, можно использовать автомат с $n + 1$ состоянием q_i, $i = 0, 1, 2, \dots, n$, и командами $q_{i-1}, a_i \to q_i$, в котором нет ε-переходов, необходимых в общей конструкции для конкатенации (рис. 90).

При построении автомата для объединения $\mathfrak{M}_1$ и $\mathfrak{M}_2$ можно слить их начальные состояния в одно, если в эти состояния нет переходов, тогда не потребуется новое начальное состояние. Можно также объединить их заключительные состояния, если из них нет переходов (рис. 91 на следующей странице). Однако нельзя объединять одновременно и начальные, и заключительные состояния, если начальное состояние является и заключительным хотя бы в одном автомате.

Если из заключительного состояния $\mathfrak{M}_1$ нет переходов в другие состояния, то при конкатенации его можно объединить с начальным

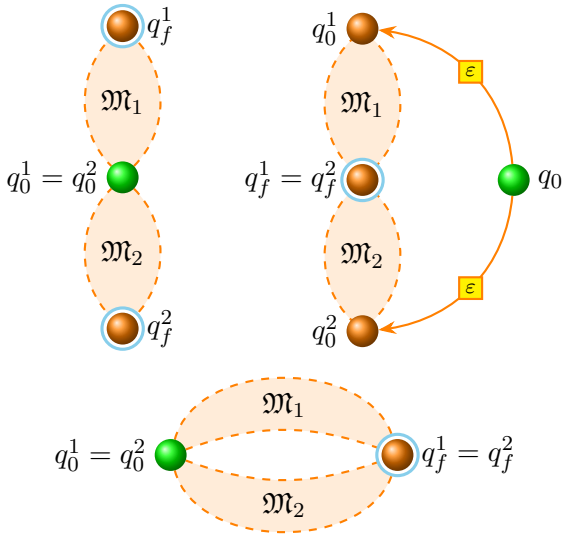


Рис. 91: «Оптимизированные» автоматы для объединения.

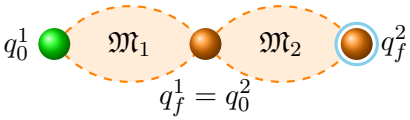


Рис. 92: «Оптимизированный» автомат для конкатенации.

состоянием $\mathfrak{M}_2$ (рис. 92). То же самое можно сделать, если в начальное состояние $\mathfrak{M}_2$ нет переходов.

Вместе с тем, утверждения задачи 245 на стр. 330 показывают, что даже наша общая конструкция достаточно экономна.

Пример 78. *Применим теорему 111 на стр. 320 к регулярному выражению*

$$r = (1 + 01 + 001)^*(\varepsilon + 0 + 00),$$

которое, как мы заметили в примере 75 на стр. 314, представляет язык, состоящий из всех слов, которые не содержат подслово «000».

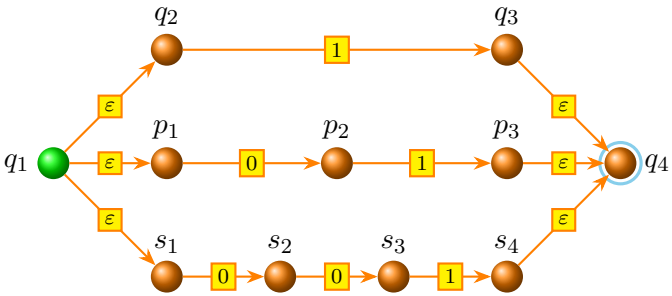


Рис. 93: Автомат для выражения $r_1 = 1 + 01 + 001$.

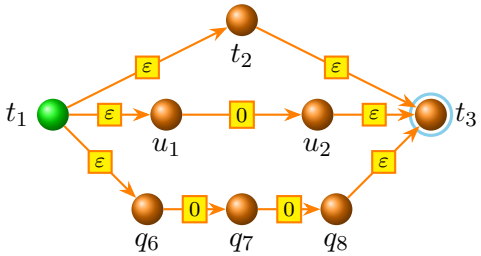


Рис. 94: Автомат для выражения $r_2 = \varepsilon + 0 + 00$.

Сначала построим автоматы для выражений-слов 1, 01, 001, ε, 0, 00. *Затем при помощи объединения построим из них автоматы для выражений* $r_1 = 1 + 01 + 001$ *(рис. 93) и* $\varepsilon + 0 + 00$ *(рис. 94).*

Как мы отмечали выше, эти автоматы можно упростить, «склеив» начальные и заключительные состояния исходных (рис. 95 на следующей странице).

Автомат $\mathfrak{M}_3$ *для выражения* $r_1^* = (1 + 01 + 001)^*$ *получается из* $\mathfrak{M}_1$ *добавлением нового начального состояния* q_0, ε*-перехода из* q_0 *в старое начальное состояние* p_1 *и* ε*-перехода из старого заключительного состояния* p_3 *в* q_0 *(рис. 96 на следующей странице).*

Результирующий автомат $\mathfrak{N}$ *для исходного выражения* r *получается последовательным соединением* $\mathfrak{M}_3$ *и* $\mathfrak{M}_2$. *Он представлен на рис. 97 на следующей странице.*

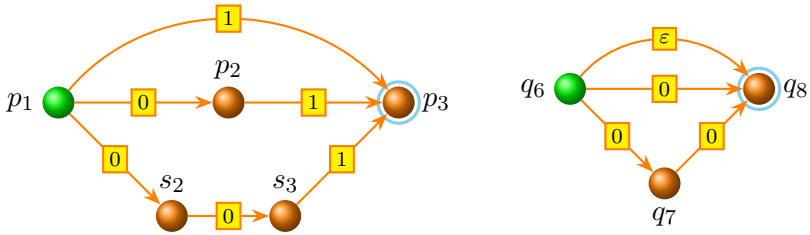


Рис. 95: Автоматы со «склеенными» состояниями.

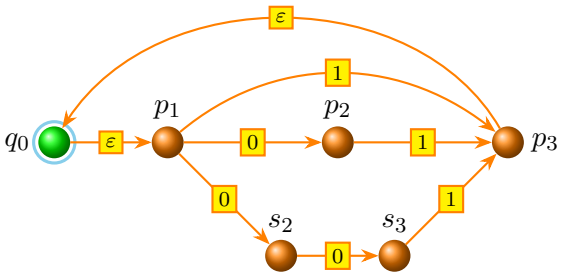


Рис. 96: Автомат для $(1 + 01 + 001)^*$.

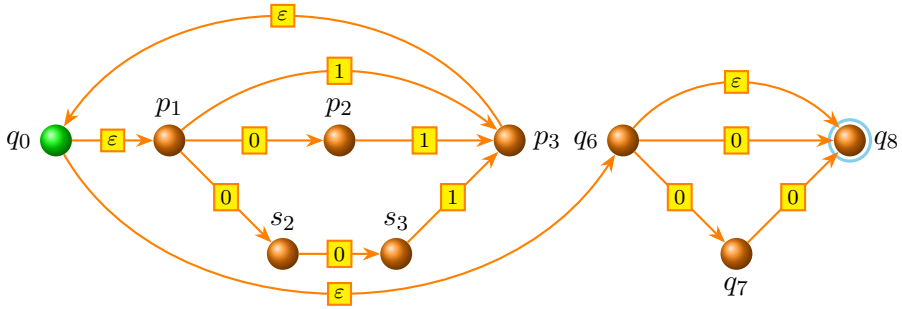


Рис. 97: Диаграмма автомата $\mathfrak{N}$, распознающего язык $L(r)$.

§ 16.3. Регулярность автоматных языков

В предыдущем параграфе мы показали, что по любому регулярному выражению, определяющему язык L, можно построить конечный автомат $\mathfrak{M}$, распознающий L. Справедливо и обратное утверждение — теорема анализа.

Теорема 113. *По всякому конечному автомату $\mathfrak{M} = (Q, \Sigma, P, q_0, F)$, распознающему язык L, можно построить регулярное выражение, которое представляет язык L.*

Доказательство. Доказательство теоремы будет следующим: мы построим всевозможные регулярные выражения r_{qp}, для которых выполняется

$$w \in L(r_{qp}) \iff (q, w) \vdash_{\mathfrak{M}}^{*} (p, \varepsilon).$$

Иначе говоря, r_{qp} задаёт в точности такие слова, с помощью которых автомат $\mathfrak{M}$ может перейти из состояния q в состояние p. Тогда, как нетрудно видеть, искомое регулярное выражение r строится при помощи суммирования по заключительным состояниям:

$$r = \sum_{f \in F} r_{q_0 f}.$$

Мы будем считать автомат $\mathfrak{M}$ недетерминированным, чтобы из его программы можно было выбрасывать команды по мере надобности. Построение выражений r_{qp} будет происходить индукцией по количеству команд в программе автомата сразу для всех пар состояний из Q. Будем считать, что команды автомата $\mathfrak{M}$ каким-либо образом пронумерованы, а автомат $\mathfrak{M}_n$ содержит первые n из этих команд. Остальные части автомата $\mathfrak{M}_i$ такие же как у $\mathfrak{M}$.

Базис индукции. Программа автомата $\mathfrak{M}_0$ не содержит ни одной команды. Тогда, очевидно, $r_{qq}^{(0)} = \varepsilon$ и $r_{qp}^{(0)} = \varnothing$ для $p \neq q$. Здесь мы записали, что по пустому слову автомат остаётся в том же состоянии, а перейти в другое не может никаким образом.

Индукционный шаг. Пусть для автомата $\mathfrak{M}_n$, содержащего n команд, всевозможные регулярные выражения $r_{qp}^{(n)}$, $q, p \in Q$, построены. Предположим, что $(n+1)$-я команда имеет вид $s, a \to t$,

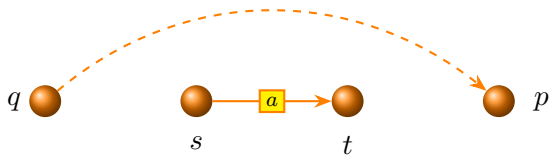


Рис. 98: Пути, не проходящие по новому ребру.

где $s, t \in Q$, $a \in \Sigma \cup \{\varepsilon\}$. Покажем, как построить $r_{qp}^{(n+1)}$. Будем рассматривать всевозможные пути P в диаграмме автомата $\mathfrak{M}_{n+1}$, ведущие из состояния q в состояние p. Эти пути можно разделить на два взаимоисключающих типа.

1) Пути P, которые не содержат ни одного ребра, соответствующего новой команде $s, a \to t$ (рис. 98). Тогда такие же пути существуют и в диаграмме автомата $\mathfrak{M}_n$, поэтому слова, которые несут все такие пути, задаются уже построенным по индукционному предположению регулярным выражением $r_{qp}^{(n)}$.

2) Пути, которые содержат хотя бы раз новое ребро (рис. 99 на противоположной странице). Каждый путь такого типа можно разделить на части, выделив все места, где встречается новое ребро:

$$\underbrace{q, \ldots, s}_{S}, \underbrace{t, \ldots, s}_{R_1}, \ldots, \underbrace{t, \ldots, s}_{R_k}, \underbrace{t, \ldots, p}_{T}.$$

Здесь при переходе от предыдущего фрагмента к следующему используется команда $s, a \to t$, а внутри каждого из фрагментов $S, R_1, \ldots, R_k, T$ это новое ребро не встречается. По индукционному предположению регулярным выражением $r_{qs}^{(n)}$ задаются все слова, которые могут нести пути вида S, регулярным выражением $r_{ts}^{(n)}$ задаются все слова, которые могут нести пути вида $R_1, \ldots, R_k$ (они все имеют идентичное описание), регулярным выражением $r_{tp}^{(n)}$ задаются все слова, которые могут нести пути вида T. Но тогда слова, которые могут нести пути P, можно описать с помощью выражения $r_{qs}^{(n)} \big(a r_{ts}^{(n)}\big)^* a r_{tp}^{(n)}$.

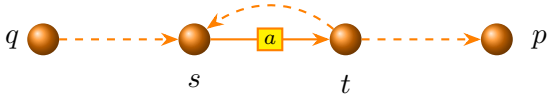


Рис. 99: Пути, проходящие по новому ребру.

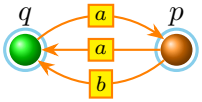


Рис. 100: Диаграмма автомата $\mathfrak{M}$ из примера 79.

Объединив выражения для этих двух случаев, окончательно получаем

$$r_{qp}^{(n+1)} = r_{qp}^{(n)} + r_{qs}^{(n)}\bigl(ar_{ts}^{(n)}\bigr)^* ar_{tp}^{(n)}.$$

В качестве окончательных выражений r_{qp} берём соответствующие $r_{qp}^{(M)}$, где M — количество команд в программе автомата $\mathfrak{M}$. □

Проиллюстрируем применение этой теоремы на несложном автомате.

Пример 79. *Рассмотрим автомат, который распознаёт язык в алфавите* $\Sigma = \{a, b\}$, *содержащий в точности такие слова, в которых буквы* b *могут стоять только на чётных позициях (рис. 100), начальным состоянием является* q.

Пронумеруем команды этого автомата: $1 : q, a \to p$, $2 : p, a \to q$, $3 : p, b \to q$. *Построение регулярных выражений представлено в следующих таблицах. Добавляем первую команду:*

i	0	$1 : q, a \to p$
$r_{qq}^{(i)}$	ε	$r_{qq}^{(0)} + r_{qq}^{(0)}\bigl(ar_{pq}^{(0)}\bigr)^* ar_{pq}^{(0)} \equiv \varepsilon + \varepsilon(a\varnothing)^* a\varnothing \equiv \varepsilon$
$r_{qp}^{(i)}$	$\varnothing$	$r_{qp}^{(0)} + r_{qq}^{(0)}\bigl(ar_{pq}^{(0)}\bigr)^* ar_{pp}^{(0)} \equiv \varnothing + \varepsilon(a\varnothing)^* a\varepsilon \equiv a$
$r_{pq}^{(i)}$	$\varnothing$	$r_{pq}^{(0)} + r_{pq}^{(0)}\bigl(ar_{pq}^{(0)}\bigr)^* ar_{pq}^{(0)} \equiv \varnothing + \varnothing(a\varnothing)^* a\varnothing \equiv \varnothing$
$r_{pp}^{(i)}$	ε	$r_{pp}^{(0)} + r_{pq}^{(0)}\bigl(ar_{pq}^{(0)}\bigr)^* ar_{pp}^{(0)} \equiv \varepsilon + \varnothing(a\varnothing)^* a\varepsilon \equiv \varepsilon$

Вторую:

i	1	$2: p, a \to q$
$r_{qq}^{(i)}$	ε	$r_{qq}^{(1)} + r_{qp}^{(1)}\left(ar_{qp}^{(1)}\right)^* ar_{qq}^{(1)} \equiv \varepsilon + a(aa)^* a\varepsilon \equiv (aa)^*$
$r_{qp}^{(i)}$	a	$r_{qp}^{(1)} + r_{qp}^{(1)}\left(ar_{qp}^{(1)}\right)^* ar_{qp}^{(1)} \equiv a + a(aa)^* aa \equiv a(aa)^*$
$r_{pq}^{(i)}$	$\varnothing$	$r_{pq}^{(1)} + r_{pp}^{(1)}\left(ar_{qp}^{(1)}\right)^* ar_{qq}^{(1)} \equiv \varnothing + \varepsilon(aa)^* a\varepsilon \equiv a(aa)^*$
$r_{pp}^{(i)}$	ε	$r_{pp}^{(1)} + r_{pp}^{(1)}\left(ar_{qp}^{(1)}\right)^* ar_{qp}^{(1)} \equiv \varepsilon + \varepsilon(aa)^* aa \equiv (aa)^*$

Третью:

i	2	$3: p, b \to q$
$r_{qq}^{(i)}$	$(aa)^*$	$r_{qq}^{(2)} + r_{qp}^{(2)}\left(br_{qp}^{(2)}\right)^* br_{qq}^{(2)} \equiv (aa)^* + a(aa)^*(ba(aa)^*)^* b(aa)^*$
$r_{qp}^{(i)}$	$a(aa)^*$	$r_{qp}^{(2)} + r_{qp}^{(2)}\left(br_{qp}^{(2)}\right)^* br_{qp}^{(2)} \equiv a(aa)^* + a(aa)^*(ba(aa)^*)^* ba(aa)^*$
$r_{pq}^{(i)}$	$a(aa)^*$	$r_{pq}^{(2)} + r_{pp}^{(2)}\left(br_{qp}^{(2)}\right)^* br_{qq}^{(2)} \equiv a(aa)^* + (aa)^*(ba(aa)^*)^* b(aa)^*$
$r_{pp}^{(i)}$	$(aa)^*$	$r_{pp}^{(2)} + r_{pp}^{(2)}\left(br_{qp}^{(2)}\right)^* br_{qp}^{(2)} \equiv (aa)^* + (aa)^*(ba(aa)^*)^* ba(aa)^*$

Так как начальное состояние — q, а заключительные — оба, то нам нужно выражение $r_{qq}^{(3)} + r_{qp}^{(3)}$:

$$r = (aa)^* + a(aa)^*(ba(aa)^*)^* b(aa)^* + a(aa)^* + a(aa)^*(ba(aa)^*)^* ba(aa)^*.$$

Упростив его (задача 248 на стр. 330), получим $(aa + ab)^*(a + \varepsilon)$.

Как видим, даже для очень простого автомата получилось чрезвычайно сложное регулярное выражение. Поэтому к такому методу построения выражений имеет смысл прибегать, только когда другие возможности исчерпаны. Обычно регулярное выражение проще построить, просто внимательно проанализировав язык, который требуется описать.

По результатам двух последних параграфов можно сделать такой вывод.

Следствие 114. *Класс автоматных языков совпадает с классом регулярных языков.*

Задачи

235. Определить конкатенацию для следующих пар языков L_1 и L_2:

(а) $L_1 = \{a, ab, abb\}$ и $L_2 = \{\varepsilon, a, b, ab, ba\}$;

(б) $L_1 = \{\varepsilon, a, ab, abb\}$ и $L_2 = \{a, b, abb, ba\}$;

(в) $L_1 = \{\varepsilon, a, b, ab, aba\}$ и $L_2 = \{\varepsilon, a, b, ab, ba\}$. ▼

236. Пусть $L = \{baa, bab, bba, bbb\}$. Какой из следующих языков является итерацией L^* этого языка?

(а) $\{w : w = bw'$ и $|w|$ делится на $3\} \cup \{\varepsilon\}$;

(б) $\{w : w = bw'$ и $|w| \geqslant 3\} \cup \{\varepsilon\}$;

(в) $\{w : w = x_1x_2x_3 \ldots x_{3n}$, $x_i \in \{a, b\}$ и $x_{3i+1} = b$ для всех $i < n\} \cup \{\varepsilon\}$;

(г) $\{w : w = bw'$ и $|w| \geqslant 12\}$. ▼

237. Доказать эквивалентности предложения 104 на стр. 313. ▼

238. Доказать правильность регулярного выражения в примере 75 на стр. 314. ▼

239. Доказать следующие эквивалентности для регулярных выражений:

(а) $p^*(p + q)^* \equiv p^*(qp^*)^* \equiv (p + qp^*)^* \equiv (p + q)^*$;

(б) $p(qp)^* \equiv (pq)^*p$;

(в) $(p^*q^*)^* \equiv (q^*p^*)^*$;

(г) $(pq)^+(q^*p^* + q^*) \equiv (pq)^*pq^+p^*$. ▼

240. Построить регулярное выражение, задающее язык язык L в алфавите $\Sigma = \{0, 1\}$:

(а) $L = \{w : w$ содержит нечётное число букв 0 и чётное число букв $1\}$;

(б) $L = \{w : w$ содержит подслово 001 или подслово $110\}$;

(в) $L = \{w : w$ содержит по крайней мере мере два подряд идущих $0\}$;

(г) $L = \{w : w$ не содержит подслов 011 и $010\}$. ▼

241. Определить, какой язык представляется следующими регулярными выражениями:

(а) 0^*1^*0;

(б) 01^*0;

(в) $(01^*)^*0$;

(г) $(00 + 11 + (01 + 10)(00 + 11)^*(01 + 10))^*$. ▼

242. Упростить следующие регулярные выражения:

(а) $00^*0 + (00)^*$;

(б) $(0 + 1)(\varepsilon + 00)^+ + (0 + 1)$;

(в) $(0 + \varepsilon)0^*1$. ▼

243. Пусть w — произвольное слово длины k, $m_1, \ldots, m_n$ — попарно различные натуральные числа, упорядоченные по возрастанию. Доказать, что язык $\{w^{m_1}, \ldots, w^{m_n}\}$ можно описать регулярным выражением длины не большей $4km_n + 4n - 7$ (с учётом всех необходимых по определению символов). ▼

244. Выше в задаче 231 на стр. 309 предлагалось построить автомат, который проверяет правильность сложения. Построить регулярное выражение, задающее распознаваемый этим автоматом язык S, то есть следующее множество слов в алфавите $\{0,1\}^3$: ▼

$$S = \left\{ \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} \cdots \begin{bmatrix} a_n \\ b_n \\ c_n \end{bmatrix} : c_n \ldots c_1 \text{ — сумма двоичных чисел } a_n \ldots a_1 \text{ и } b_n \ldots b_1 \right\}.$$

245. Пусть $\mathfrak{N}_r$ — это автомат, который строится в доказательстве теоремы 111 на стр. 320 по регулярному выражению r. Доказать следующие утверждения:

(а) в диаграмме $\mathfrak{N}_r$ из каждой вершины выходит не более двух рёбер, а из заключительных — не более одного;

(б) число состояний $\mathfrak{N}_r$ не более чем в три раза превосходит длину выражения r, то есть $|Q| \leqslant 3|r|$;

(в) при использовании оптимизированных методов число состояний $\mathfrak{N}_r$ не более чем в два раза превосходит длину выражения r, то есть $|Q| \leqslant 2|r|$. ▼

246. Завершить доказательство предложения 110 на стр. 319, показать, что если $w \in (L(\mathfrak{M}_1))^*$, то $w \in L(\mathfrak{N})$. ▼

247. Применить процедуру детерминизации из теоремы 100 на стр. 301 и построить ДКА, эквивалентный НКА $\mathfrak{N}$ из примера 78 на стр. 322. ▼

248. С помощью эквивалентных преобразований регулярных выражений упростить результат r, полученный в примере 79 на стр. 327. ▼

Глава 17

Кодирования. Неавтоматные языки

Краткое содержание: кодирование для языков, построение конечного автомата для кода языка, гомоморфные образы и прообразы языков как частные случаи кодирования, проекции, разнозначность кодирования, критерий Маркова, оптимальное кодирование Хаффмана, лемма о разрастании для автоматных языков, доказательство неавтоматности языков, примеры неавтоматных языков.

Ключевые слова: схема кодирования, код языка, гомоморфизмы, гомоморфный образ, гомоморфный прообраз, проекция, разнозначный гомоморфизм, критерий Маркова, оптимальное кодирование, алгоритм Хаффмана, лемма о разрастании, неавтоматный язык.

§ 17.1. Кодирование языков

Рассмотрим ещё одну очень мощную операцию, которую можно выполнять над языками.

Определение 134 (Схема кодирования)**.** *Пусть Σ и Ω — два алфавита. Схемой кодирования из Σ в Ω называется конечное множество C пар вида (u, v), где $u \in \Sigma^*$, $v \in \Omega^*$. В частности, слова u, v могут быть пустыми. Другими словами, схема кодирования — это конечное бинарное отношение между Σ^* и Ω^*.*

Слово $y \in \Omega^*$ получено из $x \in \Sigma^*$ при помощи схемы кодирования C, если $x = u_1 \dots u_n$, $y = v_1 \dots v_n$ и $(u_1, v_1), \dots, (u_n, v_n) \in C$ для некоторых $u_1, \dots, u_n \in \Sigma^*$ и $v_1, \dots, v_n \in \Omega^*$. Ещё говорят, что y — это код слова x. В частности, при $n = 0$ из пустого слова всегда можно получить пустое.

С помощью $C[x]$ обозначаем множество кодов слова x, которые получены при помощи C.

Иначе говоря, чтобы закодировать слово x мы разбиваем его на части, для каждой из этих частей находим в C какое-то соответствие, а затем «собираем» из них результат y.

Отметим, что для одного слова u может быть несколько пар вида (u, v), при кодировании можно выбрать любую из них. Кроме того, разбиение слова x на части тоже неоднозначно. Поэтому для одного исходного слова при помощи кодирования можно получить разные результаты.

Пример 80. *Рассмотрим следующую схему кодирования из алфавита $\{a, b\}$ в алфавит $\{0, 1\}$:*

$$C = \{(ab, 0), (a, 1), (b, 11), (b, 101)\}.$$

Тогда слово aab можно закодировать следующими способами:

$$aab = (a)(a)(b) \mapsto (1)(1)(11) = 1111;$$
$$aab = (a)(a)(b) \mapsto (1)(1)(101) = 11101;$$
$$aab = (a)(ab) \mapsto (1)(0) = 10.$$

Не обязательно, что все слова исходного алфавита можно закодировать, например, при помощи схемы $C = \{(ab, 0), (a, 1)\}$ нельзя закодировать слово b.

Определение 135 (Код языка)**.** *Пусть C — схема кодирования из алфавита Σ в алфавит Ω, L — язык в алфавите Σ. Тогда с помощью $C(L)$ обозначаем код языка L, то есть множество всех слов, которые можно получить, кодируя слова из L при помощи C:*

$$C(L) = \bigcup_{x \in L} C[x].$$

Операцию построения кода языка или отдельного слова назовём кодированием.

Например, если взять язык L, заданный регулярным выражением a^*b^* и схему кодирования C из предыдущего примера, то язык $C(L)$ будет задаваться выражением $1^*(\varepsilon + 0)(11 + 101)^*$.

Как видим, с помощью кодирований можно задавать довольно сложные преобразования языков. Тем не менее оказывается, класс автоматных языков замкнут относительно произвольных кодирований.

Теорема 115. *Класс автоматных языков замкнут относительно любых кодирований.*

Доказательство. Пусть L — автоматный язык в алфавите Σ, $\mathfrak{M} = (Q, \Sigma, P, q_0, F)$ — детерминированный конечный автомат, который распознаёт L, C — схема кодирования из Σ в алфавит Ω.

Построим новый автомат $\mathfrak{N} = (Q', \Omega, P', q_0, F)$, который будет распознавать $C(L)$. При построении будет выполняться $Q \subseteq Q'$, поэтому q_0 и F мы можем оставить теми же самыми.

Рассмотрим любую пару $c = (u, v) \in C$, где $v = a_1 \dots a_k$, $a_i \in \Omega$, $i = 1, \dots, k$. Для каждого $q \in Q$ введём в Q' новые состояния q^{ci}, $i = 2, \dots, k$. Если $(q, u) \vdash^*_{\mathfrak{M}} (p, \varepsilon)$ и $k \geqslant 2$, то P' включает следующее множество команд $P'_{q,c}$:

$$P'_{q,c} = \{q, a_1 \to q^{c,2};\ q^{c,2}, a_2 \to q^{c,3};\ \dots \\ \dots;\ q^{c,k-1}, a_{k-1} \to q^{c,k};\ q^{c,k}, a_k \to p\}.$$

Если $k = 1$ и $v = a_1$, то будет только команда: $P'_{q,c} = \{q, a_1 \to p\}$, если $k = 0$ и $v = \varepsilon$, то $P'_{q,c} = \{q, \varepsilon \to p\}$. Заметим, что эти команды

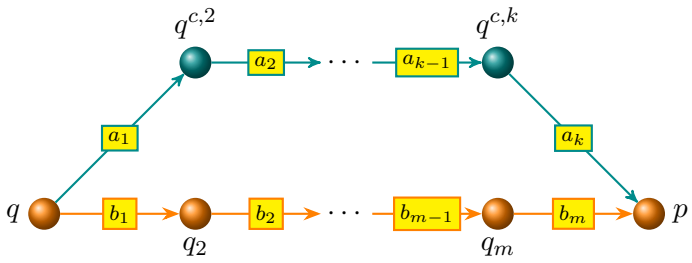


Рис. 101: Построение автомата $\mathfrak{N}$ в теореме 115.

гарантируют, что $(q, v) \vdash^*_{\mathfrak{N}} (p, \varepsilon)$. Таким образом,

$$Q' = Q \cup \bigcup_{q \in Q} \bigcup_{c \in C} \{q^{c,i} : c = (u, a_1 \dots a_k), i = 2, \dots, k\};$$
$$P' = \bigcup_{q \in Q} \bigcup_{c \in C} P'_{q,c}.$$

Менее формально можно описать предложенную конструкцию так (рис. 101): если в старом автомате был путь $P_u = (q, q_2, \dots, q_m, p)$, который нёс слово $u = b_1 \dots b_m$ из пары $(u, v) \in C$ (показан оранжевым цветом), то в новом автомате мы создаём новый уникальный путь $P'_u = (q, q^{c,2}, \dots, q^{c,k}, p)$, который несёт слово $v = a_1 \dots a_k$ (показан тёмно-голубым). Уникальность в данном случае означает, что разные такие пути не имеют общих вершин, кроме, может быть, крайних.

Покажем, что так построенный (вообще говоря, недетерминированный) автомат $\mathfrak{N}$ распознаёт язык $C(L)$.

Пусть $w' \in C(L)$. Это означает $w' = v_1 \dots v_n$, причём существует $w = u_1 \dots u_n \in L$ и $(u_1, v_1), \dots, (u_n, v_n) \in C$. Так как $w \in L$, то $\mathfrak{M}$ распознаёт w, то есть мы имеем

$$(q_0 = q_{j_0}, u_1 \dots u_n) \vdash^*_{\mathfrak{M}} (q_{j_1}, u_2 \dots u_n) \vdash^*_{\mathfrak{M}} (q_{j_2}, u_3 \dots u_n) \vdash^*_{\mathfrak{M}} \cdots$$
$$\cdots \vdash^*_{\mathfrak{M}} (q_{j_{n-1}}, u_n) \vdash^*_{\mathfrak{M}} (q_{j_n}, \varepsilon),$$

причём $q_{j_n} \in F$. Тогда $(q_{j_{i-1}}, u_i) \vdash^*_{\mathfrak{M}} (q_{j_i}, \varepsilon)$, поэтому P' будет включать в себя множество команд $P'_{q_{j_{i-1}}, (u_i, v_i)}$, благодаря которым получаем

$(q_{j_{i-1}}, v_i) \vdash^*_{\mathfrak{N}} (q_{j_i}, \varepsilon)$. Но тогда для слова w' имеем

$$(q_0 = q_{j_0}, v_1 \dots v_n) \vdash^*_{\mathfrak{N}} (q_{j_1}, v_2 \dots v_n) \vdash^*_{\mathfrak{N}} (q_{j_2}, v_3 \dots v_n) \vdash^*_{\mathfrak{N}} \cdots \\ \cdots \vdash^*_{\mathfrak{N}} (q_{j_{n-1}}, v_n) \vdash^*_{\mathfrak{N}} (q_{j_n}, \varepsilon),$$

что означает: $\mathfrak{N}$ принимает w'.

Пусть теперь слово w' принимается автоматом $\mathfrak{N}$: $(q_0, w') \vdash^*_{\mathfrak{N}} (q_f, \varepsilon)$. В этой последовательности конфигураций выделим те, состояния в которых являются «старыми», принадлежащими Q:

$$(q_0 = q_{j_0}, v_1 \dots v_n) \vdash^*_{\mathfrak{N}} (q_{j_1}, v_2 \dots v_n) \vdash^*_{\mathfrak{N}} (q_{j_2}, v_3 \dots v_n) \vdash^*_{\mathfrak{N}} \cdots \\ \cdots \vdash^*_{\mathfrak{N}} (q_{j_{n-1}}, v_n) \vdash^*_{\mathfrak{N}} (q_{j_n} = q_f, \varepsilon),$$

где $q_{j_0}, \dots, q_{j_n} \in Q$, а между ними встречаются только новые состояния. Но множество команд P' так устроено, что из конфигурации $(q_{j_{i-1}}, v_i)$ можно попасть в (q_{j_i}, ε), не используя «старых» состояний, только если в C есть пара (u_i, v_i) и $(q_{j_{i-1}}, u_i) \vdash^*_{\mathfrak{M}} (q_{j_i}, \varepsilon)$. Тогда автомат $\mathfrak{M}$ на слове $w = u_1 \dots u_n$ работает так:

$$(q_0 = q_{j_0}, u_1 \dots u_n) \vdash^*_{\mathfrak{M}} (q_{j_1}, u_2 \dots u_n) \vdash^*_{\mathfrak{M}} (q_{j_2}, u_3 \dots u_n) \vdash^*_{\mathfrak{M}} \cdots \\ \cdots \vdash^*_{\mathfrak{M}} (q_{j_{n-1}}, u_n) \vdash^*_{\mathfrak{M}} (q_{j_n} = q_f, \varepsilon),$$

то есть принимает w. Следовательно, $w \in L$. Но $w' \in C[w]$, поэтому $w' \in C(L)$. □

Пример 81. *Рассмотрим схему кодирования* $C = \{c_1 = (ab, 0), c_2 = (a, 1), c_3 = (b, 11), c_4 = (b, 101)\}$ *и язык* $L = (ab^*a + b)^*$, *распознаваемый автоматом* $\mathfrak{M}$ *на рис. 102 на следующей странице. Для него получаем следующие соотношения:*

$$(q_0, ab) \vdash^*_{\mathfrak{M}} (q_1, \varepsilon); \quad (q_1, ab) \vdash^*_{\mathfrak{M}} (q_0, \varepsilon);$$
$$(q_0, a) \vdash^*_{\mathfrak{M}} (q_1, \varepsilon); \quad (q_1, a) \vdash^*_{\mathfrak{M}} (q_0, \varepsilon);$$
$$(q_0, b) \vdash^*_{\mathfrak{M}} (q_0, \varepsilon); \quad (q_1, b) \vdash^*_{\mathfrak{M}} (q_1, \varepsilon).$$

Тогда строим автомат $\mathfrak{N}$ *следующим образом (рис. 103 на следующей странице). Вверху диаграммы изображены новые промежуточные состояния для чтения подслов* 11 *из* c_3, *а внизу — подслов* 101 *из* c_4.

Важным частным случаем кодирований являются гомоморфизмы.

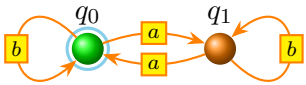


Рис. 102: Диаграмма автомата $\mathfrak{M}$ из примера 81

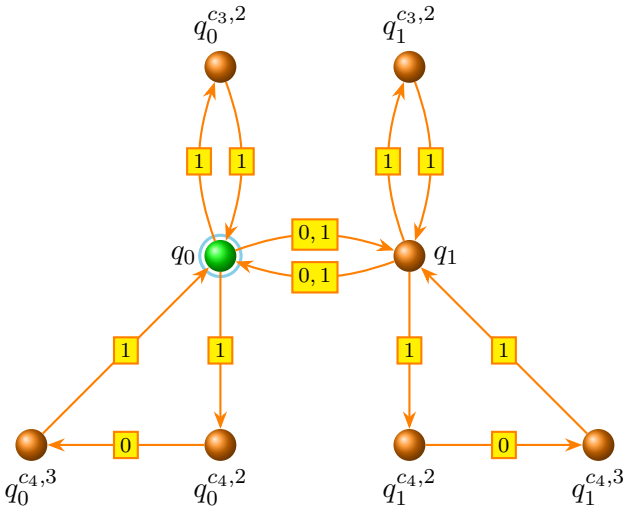


Рис. 103: Диаграмма автомата $\mathfrak{N}$ из примера 81

Определение 136 (Гомоморфизм языков)**.** *Функция* $\varphi : \Sigma^* \to \Omega^*$, *где* Σ *и* Ω *— два алфавита, называется* г о м о м о р ф и з м о м, *если выполняется следующее свойство: для всяких двух слов* $w_1, w_2 \in \Sigma^*$ *имеет место равенство*

$$\varphi(w_1 w_2) = \varphi(w_1)\varphi(w_2).$$

Из определения легко получить

Следствие 116. *Для любого гомоморфизма* φ *выполнено* $\varphi(\varepsilon) = \varepsilon$.

Доказательство. Пусть $\varphi(\varepsilon) = x$. Тогда получаем

$$x = \varphi(\varepsilon) = \varphi(\varepsilon\varepsilon) = \varphi(\varepsilon)\varphi(\varepsilon) = xx.$$

Но равенству $x = xx$ удовлетворяет только пустое слово. □

Как мы уже подчеркнули при определении, схема кодирования — бинарное отношение. В частном случае она может быть функцией.

Предложение 117. *Пусть Σ и Ω — два алфавита. Если схема кодирования C из Σ в Ω является функцией на Σ, то она задаёт гомоморфизм φ:*

$$\varphi(x) = y \iff y \in C[x].$$

Верно и обратное: каждый гомоморфизм может быть так задан.

Доказательство. Пусть C является определённой на Σ функцией. Пусть $x = a_1 \dots a_k$, где $a_i \in \Sigma$, $i = 1, \dots, k$. Так как C является функцией на Σ, то $C(a_i)$ является значением функции на a_i для $i = 1, \dots, k$, а $C[x]$ содержит в точности одно слово $C(a_1) \dots C(a_k)$. Таким образом однозначно определена функция φ:

$$\varphi(x) = \varphi(a_1 \dots a_k) = C(a_1) \dots C(a_k),$$

причём

$$\varphi(x) = y \iff y = C(a_1) \dots C(a_k) \iff y \in C[x]. \tag{26}$$

Обратно, пусть дан некоторый гомоморфизм φ. Тогда для любого слова $x = a_1 \dots a_k$, где $a_i \in \Sigma$, $i = 1, \dots, k$, из определения гомоморфизма получаем $\varphi(x) = \varphi(a_1) \dots \varphi(a_k)$. Если взять схему кодирования $C = \{(a, \varphi(a)) : a \in \Sigma\}$, то получим эквивалентность (26). □

С примерами гомоморфизмов мы уже сталкивались. Например, в параграфе 4.2 мы определяли операцию замены $(\Phi)^x_\Psi$ в формуле логики высказываний Φ пропозициональной переменной x на формулу Ψ (определение 32 на стр. 82). Нетрудно понять, что это пример гомоморфизма: $\varphi(x) = \Psi$, $\varphi(a) = a$ для всех других символов.

Замена предметной переменной x на y в формуле логики предикатов (параграф 7.4, определение 59 на стр. 148) гомоморфизмом не

является, так как заменяются только свободные вхождения y, а связанные остаются без изменений. При гомоморфизме все вхождения символов должны заменяться одинаковым образом.

Определение 137 (Гомоморфные образ и прообраз)**.** *Пусть дан произвольный гомоморфизм $\varphi : \Sigma^* \to \Omega^*$.*

Если $L \subseteq \Sigma^*$ — язык в алфавите Σ, то образом языка L при гомоморфизме φ называется язык

$$\varphi(L) = \{\varphi(w) : w \in L\},$$

состоящий из образов всех слов языка L.

Если $L \subseteq \Omega^*$ — язык в алфавите Ω, то (полным) прообразом языка L при гомоморфизме φ называется язык

$$\varphi^{-1}(L) = \{w \in \Sigma^* : \varphi(w) \in L\},$$

состоящий из всех таких слов в алфавите Σ, чьи образы при гомоморфизме φ попадают в L.

Из определения легко получаем

Следствие 118. *Пусть $\varphi : \Sigma^* \to \Omega^*$ — гомоморфизм, который задаётся схемой кодирования C из предыдущего предложения. Тогда*

1) *$\varphi(L) = C(L)$ для любого языка $L \subseteq \Sigma^*$;*
2) *$\varphi^{-1}(L) = C^{-1}(L)$ для любого языка $L \subseteq \Omega^*$.*

Напомним, что C^{-1} — обратное C отношение.

Это следствие тривиально влечёт следующее утверждение.

Теорема 119. *Класс автоматных языков замкнут относительно гомоморфных образов и прообразов.*

Доказательство. Очевидным образом вытекает из следствия 118 и теоремы 115 на стр. 333. □

Заметим, что для гомоморфизмов конструирование автомата $\mathfrak{N}$ из теоремы 115 на стр. 333 несколько упрощается: вместо переходов $(q, u) \vdash^*_{\mathfrak{M}} (p, \varepsilon)$ достаточно рассматривать команды $q, a \to p$, так как каждое слово u является одной буквой.

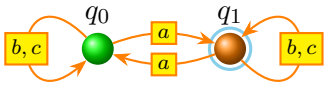


Рис. 104: Диаграмма автомата $\mathfrak{M}$ из примера 82

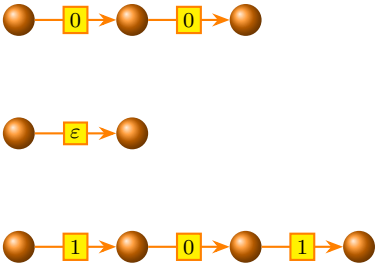


Рис. 105: Замены переходов в $\mathfrak{M}$ из примера 82

Пример 82. *Пусть гомоморфизм φ определён на символах алфавита $\Sigma = \{a, b, c\}$ следующим образом: $\varphi(a) = 00$, $\varphi(b) = \varepsilon$, $\varphi(c) = 101$, $\Omega = \{0, 1\}$. На рис. 104 изображён автомат $\mathfrak{M}$, распознающий язык*

$$L = \{w : \text{число букв } a \text{ в слове } w \text{ нечётно}\}.$$

Для построения нового автомата $\mathfrak{N}$ мы в $\mathfrak{M}$ поменяем все команды для букв a, b и c на фрагменты, показанные на рис. 105, соответственно. Тогда получим автомат $\mathfrak{N}$, изображённый на рис. 106 на следующей странице.

Отметим, что конструкция автомата $\mathfrak{N}$ в теореме 115 на стр. 333 универсальна, но может быть избыточной. В нашем случае, например, ε-переходы, очевидно, не нужны.

Для гомоморфных прообразов построение автомата $\mathfrak{N}$ из теоремы 115 на стр. 333 тоже проще, чем в общем случае. Поскольку все слова v_i имеют длину один, то это означает, что новых промежуточных состояний q^{ci} не потребуется.

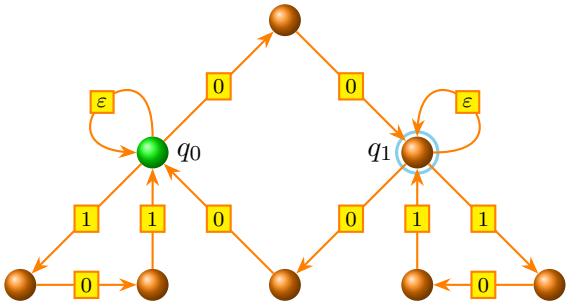


Рис. 106: Диаграмма автомата $\mathfrak{M}$ из примера 82

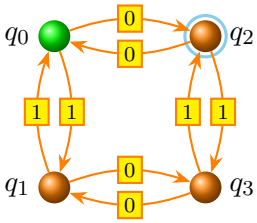


Рис. 107: Диаграмма автомата $\mathfrak{M}$ из примера 83

Пример 83. *Пусть алфавиты* Σ, Δ *и гомоморфизм* φ *определены как в предыдущем примере. Рассмотрим язык*

$$L = \{w : \text{число букв } 0 \text{ в слове } w \text{ нечётно, а число букв } 1 \text{ — чётно}\}.$$

На рис. 107 показана диаграмма ДКА $\mathfrak{M}$*, распознающего язык* L*.*

Обратное для φ *кодирование выглядит так:* $C = \{(00, a), (\varepsilon, b), (101, c)\}$*. Строим переходы автомата* $\mathfrak{N}$*. Для* ε *состояние, естественно, остаётся тем же самым, а для остальных слов:*

$$\begin{aligned}
&(q_0, 00) \vdash^*_{\mathfrak{N}} (q_0, \varepsilon); \quad (q_0, 101) \vdash^*_{\mathfrak{N}} (q_2, \varepsilon);\\
&(q_1, 00) \vdash^*_{\mathfrak{N}} (q_1, \varepsilon); \quad (q_1, 101) \vdash^*_{\mathfrak{N}} (q_3, \varepsilon);\\
&(q_2, 00) \vdash^*_{\mathfrak{N}} (q_2, \varepsilon); \quad (q_2, 101) \vdash^*_{\mathfrak{N}} (q_0, \varepsilon);\\
&(q_3, 00) \vdash^*_{\mathfrak{N}} (q_3, \varepsilon); \quad (q_3, 101) \vdash^*_{\mathfrak{N}} (q_1, \varepsilon).
\end{aligned}$$

Применяем к этому автомату конструкцию из теоремы 115 на стр. 333. Результат показан на рис. 108 на противоположной странице. Легко заметить,

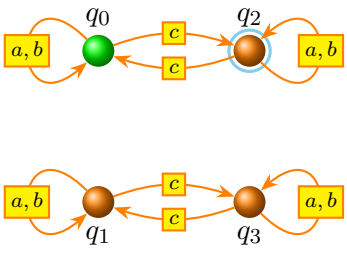


Рис. 108: Диаграмма автомата $\mathfrak{N}$ из примера 83

что в нём состояния q_2 и q_3 недостижимы из начального состояния q_0 и что этот автомат $\mathfrak{N}$ распознаёт язык

$$L_{\mathfrak{N}} = \varphi^{-1}(L) = \{u : \text{слово } u \text{ содержит нечётное число символов } c\}.$$

Интересным частным случаем гомоморфизма является проекция.

Определение 138 (Проекция). *Пусть $\Omega \subseteq \Sigma$. Проекцией слова $w \in \Sigma^*$ на подалфавит Ω называется слово $\pi_\Omega(w)$, полученное из w удалением всех символов a, не входящих в Ω.*

Проекцией языка L в алфавите Σ на подалфавит Ω называется язык

$$\pi_\Omega(L) = \{\pi_\Omega(w) : w \in L\}.$$

Предложение 120. *Проекция языка является его гомоморфным образом.*

Доказательство. Определим гомоморфизм $\pi : \Sigma^* \to \Omega^*$ так: $\pi(a) = a$ при $a \in \Omega$ и $\pi(a) = \varepsilon$ при $a \notin \Omega$. Тогда для любого языка L в алфавите Σ имеет место равенство $\pi_\Omega(L) = \pi(L)$. □

Отсюда сразу получаем замкнутость класса автоматных языков относительно проекции.

Следствие 121. *Для любых алфавитов Ω и Σ таких, что $\Omega \subseteq \Sigma$, и любого автоматного языка L в алфавите Σ проекция $\pi_\Omega(L)$ также является автоматным языком.*

Доказательство. Вытекает из доказанного предложения и теоремы 119 на стр. 338. □

Для проекции построение автомата $\mathfrak{N}$ по $\mathfrak{M}$ становится совсем простым: все команды вида $q, a \to p$ меняем на $q, \varepsilon \to p$ для $a \in \Sigma \setminus \Omega$.

Имеется ещё много операций, относительно которых замкнут класс автоматных языков. Некоторые из них приведены далее в разделе задач.

§ 17.2. Проблема однозначности декодирования

Гомоморфизмы часто появляются на практике как способы кодировки букв и других знаков в технических устройствах.

Например, одной из первых исторически была кодировка С. Морзе (часто её называют «азбукой Морзе»), предназначенная для представления букв английского алфавита и десятичных цифр в двоичном виде: при помощи короткого сигнала «точки» и длинного сигнала «тире». Поэтому азбука Морзе — это гомоморфизм φ_m из алфавита $\Sigma = \{a, b, \ldots, z, 0, 1, \ldots, 9\}$ в $\Omega = \{\cdot, -\}$, для которого в частности

$$\begin{aligned} \varphi_m(a) &= \text{«}\cdot - \text{»}; & \varphi_m(d) &= \text{«}- \cdot\cdot \text{»}; \\ \varphi_m(b) &= \text{«}- \cdot\cdot\cdot \text{»}; & \varphi_m(e) &= \text{«}\cdot \text{»}; \\ \varphi_m(c) &= \text{«}- \cdot - \cdot \text{»}; & \varphi_m(f) &= \text{«}\cdot\cdot - \cdot \text{»}. \end{aligned}$$

В современных вычислительных устройствах применяют более сложные способы кодирования, которые тоже носят двоичный характер. Например, самая популярная кодировка UTF-8 позволяет кодировать 2^{21} (то есть более двух миллионов) различных символов. Некоторые примеры кодов символов в UTF-8 приведены ниже:

$$\varphi_u(V) = 01010110;\ \varphi_u(!) = 00100001;\ \varphi_u(Я) = 1101000010101111;$$
$$\varphi_u(\subset) = 111000101000100110000010;$$
$$\varphi_u(\mathfrak{W}) = 11110000100111011001010010011010.$$

При использовании кодировок возникает следующая проблема: можно ли по коду слова однозначно восстановить само слово? Как правило, на практике имеют ценность лишь такие коды, где это можно сделать. Например, кодировка UTF-8 спроектирована именно так, чтобы по двоичному представлению строки можно было восстановить оригинал.

Азбука Морзе эти свойством не обладает. Скажем, слово «—···» является одновременно кодом слов b и de. Поэтому при использовании азбуки Морзе приходится вводить специальный разделитель между кодами отдельных символов исходного слова.

С математической точки зрения проблема однозначного декодирования заключается в следующем: для заданного гомоморфизма φ определить, можно ли по гомоморфному образу $\varphi(w)$ слова w однозначно восстановить само этого слово. Иными словами, мы должны ответить на вопрос: является ли φ разнозначной функцией?

Напомним, что любой гомоморфизм $\varphi : \Sigma^* \to \Omega^*$ полностью определяется своими значениями на буквах исходного алфавита Σ. Для краткости мы в дальнейшем будем обозначать эти значения с помощью φ_a: $\varphi_a = \varphi(a)$ для любого $a \in \Sigma$. Для произвольных слов $w \in \Sigma^*$ по прежнему будем использовать запись $\varphi(w)$.

Конечно, однозначно восстановить исходное слово нельзя, если $\varphi_a = v = \varphi_b$ для разных символов a и b. В этом случае образ v соответствует двум разным исходным словам, поэтому их восстановить по v невозможно. Другой явный случай неоднозначности — если для некоторой буквы a образ является пустым словом: $\varphi_a = \varepsilon$. В этом случае пустое слово ε будет кодом и любого слова вида a^n.

Поэтому нас будет интересовать только случай, когда неоднозначностей указанных двух видов нет. Для ответа на вопрос о разнозначности гомоморфизма введём следующее понятие.

Определение 139 (Кодовые слова, граф кодирования)**.** *Пусть гомоморфизм задан своими значениями на буквах исходного алфавита: φ_a, $a \in \Sigma$. Назовём слова φ_a, $a \in \Sigma$, к о д о в ы м и, обозначим их множество V_φ. С помощью V обозначим множество слов, ко-*

торые сами не являются кодовыми, но являются префиксом или суффиксом какого-то кодового слова.

Графом кодирования для гомоморфизма φ назовём размеченный ориентированный мультиграф $\mathfrak{G}(\varphi) = (V, E)$, в котором ребро (x, y), $x, y \in V$, существует и помечено набором слов $(w_1, \ldots, w_k)$, если и только если $w_1, \ldots, w_k \in V_\varphi$ и $xw_1 \ldots w_k y \in V_\varphi$. Петлю $(\varepsilon, \varepsilon)$, помеченную одним кодовым словом φ_a, в графе $\mathfrak{G}(\varphi)$ назовём тривиальной.

Говоря другим языком, если некоторое кодовое слово φ_a можно представить в виде $x\varphi_{a_1} \ldots \varphi_{a_k} y$, где φ_{a_j}, $j = 1, \ldots, k$ — тоже кодовые слова, а x, y — нет, то из x в y ведёт ребро, помеченное $(\varphi_{a_1}, \ldots, \varphi_{a_k})$. В частном случае k может быть равно нулю, тогда соответствующее ребро помечается пустым набором слов. Мы будем обозначать такой набор как и пустое слово с помощью ε.

Заметим, что если рассматривать только кодирования, в которых пустое слово не является кодовым, то пустое слово ε обязательно будет в числе вершин графа кодирования — оно является префиксом и суффиксом любого слова. Тривиальные петли в этом случае получаются из тривиальных разбиений кодовых слов: $\varphi_a = \varepsilon\varphi_a\varepsilon$.

Пример 84. *Рассмотрим гомоморфизм φ из алфавита $\Sigma = \{a, b, c, d, e\}$ в двоичный алфавит $\Omega = \{0, 1\}$ заданный следующим образом:*

$$\varphi_a = 01;\ \varphi_b = 100;\ \varphi_c = 011;\ \varphi_d = 000;\ \varphi_e = 11.$$

Здесь множество кодовых слов: $V_\varphi = \{01, 100, 011, 000, 11\}$. Множество всевозможных их префиксов и суффиксов:

$$\{\varepsilon, 0, 1, 01, 10, 00, 100, 011, 11, 000\}.$$

Вычитая V_φ из последнего множества, находим V:

$$V = \{\varepsilon, 0, 1, 10, 00\}.$$

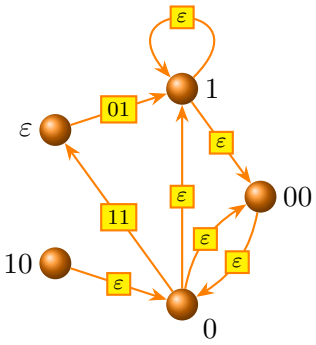


Рис. 109: Граф кодирования.

Далее для каждого кодового слова находим разбиения и соответствующие им рёбра:

Слово	*Вариант 1*		*Вариант 2*	
01	0|1	$(0,1)$		
100	1|00	$(1,00)$	10|0	$(10,0)$
011	0|11|ε	$(0,\varepsilon)$	ε|01|1	$(\varepsilon,1)$
000	00|0	$(00,0)$	0|00	$(0,00)$
11	1|1	$(1,1)$		

В результате получаем граф, показанный на рис. 109. Тривиальные петли мы в этом графе не показали.

Ответ на вопрос о разнозначности гомоморфизма даёт следующая теорема, доказанная А. А. Марковым.

Теорема 122 (Критерий Маркова)**.** *Пусть гомоморфизм φ задан своими значениями на символах исходного алфавита: $\varphi_a \in \Omega^*$, где $a \in \Sigma$ и все φ_a непусты и попарно различны. Тогда φ разнозначен, если и только если в графе кодирования $\mathfrak{G}(\varphi)$, кроме тривиальных петель, нет простых циклов, проходящих через вершину ε.*

Доказательство. Пусть гомоморфизм φ разнозначен. Будем рассуждать от противного: допустим, в графе $\mathfrak{G}(\varphi)$ есть цикл

$$\varepsilon = x_0 \xrightarrow{w_0} x_1 \xrightarrow{w_1} x_2 \xrightarrow{w_2} \cdots \xrightarrow{w_{k-1}} x_k \xrightarrow{w_k} x_{k+1} = x_0 = \varepsilon,$$

начинающийся и заканчивающийся в ε и не содержащий тривиальных петель. С помощью w_i мы обозначаем конкатенацию кодовых слов, которыми помечено ребро (x_i, x_{i+1}). Сразу напомним, что в этом случае слово $u_i = x_i w_i x_{i+1}$ является кодовым. Рассмотрим слово z, которое образуют вершины и рёбра этого пути:

$$z = x_0 w_0 x_1 w_1 x_2 w_2 \dots w_{k-1} x_k w_k.$$

Покажем, что оно является кодом двух разных слов.

Предположим, что k нечётно. Тогда слово z можно представить следующими двумя способами:

$$z = \underbrace{x_0 w_0 \overbrace{x_1}^{u_1} w_1}_{u_0} \underbrace{x_2 w_2 \overbrace{x_3}^{u_3} w_3 x_4}_{u_2} \dots \underbrace{x_{k-1} w_{k-1} \overbrace{x_k}^{u_k} w_k x_0}_{u_{k-1}}.$$

Отмеченные скобкой фрагменты u_i являются кодовыми словами, а промежуточные w_i — конкатенации кодовых слов. Мы получили два представления слова z в виде конкатенации кодовых слов:

$$z = u_0 w_1 u_2 w_3 \dots u_{k-1} w_k = w_0 u_1 w_2 u_3 \dots w_{k-1} u_k. \tag{27}$$

Покажем, что это два разных представления. В самом деле, x_0 пусто, а w_0 — это часть u_0. Если бы w_0 тоже было пустым, то x_1 совпало бы с кодовым словом u_0, а это противоречит тому, что x_i сами не являются кодовыми словами. Если бы w_0 совпало с u_0 и состояло бы из одного кодового слова, то мы бы получили тривиальную петлю. Следовательно, w_0 является конкатенацией одного или нескольких кодовых слов, из которых хотя бы одно не совпадает с u_0. В результате мы получили, что в представлениях (27) первые два кодовых слова различны. Но тогда гомоморфизм не является разнозначным, так как слово z можно получить из двух разных исходных слов.

Если k чётно, то аналогично получаем два разных способа:

$$z = \underbrace{x_0 w_0 \overbrace{x_1}^{u_1} w_1}_{u_0} \underbrace{x_2 w_2 \overbrace{x_3}^{u_3} w_3 x_4}_{u_2} \dots \overbrace{x_{k-1} w_{k-1} \underbrace{x_k}_{u_k} w_k x_0}^{u_{k-1}}.$$

и точно также устанавливаем, что слово z является гомоморфным образом двух разных исходных слов.

Докажем теорему в обратную сторону. Предположим, что гомоморфизм φ неразнозначен, то есть некоторое слово z можно получить из двух разных исходных слов.

Выберем самое короткое из таких слов z:

$$z = \overbrace{\underbrace{t_0}\underbrace{t_1}\underbrace{t_2}\underbrace{t_3}\underbrace{t_4}}\ldots\underbrace{t_{k-2}}\overbrace{\underbrace{t_{k-1}}\ \underbrace{t_k}}.$$

Также без ограничения общности считаем, что представленные разбиения на кодовые слова, имеют наибольшее количество частей.

Сначала рассмотрим случай, когда одно из разбиений, например, верхнее, состоит из одного фрагмента:

$$z = \overbrace{\underbrace{t_1}\ \ldots\ \underbrace{t_k}}.$$

Поскольку два разбиения различны, то нижнее из одного фрагмента состоять не может, то есть $k > 1$. Но тогда z — это кодовое слово, которое разбито так: $z = \varepsilon t_1 \ldots t_k \varepsilon$. Следовательно, в графе $\mathfrak{G}(\varphi)$ у вершины ε есть нетривиальная петля, помеченная $(t_1, \ldots, t_k)$. Эта петля и образует искомый простой цикл.

Теперь пусть оба разбиения, верхнее и нижнее, состоят из более чем одного фрагмента. Прежде всего отметим, что слова в верхнем и нижнем разбиении не могут иметь общих границ, кроме начала и конца всего слова z. В самом деле, если бы в середине слова у них была общая граница, то мы смогли бы разделить слово z по этой границе на два более коротких слова: $z = z_1 z_2$, хотя бы одно из них имело два два различных разбиения. А это противоречит тому, что слово z изначально выбрано самым коротким.

Для удобства верхний (нижний) фрагмент, который входит целиком в соответствующий нижний (верхний), будем называть к о р о т к и м, а в противном случае — д л и н н ы м.

Рассмотрим произвольный некрайний длинный фрагмент, например, верхнего разбиения. Поскольку границы этого фрагмента не

могут совпадать с границами никакого нижнего, то разбиение слова z около этого фрагмента имеет следующий вид, где x, x', y и y' непусты:

$$\overbrace{\underbrace{x'}\underbrace{x}\ \underbrace{v_1}\ \dots\ \underbrace{v_k}\ \underbrace{y}}\underbrace{y'}$$

Сразу следует отметить, что нижние фрагменты $x'x$ и yy', частично перекрывающие верхний длинный фрагмент, тоже являются длинными, так как не входят целиком ни в какой верхний, а нижние фрагменты $v_1, \dots, v_k$ — короткими.

Фрагмент x не является кодовым словом, так как иначе можно получить «верхнее» разбиение с бóльшим количеством частей:

$$\overbrace{x'}\ \overbrace{x}\ \ \overbrace{v_1}\ \dots\ \overbrace{v_k}\ \overbrace{y\,y'}.$$

Здесь левее x слово разбито как сверху, а правее — как снизу. Это противоречит тому, что выбранные разбиения имеют наибольшее число фрагментов. По той же самой причине кодовым словом не может быть y. Следовательно, в кодовом слове $xv_1 \dots v_k y$ префикс x и суффикс y не являются кодовыми словами, а средняя часть $v_1 \dots v_k$, очевидно, является конкатенацией кодовых слов. Это означает, что в графе кодирования имеется ребро (x, y), имеющее метку $(v_1, \dots, v_k)$. Точно так же рассматриваются некрайние длинные фрагменты снизу.

Рассмотрим теперь крайний левый длинный, например, верхний фрагмент:

$$\overbrace{\underbrace{v_1}\ \underbrace{v_2}\ \dots\ \underbrace{v_k}\ \underbrace{y}}\underbrace{y'}.$$

По тем же соображениям, что и раньше, y не может быть пустым или кодовым словом. Следовательно, в графе кодирования есть ребро (ε, y), помеченное $v_1 \dots v_k$. Точно также рассматриваются длинные крайние фрагменты снизу и справа.

Итак, мы получили, что два разбиения слова z имеют такой вид:

$$\overbrace{w_1\underbrace{x_1}w_2\overbrace{x_2}w_3\underbrace{x_3}w_4\overbrace{x_4}w_5x_5}\dots\underbrace{x_{n-1}w_n\overbrace{x_n}w_{n+1}}.$$

Здесь фигурными скобками выделены длинные фрагменты, слова w_i состоят из коротких фрагментов, а x_i кодовыми словами не являются. Изображённые разбиения соответствуют случаю, когда количество длинных фрагментов нечётно. В противном случае правый конец слова будет разбит противоположным образом.

Такое разбиение даёт цикл в графе кодирования:

$$\varepsilon \xrightarrow{w_1} x_1 \xrightarrow{w_2} x_2 \xrightarrow{w_3} x_3 \xrightarrow{w_4} x_4 \xrightarrow{w_5} x_5 \to \cdots \xrightarrow{w_{n-1}} x_{n-1} \xrightarrow{w_n} x_n \xrightarrow{w_{n+1}} \varepsilon,$$

где рёбра с нечётными номерами получаются из верхних длинных фрагментов, а с чётными — из нижних длинных. Поскольку все x_i непусты, то этот цикл не содержит петель около вершины ε, следовательно, его можно сократить до простого цикла без петель, проходящего через ε. □

Применим эту теорему к предыдущему примеру.

Пример 85. *В графе на рис. 109 на стр. 345 есть цикл*

$$\varepsilon \xrightarrow{01} 1 \xrightarrow{\varepsilon} 00 \xrightarrow{\varepsilon} 0 \xrightarrow{11} \varepsilon,$$

проходящий через вершину ε. Ему соответствует слово $z = 01100011$, которое можно разбить двумя способами на кодовые слова:

$$\overbrace{01}^{\varphi_a}\ \overbrace{100}^{\varphi_b}\ \overbrace{0}^{\varphi_c}\ \overbrace{11}^{\varphi_d}.$$
$$\underbrace{01}_{\varphi_c}\ \underbrace{10}_{\varphi_d}\ \underbrace{0}_{\varphi_e}\ \underbrace{11}_{\varphi_e}.$$

Таким образом, z является гомоморфным образом слов abc и cde, гомоморфизм не является разнозначным.

§ 17.3. Оптимальное кодирование Хаффмана

Разнозначность гомоморфизма ещё не означает его практическую ценность для задач кодирования слов.

Пример 86. *Рассмотрим такой гомоморфизм из алфавита $\Sigma = \{a, b, c\}$ в двоичный алфавит $\Omega = \{0, 1\}$: $\varphi_a = 1$, $\varphi_b = 10$, $\varphi_c = 00$. С помощью критерия Маркова нетрудно проверить, что такой гомоморфизм будет разнозначным, поэтому декодирование возможно. Граф изображён на рис. 110 на следующей странице.*

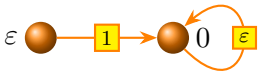


Рис. 110: Граф кодирования для примера 86.

Рассмотрим задачу восстановления исходного слова по коду 10^n. *Легко понять, что результат будет зависеть от того, чётно число* n *или нет. В первом случае получаем исходное слово* $ac^{n/2}$, *во втором —* $bc^{(n-1)/2}$.

Приведённый пример иллюстрирует вот какую проблему: декодирование может оказаться возможным только по прочтении всего слова целиком. В нашем случае, чтобы определить, сколько нулей содержится в слове, мы должны прочитать его до конца. Если такое слово является длинным, то нам придётся понести больши́е накладные расходы, связанные с его хранением, поскольку декодирование в режиме «реального времени», по мере поступления символов, невозможно.

Существует класс гомоморфизмов, которые этим недостатком заведомо не обладают.

Определение 140 (Беспрефиксные множество и гомоморфизм)**.** *Множество слов* W *называется б е с п р е ф и к с н ы м, если ни одно слово из* W *не является собственным префиксом другого.*

Гомоморфизм б е с п р е ф и к с н ы й, если кодовые слова непусты, попарно различны и образуют беспрефиксное множество.

Заметим, что если исходный алфавит Σ гомоморфизма содержит более одного символа, то из двух последних условий автоматически следует первое: пустое слово было бы префиксом второго кодового слова, отличного от пустого.

Беспрефиксные гомоморфизмы обязательно разнозначны.

Предложение 123. *Если гомоморфизм* $\varphi : \Sigma^* \to \Omega^*$ *является беспрефиксным, то он разнозначен.*

Доказательство. Докажем индукцией по длине кода $|\varphi(w)|$, что у $\varphi(w)$ не может быть никаких прообразов отличных от w.

Базис индукции: если $|\varphi(w)| = 0$, то $\varphi(w) = \varepsilon$. Если бы w не было пустым, например, содержало букву a, то мы бы получили $\varphi(a) = \varepsilon$, что невозможно для беспрефиксного гомоморфизма.

Индукционный шаг: пусть для кодов меньшей чем $|\varphi(w)|$ длины утверждение доказано и $|\varphi(w)| > 0$. Тогда $w \neq \varepsilon$, пусть a — первая буква w: $w = aw'$. По определению гомоморфизма $\varphi(w) = \varphi_a\varphi(w')$. Если предположить, что существует слово v вида bv', где $b \in \Sigma$ и $b \neq a$, для которого $\varphi(v) = \varphi(w)$, то мы получим

$$\varphi_a\varphi(w') = \varphi(w) = \varphi(v) = \varphi_b\varphi(v'),$$

то есть φ_a является префиксом φ_b или наоборот. В любом случае, гомоморфизм φ не будет беспрефиксным. Значит, такого слова v не существует. С другой стороны, пустым слово v быть тоже не может, так как образом пустого слова при гомоморфизме всегда является пустое, а у нас $|\varphi(w)| > 0$. Поэтому единственный возможный вариант: слово v начинается с a: $v = av'$ для какого-то v'. Тогда получаем

$$\varphi_a\varphi(w') = \varphi(w) = \varphi(v) = \varphi_a\varphi(v'),$$

откуда следует $\varphi(v') = \varphi(w')$. По индукционному предположению делаем вывод, что $v' = w'$ и $v = w$. □

Беспрефиксные коды в отличие от произвольных дают возможность декодирования в реальном времени: как только прочитан какой-то фрагмент, являющийся кодовым словом, можно сразу делать вывод о том, какая буква исходного слова ему соответствует, так как никакое продолжение этого фрагмента кодовым словом уже быть не может. Упомянутая в предыдущем параграфе кодировка UTF-8 является беспрефиксной.

Определение 141 (Оптимальный гомоморфизм)**.** *Назовём беспрефиксный гомоморфизм $\varphi : \Sigma^* \to \Omega^*$ оптимальным для слова $w \in \Sigma^*$, если $|\varphi(w)| \leqslant |\psi(w)|$ для каждого беспрефиксного гомоморфизма $\psi : \Sigma^* \to \Omega^*$.*

Таким образом, оптимальный для слова w гомоморфизм обеспечивает наименьшую возможную длину кода.

Существует эффективный алгоритм построения оптимального гомоморфизма для заданного слова. Мы рассмотрим его д в о и ч н ы й вариант, то есть когда кодовые слова состоят из нулей и единиц. Но сначала докажем несколько вспомогательных утверждений.

Лемма 124. *Пусть слово $w \in \Sigma^*$ содержит все символы Σ. Тогда для оптимального гомоморфизма $\varphi : \Sigma^* \to \Omega^*$ два самых длинных кодовых слова имеют равную длину.*

Доказательство. Возьмём φ_a и φ_b — самые длинные кодовые слова, причём $|\varphi_a| \leqslant |\varphi_b|$. Допустим, $|\varphi_a| < |\varphi_b|$. Пусть $\varphi_b = v_1 v_2$, где $|v_1| = |\varphi_a|$, то есть v_1 — префикс слова φ_b равный по длине слову φ_a. Построим новый гомоморфизм ψ, заменив в φ кодовое слово φ_b на v_1: $\psi_b = v_1$, $\psi_c = \varphi_c$ при $c \neq b$. Гомоморфизм ψ останется беспрефиксным: кодовое слово ψ_b теперь имеет наибольшую длину (вместе с ψ_a), поэтому оно не может быть префиксом никакого другого. Но тогда исходный код был неоптимальным: слово $\psi(w)$ короче $\varphi(w)$ за счёт сокращения длины кодового слова для b. □

Лемма 125. *Пусть слово $w \in \Sigma^*$ содержит все символы Σ. Тогда в оптимальном двоичном гомоморфизме $\varphi : \Sigma^* \to \{0,1\}^*$ существуют два самых длинных кодовых слова, которые имеют вид $u0$ и $u1$ соответственно для некоторого $u \in \{0,1\}^*$.*

Доказательство. Рассмотрим все кодовые слова максимальной длины: $v_{a_1}, \ldots, v_{a_k}$, согласно лемме 124, $k \geqslant 2$. Выделим в каждом из них последний символ: $\varphi_{a_i} = u_i x_i$, где $x_i \in \{0,1\}$. Из-за беспрефиксности эти u_i не совпадают с другими кодовыми словами.

Предположим, что все u_i попарно различны. Построим новый гомоморфизм ψ, заменив код каждого a_i на u_i: $\psi_{a_i} = u_i$, $\psi_c = \varphi_c$ для остальных $c \in \Sigma$. Теперь u_i будут кодовыми словами максимальной длины. Поэтому полученный гомоморфизм ψ снова будет беспрефиксным. Но за счёт сокращения длины кодовых слов мы получим $|\psi(w)| < |\varphi(w)|$, то есть φ — неоптимальный. Полученное противоречие показывает, что среди слов u_i есть равные.

Пусть например, $u_i = u_j = u$ при $i \neq j$. Получаем

$$ux_i = u_i x_i = \varphi_{a_i} \neq \varphi_{a_j} = u_j x_j = ux_j,$$

то есть $x_i \neq x_j$. Но тогда один из x_i и x_j равен единице, а второй — нулю, следовательно, φ_{a_i} и φ_{a_j} — искомые кодовые слова. □

Лемма 126. *Пусть каждый символ $a \in \Sigma$ встречается n_a раз в слове w, а φ — оптимальный для w гомоморфизм. Тогда из $n_a < n_b$ следует $|\varphi_a| \geqslant |\varphi_b|$.*

Доказательство. Если допустить $n_a < n_b$ и $|\varphi_a| < |\varphi_b|$, то можно построить новый гомоморфизм ψ, поменяв кодовые слова для a и b местами: $\psi_a = \varphi_b$, $\psi_b = \varphi_a$, $\psi_c = \varphi_c$ при $c \notin \{a, b\}$. Тогда получим

$$\begin{aligned}|\psi(w)| &= |\varphi(w)| + n_a(|\psi_a| - |\varphi_a|) + n_b(|\psi_b| - |\varphi_b|) = \\ &= |\varphi(w)| + n_a(|\varphi_b| - |\varphi_a|) + n_b(|\varphi_a| - |\varphi_b|) = \\ &= |\varphi(w)| + (n_a - n_b)(|\varphi_b| - |\varphi_a|) < |\varphi(w)|,\end{aligned}$$

то есть φ — неоптимальный. □

С помощью $(w)_b^a$ как и прежде будем обозначать результат замены в слове w всех вхождений буквы a на b.

Лемма 127. *Пусть каждый символ $a \in \Sigma$ встречается n_a раз в слове $w \in \Sigma^*$, a и b — два самых редких символа, с наименьшими n_a и n_b, $\varphi : \Sigma^* \to \{0,1\}^*$ — оптимальный двоичный гомоморфизм для слова w. Тогда существует оптимальный гомоморфизм $\psi : \Sigma^* \to \{0,1\}^*$ такой, что $\psi_a = u0$ и $\psi_b = u1$ для некоторого $u \in \{0,1\}^*$.*

Доказательство. Рассмотрим два варианта: $n_a < n_b$ и $n_a = n_b$.

Первый случай: $n_a < n_b$. Тогда $n_a < n_c$ для всех символов $c \neq a$, в силу леммы 126 получаем, что φ_a — самое длинное кодовое слово. Теперь рассмотрим все символы c, для которых $n_c = n_b$. Выберем из них символ d с самым длинным кодом φ_d. Так как $n_b = n_d$, то можно построить гомоморфизм θ, поменяв в φ кодовые слова для b и d местами (как в предыдущей лемме), при этом $|\theta(w)| = |\varphi(w)|$, так как обе буквы b и d встречаются одинаково часто в w.

Теперь θ_a и θ_b — два самых длинных кодовых слова гомоморфизма θ. По лемме 124 на стр. 352 они имеют одинаковую длину ℓ. Используя лемму 125 на стр. 352, среди всех кодовых слов длины ℓ можно найти пару слов вида $u0$ и $u1$. Снова поменяем в θ кодовые слова так, чтобы $u0$ и $u1$ стали кодами букв a и b соответственно, получим гомоморфизм ψ. Так как длины кодовых слов не изменились, то $|\psi(w)| = |\theta(w)| = |\varphi(w)|$, что и требуется.

Второй случай: $n_a = n_b$. Рассмотрим все символы c, для которых $n_c = n_a = n_b$. В силу предыдущей леммы кодовые слова φ_c имеют длину не меньшую остальных. Выберем из них пару самых длинных φ_{c_1} и φ_{c_2}, согласно лемме 124 на стр. 352 они имеют одинаковую длину. Построим новый гомоморфизм θ, поменяв в φ кодовые слова так, чтобы $\theta_a = \varphi_{c_1}$ и $\theta_b = \varphi_{c_2}$. Теперь θ_a и θ_b являются самыми длинными из кодовых слов. Дальше рассуждаем как в предыдущем случае. □

Лемма 128. *Пусть каждый символ $a \in \Sigma$ встречается n_a раз в слове $w \in \Sigma^*$, a и b — два самых редких символа, с наименьшими n_a и n_b, $\varphi : \Sigma^* \to \{0,1\}^*$ — оптимальный двоичный гомоморфизм для слова $(w)_b^a$. Тогда оптимальным для слова w двоичным гомоморфизмом будет такой: $\psi_a = \varphi_b 0$, $\psi_b = \varphi_b 1$, $\psi_c = \varphi_c$, если $c \notin \{a, b\}$.*

Доказательство. Допустим, ψ неоптимален, то есть существует оптимальный двоичный гомоморфизм $\eta : \Sigma^* \to \{0,1\}$ такой, что $|\eta(w)| < |\psi(w)|$. По предыдущей лемме найдётся оптимальный гомоморфизм ψ' такой, что $|\psi'(w)| = |\eta(w)|$ и $\psi'_a = u0$, $\psi'_b = u1$ для некоторого $u \in \{0,1\}^*$.

Для слова $(w)_b^a$ построим гомоморфизм φ': $\varphi'_b = u$, $\varphi'_c = \psi'_c$ при $c \neq b$. Он является беспрефиксным (задача 258 на стр. 367).

Поскольку выполнены соотношения $|\psi(w)| = \left|\varphi\left((w)_b^a\right)\right| + n_a + n_b$ и $\left|\varphi'\left((w)_b^a\right)\right| = |\psi'(w)| - n_a - n_b$, то получаем

$$\left|\varphi'\left((w)_b^a\right)\right| = |\psi'(w)| - n_a - n_b = |\eta(w)| - n_a - n_b < \\ < |\psi(w)| - n_a - n_b = \left|\varphi\left((w)_b^a\right)\right|,$$

то есть гомоморфизм φ не был оптимальным для $(w)_b^a$. Получили противоречие, следовательно, ψ должен быть оптимальным. □

Заметим, что для построения оптимального для w двоичного гомоморфизма нет нужды в самом этом слове, достаточно лишь знать, сколько раз та или иная буква в этом слове встречается. Удобнее всего это делать при помощи частот.

Определение 142 (Частота символа)**.** *В слове $w \in \Sigma^*$ частотой символа $a \in \Sigma$ называется отношение количества вхождений a в w к длине w.*

Очевидно, что если в словах w и w' частоты всех символов совпадают, то и оптимальные гомоморфизмы для них будут одни и те же. Ценность этого факта заключается в том, что в текстах на «естественных» языках (русском, английском и т. д.) частоты букв весьма стабильны. Следовательно, можно построить оптимальный гомоморфизм по этим частотам. Такой гомоморфизм «в среднем» будет оптимальным для текстов на соответствующем языке. В реальности, конечно, в разных текстах частоты будут слегка отличаться.

Используя понятие частоты, можно построить оптимальное кодирование для слов с заданными частотами методом, предложенным Д. Хаффманом (рис. 111 на следующей странице).

Теорема 129. *В результате вызова* ОптКод(F) *в массиве C будет построен оптимальный гомоморфизм для слов с частотами символов, заданными в массиве F.*

Доказательство. Используем индукцию по количеству символов в алфавите Σ.

Базис индукции: если $|\Sigma| = 2$, то $\Sigma = \{a, b\}$ для каких-то символов a и b. Наименьшая длина кодовых слов равна единице, и этого достаточно: код $\varphi(a) = 0$ и $\varphi(b) = 1$ является оптимальным.

Индукционный шаг: предположим, что для любого алфавита Ω, для которого $|\Omega| < |\Sigma|$, алгоритм ОптКод строит оптимальный гомоморфизм. Заметим, что слово $(w)_b^a$ имеет алфавит $\Sigma \setminus \{a\}$, частоты в нём всех букв кроме a и b не изменились, а частота b стала равна сумме частот a и b в слове w.

```
1: Алгоритм ОптКод(F)                    # Оптимальный гомоморфизм
2:     Вход: F[Σ] — массив частот символов алфавита Σ
3:     Выход: C[Σ] — массив кодов символов
4:     Если Σ = {a, b} то                # Для каких-то a и b
5:         C[a] ← 0
6:         C[b] ← 1
7:     Иначе
8:         выбрать в F наименьшие элементы F[a] и F[b]
9:         F[b] ← F[b] + F[a]
10:        удалить элемент F[a] из F
11:        C ← ОптКод (F)
12:        C[a] ← C[b] & 0
13:        C[b] ← C[b] & 1
14:    Конец Если
15: Конец Алгоритм
```

Рис. 111: Алгоритм Хаффмана

Значит, если массив F содержал частоты букв в слове w, то после выполнения строк 8–9 он будет содержать частоты букв в слове $(w)_b^a$. Но поскольку размер алфавита стал меньше, то по индукционному предположению в результате вызова ОптКод (F) мы получим в C некоторый оптимальный гомоморфизм φ для слова $(w)_b^a$. Согласно лемме 128 на стр. 354 оптимальный для слова w гомоморфизм ψ можно получить из оптимального для слова $(w)_b^a$ гомоморфизма φ, приписав к φ_b единицу и ноль соответственно для b и a, что и происходит в строках 11–12. □

Пример 87. *Рассмотрим процесс построения оптимального гомоморфизма для слова «ВОДОВОРОТ». Исходная таблица частот F для него имеет следующий вид (частоты во всех таблицах упорядочены по убыванию):*

О	*В*	*Д*	*Р*	*Т*
$4/9$	$2/9$	$1/9$	$1/9$	$1/9$

При первом вызове алгоритма две самые редкие буквы «Р» и «Т» сольются в одну (обозначена «Р'») и мы получим

О	*В*	*Р'*	*Д*
$4/9$	$2/9$	$2/9$	$1/9$

Теперь то же самое произойдёт с буквами «Д» и «Р'», получится «Д':

О	*Д'*	*В*
$4/9$	$3/9$	$2/9$

Наконец, будут слиты «В» и «Д'» в «В':

В'	*О*
$5/9$	$4/9$

На этой стадии будет построен код для двухбуквенного алфавита

В'	*О*
1	0

Теперь процесс пойдёт в обратном направлении: из кода «В'» будут получены коды для «В» и «Д':

О	*Д'*	*В*
0	10	11

Далее из кода «Д'» получим коды для «Д» и «Р':

О	*В*	*Р'*	*Д*
0	11	100	101

Наконец, из кода «Р'» получим коды для «Р» и «Т»:

О	*В*	*Д*	*Р*	*Т*
0	11	101	1001	1000

Таким образом, оптимальный код слова «ВОДОВОРОТ» имеет вид:

$$\underbrace{11}_{В}\ \underbrace{0}_{О}\ \underbrace{101}_{Д}\ \underbrace{0}_{О}\ \underbrace{11}_{В}\ \underbrace{0}_{О}\ \underbrace{1001}_{Р}\ \underbrace{0}_{О}\ \underbrace{1000}_{Т}.$$

Работу алгоритма можно показать на ориентированном дереве. Вначале движемся от листьев к корню «объединяя» символы (рис. 112 на следующей странице), а затем — спускаемся от корня к листьям и строим коды символов (рис. 113 на следующей странице).

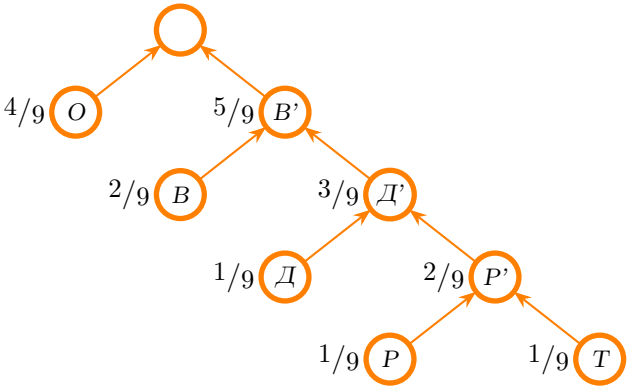


Рис. 112: Построение дерева.

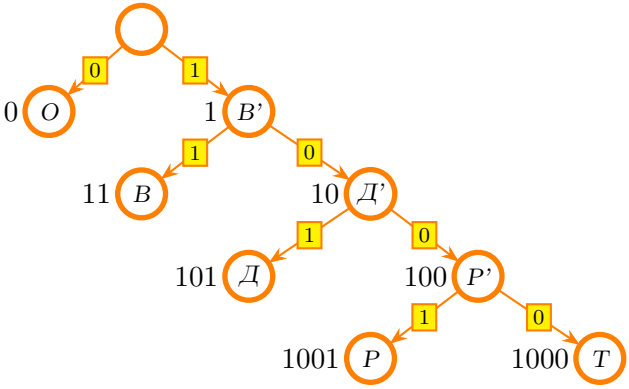


Рис. 113: Построение кодов символов.

§ 17.4. Лемма о разрастании для автоматных языков

До сих пор мы встречались лишь с автоматными языками и накопили достаточно много средств для доказательства того, что некоторый язык является автоматным. Для этого, например, достаточно построить для него регулярное выражение или получить его

с помощью различных рассмотренных выше операций из заведомо автоматных языков. В этом разделе мы установим некоторое необходимое условие, которому удовлетворяют все автоматные языки. После этого, проверив, что некоторый язык этому условию не удовлетворяет, можно заключить, что он не является автоматным.

Теорема 130 (Лемма о разрастании)**.** *Для всякого бесконечного автоматного языка L существует положительная натуральная константа n, для которой имеет место следующее.*

Если любое слово $w \in L$ разбить на три части $w = \alpha u \beta$ так, что длина средней части u не меньше n, то найдутся слова x, y и z, для которых выполняются условия:

1) $u = xyz$;
2) $1 \leqslant |y| \leqslant n$;
3) *для каждого натурального m слово $w_m = \alpha x y^m z \beta$ принадлежит языку L.*

Доказательство. Так как язык L автоматный, то существует ДКА $\mathfrak{M} = (\Sigma, Q, P, q_0, F)$, распознающий L. Положим $n = |Q|$.

Рассмотрим любое слово $w \in L$, которое имеет вид $\alpha u \beta$, $|u| \geqslant n$. Рассмотрим путь

$$p = (q_0 = q_{i_0}, q_{i_1}, \ldots, q_{i_f} \in F)$$

в диаграмме автомата $\mathfrak{M}$, несущий слово w. Он естественным образом разбивается на три части в соответствии с разбиением слова w:

$$p = (\underbrace{q_0 = q_{i_0}, q_{i_1}, \ldots,}_{p_\alpha} \underbrace{q_{i_t}, \ldots, q_{i_s},}_{p_u} \underbrace{\ldots, q_{i_f} \in F}_{p_\beta}),$$

Длина средней части p_u пути равна длине слова u. Она содержит не менее n рёбер и, следовательно, не менее $n+1$ вершин графа, то есть не менее чем $n+1$ состояние. Так как всего состояний у автомата n, то делаем вывод, что в пути p_u какие-то состояния повторяются,

имеется цикл. Пусть q_{i_k} — первое состояние в p_u, которое совпадает с одним из предыдущих q_{i_ℓ}:

$$p_u = (q_{i_t}, \ldots, q_{i_\ell}, \ldots, q_{i_k} = q_{i_\ell}, \ldots, q_{i_s}),$$

причём $\ell < k$, а в части, предшествующей q_{i_k}, повторов нет. Пусть путь $(q_{i_t}, \ldots, q_{i_\ell})$ несёт слово x, путь $(q_{i_\ell}, \ldots, q_{i_k})$ — слово y, путь $(q_{i_k}, \ldots, q_{i_s})$ — слово z. Тогда $u = xyz$, что доказывает 1).

Из $\ell < k$ следует, что $1 \leqslant |y|$. Так как в $(q_{i_t}, \ldots, q_{i_{k-1}})$ нет повторов, то $k - t \leqslant n$, следовательно, $k - \ell \leqslant n$ и длина y не превосходит n. Итак, $1 \leqslant |y| \leqslant n$, то есть выполнено 2).

Возьмём произвольное натуральное m и рассмотрим работу автомата $\mathfrak{M}$ на слове $w_m = \alpha x y^m z \beta$.

Так как автомат детерминированный, то прочитав начальную часть αx, он окажется в том же самом состоянии q_{i_ℓ}, что при работе на слове $w = \alpha xyz\beta$.

Далее, из состояния q_{i_ℓ}, прочитав фрагмент y, автомат вернётся в тоже самое состояние $q_{i_k} = q_{i_\ell}$, поэтому, прочитав y^m, он по-прежнему будет находиться в состоянии $q_{i_k} = q_{i_\ell}$.

Наконец, из состояния q_{i_k}, прочитав заключительный фрагмент $z\beta$, автомат переходит в заключительное состояние q_{i_f}, то есть принимает слово.

Поскольку автомат $\mathfrak{M}$ принимает слово w_m, то $w_m \in L$. Это доказывает 3). □

Содержательно, эта теорема утверждает, что во всяком достаточно длинном фрагменте слова из автоматного языка имеется непустая часть, которую можно вырезать или повторить сколько угодно раз, а само слово при этом останется в языке.

Как, используя лемму о разрастании, доказать, что некоторый язык L не является автоматным? Это можно сделать, используя схему доказательства «от противного»:

(a) Предположим, что L — автоматный язык. Тогда для него имеется константа n из утверждения леммы о разрастании.

(б) Определим по n некоторое «специальное» слово $w \in L$ и его фрагмент u длины не меньшей, чем n.

(в) Докажем, что для каждого разбиения $u = xyz$, удовлетворяющего условиям 1) и 2) леммы, найдётся такое $m \geqslant 0$, что слово $w_m = \alpha x y^m z \beta$ не принадлежит L.

(г) Так как последнее противоречит пункту 3) леммы о разрастании, то делаем вывод, что наше допущение неверно и L — неавтоматный язык.

Следует особо обратить внимание на следующее. Лемма о разрастании утверждает, что части x, y, z существуют, но ничего не говорит о том, какой именно вид они имеют. Поэтому в пункте (в) мы должны рассмотреть все возможные разбиения слова u, чтобы сделать вывод о противоречии с пунктом 3) леммы.

Разумеется, в этой схеме самым нетривиальным является выбор «специального» слова $w \in L$ и его фрагмента u в пункте (б). Что касается подбора такого $m \geqslant 0$, для которого $w_m \notin L$, то во многих случаях (но не всегда, конечно) достаточно рассмотреть $m = 0$ или $m = 2$, то есть убрать y совсем или вставить его ещё один раз.

§ 17.5. Неавтоматные языки

Приведём несколько примеров применения леммы о разрастании.

Предложение 131. *Язык* $L_1 = \{0^i 1^i : i \geqslant 1\}$ *не является автоматным.*

Доказательство. Предположим, что L автоматный. Тогда для него имеется константа n из леммы 130 на стр. 359. Рассмотрим следующее «специальное» слово $w = 0^n 1^n$. Очевидно, что $w \in L_1$. Разобьём его на три части: $\alpha = \varepsilon$, $u = 0^n$, $\beta = 1^n$.

По лемме о разрастании средняя часть u разбивается так: $u = xyz$, причём $1 \leqslant |y| \leqslant n$. Так как слово u состоит только из символов 0, то все три части x, y, z тоже состоят только из символов 0. Тогда $x = 0^p$, $y = 0^q$, $z = 0^{n-p-q}$ для каких-то натуральных чисел p и q, где $1 \leqslant q \leqslant n$.

Построим слово w_m:

$$w_m = \alpha x y^m z \beta = \varepsilon 0^p 0^{mq} 0^{n-p-q} 1^n = 0^{n+q(m-1)} 1^n.$$

При $m = 0$ получаем, что количество нулей в слове w_0 равно $n-q < n$, а количество единиц равно n. Такое слово языку L_1 не принадлежит, а по лемме о разрастании — должно.

Полученное противоречие показывает, что наше предположение неверно, язык L_1 неавтоматный. □

Предложение 132. *Язык* СКОБ *правильных скобочных последовательностей в алфавите* $\{(,)\}$ *не является автоматным.*

Доказательство. Схема доказательства та же. В качестве специального слова выберем слово $w = (^n)^n$, оно, очевидно, принадлежит СКОБ. Полагаем $\alpha = \varepsilon$, $u = (^n, \beta =)^n$. Тогда для всякого разбиения $u = xyz$ получаем $x = (^p$, $y = (^q$, $z = (^{n-p-q}$, где $1 \leqslant q \leqslant n$. И, как и в предыдущем утверждении, слово $w_0 = \alpha x z \beta = (^{n-q})^n$ не принадлежит языку СКОБ, что противоречит условию 3) теоремы. Следовательно, язык СКОБ неавтоматный. □

Предложение 133. *Язык* $L_2 = \{w = 0^i 1^j : i \leqslant 2j + 1\}$ *не является автоматным.*

Доказательство. Здесь, предположив, что L_2 автоматный язык и зафиксировав константу n из теоремы 130 на стр. 359, рассмотрим слово $w = 0^{2n+1}1^n \in L_2$. Разобьём его на три части: $\alpha = 0^{2n+1}$, $u = 1^n$, $\beta = \varepsilon$. Если $u = xyz$, то как и раньше получаем $x = 1^p$, $y = 1^q$, $z = 1^{n-p-q}$, где $1 \leqslant q \leqslant n$. Рассмотрим слово $w_0 = \alpha x z \beta = 0^{2n+1}1^{n-q}$. Но $2n + 1 > 2(n - q) + 1$, поэтому $w_0 \notin L_2$ и язык L_2 не является автоматным. □

Предложение 134. *Язык «квадратов» в алфавите* $\{a\}$*:*

$$L_3 = \{a^{i^2} : i \in \omega\}$$

не является автоматным.

Доказательство. Здесь, предположив, что L_3 автоматный язык и зафиксировав константу n из теоремы 130 на стр. 359, рассмотрим

слово $w = a^{n^2}$ и его разбиение: $\alpha = \varepsilon$, $u = a^{n^2}$, $\beta = \varepsilon$. Так как $n^2 \geqslant n$, то можно применить лемму о разрастании: пусть $u = xyz$. Тогда $x = a^p$, $y = a^q$, $z = a^{n^2-p-q}$, причём $1 \leqslant q \leqslant n$. Получаем,

$$w_m = \alpha x y^m z \beta = \varepsilon a^p a^{mq} a^{n^2-p-q} \varepsilon = a^{n^2+q(m-1)}.$$

Согласно лемме, $w_m \in L_3$, это значит, что число $n^2 + q(m-1)$ является квадратом. Заметим, то между числами n^2 и $(n+1)^2$ квадратов нет, поэтому должно быть выполнено одно из двух неравенств $n^2 + q(m-1) \leqslant n^2$ или $n^2 + q(m-1) \geqslant (n+1)^2$. При $m = 2$ получаем $n^2 + q \geqslant n^2 + 1 > n^2$, то есть первое не выполнено. С другой стороны,

$$n^2 + q \leqslant n^2 + n < n^2 + 2n + 1 = (n+1)^2,$$

то есть и второе не выполнено.

Полученное противоречие показывать ложность нашего предположения, то есть язык «квадратов» L_3 не является автоматным. □

Рассмотрим теперь пример, когда подбор слова w и числа m менее очевиден.

Предложение 135. *Язык «простых чисел» в алфавите* $\{a\}$*:*

$$L_{pr} = \{a^k : k \text{ — простое число}\}$$

не является автоматным.

Доказательство. Предположим, что L_{pr} — автоматный язык, и зафиксируем для него константу n из леммы о разрастании. Поскольку существуют сколь угодно больши́е простые числа, то можно выбрать такое простое k, что $k \geqslant n$. Рассмотрим слово $w = a^k$.

Как и в предыдущем случае возьмём разбиение $\alpha = \varepsilon$, $u = a^k$, $\beta = \varepsilon$, которое подходит под условия леммы, и мы сможем найти x, y, z. Тогда $x = a^p$, $y = a^q$ и $z = a^{k-p-q}$, $1 \leqslant q \leqslant n$. Строим w_m:

$$w_m = \alpha x y^m z \beta = \varepsilon a^p a^{mq} a^{k-p-q} \varepsilon = a^{k+q(m-1)}.$$

Возьмём $m = k+1$, тогда получим $w_{k+1} = a^{k+q(k+1-1)} = a^{k(q+1)}$. Так как k — простое, то $k \geqslant 2$, с другой стороны, $q + 1 \geqslant 2$. Это означает,

что число $k(q+1)$ является произведением двух чисел, ни одно из которых единице не равно, следовательно, $k(q+1)$ — составное. Тогда слово $w_{k+1} = a^{k(q+1)}$ не принадлежит L_{pr}, что противоречит лемме о разрастании.

Из полученного противоречия делаем вывод, что предположение об автоматности L_{pr} неверно. Заметим, что в этом примере m выбирается для каждого n по-своему. □

Ещё один приём доказательства неавтоматности языка L заключается в применении свойств замкнутости автоматных языков. Если O — какая-то операция, относительно которой класс автоматных языков замкнут, и про язык $L = O(L_1, \ldots, L_n)$ доказано, что он неавтоматный, то хотя бы один из языков $L_1, \ldots, L_n$ тоже не должен быть автоматным.

Предложение 136. *Язык $L_4 = \{0^i 1^j : i \neq j\}$ неавтоматный.*

Доказательство. Пусть $L_5 = \{0^i 1^j : i, j \in \omega\}$. Очевидно, что язык L_5 автоматный. Нетрудно заметить, что $L_1 = \bar{L}_4 \cap L_5$, где L_1 — язык из предложения 131 на стр. 361. Так как мы установили, что L_1 неавтоматный, то и L_4 не является автоматным. □

Являются ли условия теоремы 130 на стр. 359 достаточными для того, чтобы язык оказался автоматным? Следующее предложение показывает, что ответ на этот вопрос отрицателен.

Предложение 137. *Существуют неавтоматные языки, для которых выполняется лемма о разрастании.*

Доказательство. Зафиксируем алфавит $\Sigma = \{a, b, c\}$ и возьмём два языка:

$$L_6 = \{(abc)^{i^2} : i \in \omega\}$$

и язык L_7, заданный регулярным выражением

$$(a+b+c)^*(aa+bb+cc+aba+aca+bab+bcb+cac+cbc)(a+b+c)^*.$$

Язык L_7, как несложно видеть, содержит такие слова, в которых есть два вхождения одной и той же буквы, разделённые не более чем одним символом. Рассмотрим язык $L_8 = L_6 \cup L_7$ и покажем, что он удовлетворяет лемме о разрастании.

В качестве константы n выберем 4. Пусть $w \in L_8$, а u — любой фрагмент слова w длины 4 (во фрагментах большей длины можно взять четыре первых символа). Тогда в этом фрагменте есть хотя бы два разных вхождения одной и той же буквы. Предположим для определённости, что два раза встречается a. Переберём все возможные варианты вхождений, укажем x, y, z в каждом случае и покажем, что условие 3) леммы о разрастании будет выполнено (с помощью p и q обозначены два оставшихся символа из x, они могут совпадать между собой и с a, для нас это несущественно):

- $u = pqaa$, тогда $x = p$, $y = q$, $z = aa$; слово w_m обязательно будет содержать aa;
- $u = paaq$, тогда $x = paa$, $y = q$, $z = \varepsilon$; слово w_m обязательно будет содержать aa;
- $u = paqa$, тогда $x = \varepsilon$, $y = p$, $z = aqa$; слово w_m обязательно будет содержать aqa;
- $u = aapq$, тогда $x = aa$, $y = p$, $z = q$; слово w_m обязательно будет содержать aa;
- $u = apaq$, тогда $x = apa$, $y = q$, $z = \varepsilon$; слово w_m обязательно будет содержать apa;
- $u = apqa$, тогда $x = a$, $y = pq$, $z = a$; для $m = 0$ слово w_m содержит aa; для $m = 1$ слово w_m совпадает с исходным и принадлежит L_8; для $m \geqslant 2$ слово w_m обязательно будет содержать $pqpq$, то есть две буквы p рядом (если $p = q$) или через одну.

Итак, мы убедились, что любой фрагмент u длины 4 (и тем более, длиннее) допускает разбиение $u = xyz$, удовлетворяющее лемме о разрастании.

Покажем, что язык L_8 тем не менее неавтоматный. Язык L_7 задан регулярным выражением, он автоматный. Следовательно, автоматным будет язык $\bar{L}_7$. Так как L_6 и L_7 не пересекаются, то $L_6 = L_8 \cap \bar{L}_7$. Если к языку L_6 применить схему кодирования $C = \{(abc, a)\}$, то мы получим язык L_3 из предложения 134 на стр. 362. Последний не является автоматным, значит, не является автоматным язык L_6 и, следовательно, L_8. □

Задачи

249. Определим операцию цилиндрификации, которая обратна операции проекции. Для любых алфавитов Ω и Σ таких, что $\Omega \subset \Sigma$, и любого языка $L \subseteq \Omega^*$ определим его цилиндрификацию до алфавита Σ как язык

$$Z_\Sigma(L) = \{w \in \Sigma^* : \pi_\Omega(w) \in L\}.$$

Показать, что для автоматного языка L язык $Z_\Sigma(L)$ также является автоматным языком. Предложить процедуру перестройки автомата, распознающего L, в автомат, распознающий $Z_\Sigma(L)$.

250. Обращением слова $w = a_1 a_2 \dots a_k$, $a_i \in \Sigma$ при $i = 1, \dots, k$, называется слово $w^{-1} = a_k \dots a_2 a_1$. Показать, что для автоматного языка L его обращение — язык $L^{-1} = \{w^{-1} : w \in L\}$ — также является автоматным.

251. Пусть L — автоматный язык в алфавите Σ. Доказать, что автоматными являются и следующие языки:

(а) ПРЕФ$(L) = \{w :$ существует такое слово $x \in \Sigma^*$, что $wx \in L\}$;

(б) СУФ$(L) = \{w :$ существует такое слово $x \in \Sigma^*$, что $xw \in L\}$;

(в) ИНФ$(L) = \{w :$ существуют такие слова $x, y \in \Sigma^*$, что $xwy \in L\}$;

(г) МАХ$(L) = \{w \in L : wx \notin L$ для всякого непустого слова $x\}$.

252. Пусть L — автоматный язык в алфавите $\Sigma = \{a_1, \dots, a_m\}$, а $L_1, \dots, L_m$ — это автоматные языки в алфавите Δ. Доказать, что автоматным является и язык ЗАМ(L), полученный из слов L заменой каждой буквы a_i на некоторое слово из L_i. Таким образом,

$$\text{ЗАМ}(L) = \{w_1 w_2 \dots w_n : w_1 \in L_{i_1}, \dots, w_n \in L_{i_n} \text{ и существует } a_{i_1} a_{i_2} \dots a_{i_n} \in L\}.$$

253. Построить граф кодирования для первых шести кодовых слов азбуки Морзе.

254. С помощью критерия Маркова проверить, будет ли следующий гомоморфизм разнозначным: $\varphi(a) = 01$, $\varphi(b) = 100$, $\varphi(c) = 0110$, $\varphi(d) = 11$, $\varphi(e) = 0100$, $\varphi(f) = 101$. Если нет, то найти слово, которое нельзя однозначно декодировать.

255. Доказать, что беспрефиксные гомоморфизмы разнозначны с помощью критерия Маркова.

256. Построить с помощью алгоритма Хаффмана оптимальные двоичные кодирования для слов (а) «каракатица», (б) «параллелепипед», (в) «телеаппаратура», (г) «индивидуальность», (д) «перераспределение». Определить длину получившихся кодов слов. Вычислить, насколько оптимальное кодирование даёт результат короче, чем двоичное равномерное, то есть когда коды всех символов имеют одну и ту же длину.

257. Построить конечный преобразователь, выполняющий декодирование из примера 87 на стр. 356.

258. Доказать, что гомоморфизм φ', построенный в доказательстве леммы 128 на стр. 354, является беспрефиксным.

259. Доказать, что следующие языки в алфавите $\Sigma = \{a, b, c\}$ не являются автоматными:

(а) множество всех слов, в которых букв a на 3 больше, чем букв b;

(б) $L = \{a^n c b^m : m > 3n\}$;

(в) $L = \{wcw^{-1} : w = a^2 b^n a \text{ для некоторого } n > 0\}$;

(г) $L = \{w : |w| = 2^n \text{ для некоторого натурального } n\}$;

(д) $L = \{wc^{|w|} : w \in \{a, b\}^*,\ |w| \text{ — длина слова } w\}$.

260. Пусть $\boldsymbol{V}$ — (конечное) множество переменных, $\boldsymbol{L} = \{\text{«(»}, \text{«)»}, \text{«}\lambda\text{»}\}$. Тогда λ-в ы р а ж е н и е — это слово в алфавите $\boldsymbol{V} \cup \boldsymbol{L}$, определяемое индуктивно: либо переменная $x \in \boldsymbol{V}$, либо «$\lambda x\, e_1$», либо «$(e_1\, e_2)$», где $x \in \boldsymbol{V}$, e_1, e_2 — λ-выражения. Например, слова «$\lambda x\, x$», «$\lambda x\, (x\, x)$», «$\lambda x\, \lambda x\, (\lambda x\, (x\, x)\, \lambda x\, (x\, x))$» — это λ-выражения, а слова «$(x\, \lambda x)$», «$\lambda x\, (\lambda x)$» и «$\lambda x\, ((x\, x)$» — λ-выражениями не являются. Доказать, что множество λ-выражений в алфавите $\boldsymbol{V} \cup \boldsymbol{L}$ не является автоматным.

261. Выше в задаче 231 на стр. 309 строился автомат, который проверял правильность сложения двоичных чисел. Доказать, что для операции умножения двоичных чисел такого автомата не существует. Точнее, следующий язык в алфавите трёхэтажных символов не является автоматным:

$$\left\{ \left[\!\!\begin{smallmatrix} x_1 \\ y_1 \\ z_1 \end{smallmatrix}\!\!\right] \dots \left[\!\!\begin{smallmatrix} x_n \\ y_n \\ z_n \end{smallmatrix}\!\!\right] : x_i, y_i, z_i \in \{0, 1\} \text{ для } i = 1, \dots, n \text{ и } z_n \dots z_1 \text{ — это произведение двоичных чисел } x_n \dots x_1 \text{ и } y_n \dots y_1 \right\}.$$

262. Доказать, что следующий язык в алфавите $\Sigma = \{a, b, c\}$ не является автоматным:

$$L = \{w \in \Sigma^* : \text{количества букв } a \text{ и } b \text{ в слове } w \text{ различны}\}.$$

Глава 18

Алгоритмы и программы

Краткое содержание: алгоритмы и модели вычислений, языки программирования, программы с метками: синтаксис и семантика, вычисление арифметических функций.

Ключевые слова: алгоритм, программа с метками, переменная, присваивание, ветвление, конфигурация программы, операционная семантика программы, вычисляемая программой функция, вычислимая функция, арифметическая функция.

§ 18.1. Что такое алгоритм?

Первоначальной целью теории алгоритмов является классификация всех задач на алгоритмически разрешимые и неразрешимые, то есть на те, для которых существуют решающие их алгоритмы, и те, для которых таких алгоритмов нет.

Неформально под а л г о р и т м о м $\mathfrak{M}$ можно понимать выраженный в некотором языке

> набор правил (предписание, рецепт, способ), позволяющий применить к исходным (входным) данным x из некоторого множества до-

пустимых данных X последовательность дискретных действий (операций, команд), приводящую к определённому результату — выходным данным $y = \mathfrak{M}(x)$ из некоторого множества Y.

В этом случае говорят, что $\mathfrak{M}$ вычисляет функцию $F : X \to Y$, для которой $F(x) = \mathfrak{M}(x)$ для всех $x \in X$. Это нестрогое определение вполне подходит в тех случаях, когда для некоторой функции нам предъявляется «объект», называемый алгоритмом её вычисления (например, алгоритм Евклида для вычисления наибольшего общего делителя двух целых чисел), и можно легко проверить, позволяет ли он действительно вычислить требуемую функцию. Однако оно совершенно не годится для доказательства того, что для заданной функции никакого алгоритма нет.

Начиная с тридцатых годов XX века был предпринят ряд исследований для формализации понятия алгоритма. Перечислим некоторые из предложенных разными авторами в разное время формальных моделей: машины Тьюринга (А. Тьюринг, Э. Пост), частично рекурсивные функции (К. Гёдель, С. Клини), λ-исчисление (А. Чёрч, С. Клини), клеточные автоматы (Дж. фон Нейман), нормальные алгорифмы (А. А. Марков), счётчиковые машины (М. Минский), автоматы на графах (А. Н. Колмогоров, В. А. Успенский, Я. М. Барздинь) и др. Заложенные в них идеи в значительной степени повлияли затем на архитектуру и языки программирования реальных компьютеров. Например, на базе λ-исчисления построен широко применяемый в задачах искусственного интеллекта язык ЛИСП, из нормальных алгорифмов Маркова произошёл хорошо подходящий для текстовой обработки язык РЕФАЛ. Каждый из многочисленных языков программирования также задаёт некоторую формальную модель алгоритмов. В этой главе мы рассмотрим один из простейших таких языков — простые программы с метками, а в следующих — сравним их с двумя другими моделями алгоритмов: описаниями частично рекурсивных функций и машинами Тьюринга.

Хотя алгоритмы в разных прикладных областях имеют дело с дискретными объектами различных видов: целыми и рациональными числами, строками, формулами, разного рода выражениями, графами, матрицами, таблицами, точечными изображениями и многими другими, — мы в этой части курса будем рассматривать только задачи вычисления функций на множестве натуральных чисел (такие функции часто называют а р и ф м е т и ч е с к и м и) и функций на множестве слов (такие функции называют с л о в а р н ы м и). Дело в том, что для любого естественного множества дискретных объектов (в частности, для всех перечисленных выше) имеется простое кодирование его элементов натуральными числами и словами. Поэтому задачи вычисления функций на произвольных множествах превращаются в задачи вычисления арифметических или словарных функций.

Напомним, что через ω обозначается множество натуральных чисел. Частичная функция на A — это функция, для которой $\operatorname{dom} f \subseteq A$. Чтобы указать, что f не определена на некотором наборе аргументов $a_1, \ldots, a_n$, будем писать $f(a_1, \ldots, a_n) = \infty$, а если f на этом наборе определена, то будем писать $f(a_1, \ldots, a_n) < \infty$. Таким образом,

$$\operatorname{dom} f = \{(a_1, \ldots, a_n) : f(a_1, \ldots, a_n) < \infty\}.$$

§ 18.2. Программы с метками

В этом разделе рассмотрим в качестве средства описания алгоритмов простейший язык программирования для написания программ с метками. Они вычисляют арифметические функции, используя минимальные средства: элементарные присваивания и ветвления. Тем не менее можно доказать (см., например, [8]), что таким образом можно реализовать все классические возможности языков программирования: сложные выражения, циклы, подпрограммы и т. д.

Определим вначале синтаксис программ. Зафиксируем некоторые два счётных множества: имён переменных $\boldsymbol{V}$ и имён меток $\boldsymbol{L}$, которые будут использоваться в программах. Для удобства полагаем, что эти множества не пересекаются. Будем считать, что первое содержит

буквы латинского алфавита, возможно, с индексами: $x, x_1, x_2, \ldots$, $y, y_1, \ldots, z, z_1, \ldots$ и т. д., а второе — греческого: α, β, γ_1 и т. д.

Определение 143 (Программа с метками)**.** *Оператор — слово одного из следующих пяти видов:*

1) «$\alpha\ x \leftarrow 0;\ \beta$»;
2) «$\alpha\ x \leftarrow y;\ \beta$»;
3) «$\alpha\ x \leftarrow s(y);\ \beta$»;
4) «$\alpha\ x \leftarrow (y < z);\ \beta$»;
5) «α **Если** x **то** β **Иначе** γ».

Здесь, как мы уже сказали, $x, y \in \boldsymbol{V}$, $\alpha, \beta, \gamma \in \boldsymbol{L}$. *Первые четыре слова называются* присваиваниями, *последнее —* ветвлением *(или* условным оператором*). Символ* α *называется* меткой оператора.

Программа с метками — конечная последовательность операторов, в которой все метки операторов попарно различны.

С помощью программ с метками (далее называемых просто программами) вычисляются (частичные) функции от натуральных аргументов, принимающие натуральные значения. С каждой программой Π свяжем множества входящих в неё переменных $\boldsymbol{V}_\Pi$ и меток $\boldsymbol{L}_\Pi$ (определите эти множества формально). В процессе работы программа изменяет значения переменных из $\boldsymbol{V}_\Pi$. Операционная семантика задаёт правила такого изменения.

Определение 144 (Конфигурация программы)**.** *Конфигурация программы* Π *— это пара* (λ, σ), *где* $\lambda \in \boldsymbol{L}_\Pi$, $\sigma : \boldsymbol{V}_\Pi \to \omega$. *При этом* λ *называется текущей меткой, а* $\sigma(x)$ *— текущим значением переменной* $x \in \boldsymbol{V}_\Pi$.

Как и для ранее рассмотренных автоматов определим отношение перехода за один шаг.

Определение 145 (Переход за один шаг)**.** *Будем говорить, что конфигурация* $c = (\lambda, \sigma)$ *программы* Π переходит за один

шаг в конфигурацию $c' = (\mu, \tau)$ *(обозначаем* $c \vdash_\Pi c'$*), если выполнено одно из пяти условий:*

1) *в* Π *есть оператор вида «*$\lambda\ x \leftarrow 0;\ \mu$*»,* $\tau(x) = 0$ *и* $\tau(u) = \sigma(u)$ *при* $u \neq x$*;*
2) *в* Π *есть оператор вида «*$\lambda\ x \leftarrow y;\ \mu$*»,* $\tau(x) = \sigma(y)$ *и* $\tau(u) = \sigma(u)$ *при* $u \neq x$*;*
3) *в* Π *есть оператор вида «*$\lambda\ x \leftarrow s(y);\ \mu$*»,* $\tau(x) = \sigma(y) + 1$ *и* $\tau(u) = \sigma(u)$ *при* $u \neq x$*;*
4) *в* Π *есть оператор вида «*$\lambda\ x \leftarrow (y < z);\ \mu$*»,* $\tau(x) = 1$*, если* $\sigma(y) < \sigma(z)$*,* $\tau(x) = 0$*, если* $\sigma(y) \geqslant \sigma(z)$*,* $\tau(u) = \sigma(u)$ *при* $u \neq x$*;*
5) *в* Π *есть оператор вида «*λ **Если** x **то** β **Иначе** γ*»,* $\tau = \sigma$*,* $\mu = \beta$*, если* $\sigma(x) \neq 0$*,* $\mu = \gamma$*, если* $\sigma(x) = 0$*.*

Определение перехода за k шагов $\vdash_\Pi^k$ и за произвольное число шагов $\vdash_\Pi^*$ такое же как и раньше.

Более одного условия в определении 145 на предшествующей странице выполняться не может, так как в Π имеется не более одного оператора с меткой λ. Поэтому программы обладают свойством детерминированности.

Предложение 138. *Для программы* Π*, её конфигурации* (λ, σ) *и натурального числа* k *существует не более одной конфигурации* (μ, τ) *такой, что* $(\lambda, \sigma) \vdash_\Pi^k (\mu, \tau)$*.*

Доказательство. Аналогично предложению 93 на стр. 285, индукцией по k. □

Конфигурация, которая не переходит ни в какую за один шаг, называется заключительной. Очевидно, что конфигурация заключительная, только если текущая метка не является меткой никакого оператора. Такие метки мы тоже будет называть заключительными, всегда можно считать, что такая метка единственна.

Как и ранее можно ввести понятие вычисления: последовательности конфигураций, в которой каждая следующая получается переходом за один шаг из предыдущей. Если вычисление, начатое конфигу-

рацией (λ, σ), заканчивается заключительной конфигурацией (μ, τ), то пишем $\Pi(\lambda, \sigma) = (\mu, \tau)$ и говорим, что вычисление останавливается. Если вычисление оказывается бесконечным, то пишем $\Pi(\lambda, \sigma) = \infty$ и говорим, что вычисление зацикливается.

Пусть Π — программа, $\boldsymbol{V}_\Pi$ — множество её переменных. Выделим среди эти переменных некоторое подмножество входных переменных $x_1, \ldots, x_n$ и одну выходную переменную y (она может быть одной из входных). Значение выходной переменной в заключительной конфигурации будет результатом вычисления. Переменные $\boldsymbol{V}_\Pi$, не являющиеся входными, будем называть вспомогательными.

С помощью $c_0(a_1, \ldots, a_n)$ обозначим начальную конфигурацию, в которой текущая метка является меткой первого оператора программы, все вспомогательные переменные имеют текущее значение ноль, а значение входной переменной x_i равно a_i.

Определение 146 (Вычисляемая программой функция)**.** *Программа* Π *с входными переменными* $x_1, \ldots, x_n$ *и выходной переменной* y *вычисляет частичную функцию* $f : \omega^n \to \omega$*, если для всякого набора значений аргументов* $a_1, \ldots, a_n \in \omega$ *выполнено одно из двух:*

1) $f(a_1, \ldots, a_n)$ *определено и равно* b*,* $\Pi(c_0(a_1, \ldots, a_n)) = (\mu, \tau)$ *и* $\tau(y) = b$*;*
2) $f(a_1, \ldots, a_n)$ *неопределено и* $\Pi(c_0(a_1, \ldots, a_n)) = \infty$*.*

Частичную функцию, вычисляемую программой Π *с входными переменными* $x_1, \ldots, x_n$ *и выходной переменной* y*, обозначим* $\Phi_{\Pi,y}(x_1, \ldots, x_n)$*.*

Арифметическая (частичная) функция $f(x_1, \ldots, x_n)$ *называется программно вычислимой, если*

$$f(x_1, \ldots, x_n) = \Phi_{\Pi,y}(x_1, \ldots, x_n)$$

для некоторой программы Π *и переменных* $x_1, \ldots, x_n, y$*.*

Заметим, что в нашем языке нет понятия подпрограммы. Мы будем иногда использовать имя одной ранее написанной програм-

мы внутри текста другой: $\Pi = \ldots\, \alpha\ \Pi_1\ \beta \ldots$. Это означает вставку программы $\alpha\ \Pi_1\ \beta$ с начальной меткой α и (единственной) заключительной меткой β в соответствующее место программы Π. При этом мы предполагаем, что метки операторов программы Π_1 (кроме α и β) не совпадают с метками других операторов программы Π.

При этом следует позаботиться, чтобы в программе Π_1 не были «испорчены» значения переменных, которые используются в других частях программы Π. Например, все такие значения можно сохранить в каких-то новых переменных, а потом — восстановить их.

Рассмотрим несколько примеров программ.

Пример 88. *Следующая программа* Π_0 *вычисляет функцию* $\Phi_{\Pi_0,x}(x)$ *всюду равную нулю:*

$\alpha\ x \leftarrow 0; \beta$

Пример 89. *Следующая программа* Π_s *вычисляет функцию* $\Phi_{\Pi_s,x}(x)$ *прибавления единицы* $\Phi_{\Pi_s,x}(x) = x + 1$*:*

$\alpha\ x \leftarrow s(x); \beta$

Пример 90. *Продемонстрируем, что приведённая ниже программа* Π_+ *с входными переменными* x, y, z *вычисляет в выходной переменной* x *функцию* $\Phi_{\Pi_+,x}(x, y, z) = x + y - z$*, если* $z \leqslant y$*:*

$$\begin{array}{ll} \alpha & u \leftarrow (z < y);\ \beta \\ \beta & \textbf{Если } u \textbf{ то} \quad \gamma \textbf{ Иначе } \zeta \\ \gamma & z \leftarrow s(z); \qquad \delta \\ \delta & x \leftarrow s(x); \qquad \alpha \end{array}$$

Используется индукция по $y - z$*. При* $y - z = 0$ *получаем* $z = y$*, поэтому переменная* u *получит значение* 0*, после ветвления текущей меткой станет* ζ *и эта конфигурация будет заключительной. Таким образом, результат равен* $x = x + y - z$*.*

Пусть $y - z = n > 0$ *и при* $y - z < n$ *утверждение уже доказано. Тогда* $z < y$*, переменная* u *будет равна единице, затем произойдёт увеличение на единицу переменных* z *и* x *и текущей меткой снова станет* α*. Но теперь* $y - z < n$*, так как* z *увеличилось, а* y *не изменилось. По индукционному предположению результат равен* $(x + 1) + y - (z + 1) = x + y - z$*, что и требуется.*

Если в этой программе входными считать только переменные x, y*, то в начальной конфигурации значение* z *будет равно нулю и мы получим*

$$\Phi_{\Pi_+,x}(x, y) = \Phi_{\Pi_+,x}(x, y, 0) = x + y - 0 = x + y,$$

то есть эта программа вычисляет сумму чисел.

Если в этой же программе входными считать переменные y, z, то в начальной конфигурации значение x будет равно нулю, тогда

$$\Phi_{\Pi_+,x}(y,z) = \Phi_{\Pi_+,x}(0,y,z) = 0 + y - z = y - z,$$

то есть эта программа вычисляет разность чисел.

Используя программу Π_+ можно построить программу для умножения двух чисел $\Pi_\times$.

Пример 91. *Покажем, что следующая программа $\Pi_\times$ с входными переменными x, y вычисляет в выходной переменной x функцию $\Phi_{\Pi_\times,x}(x,y) = y \times x$:*

$$\begin{array}{llll}
\eta & v \leftarrow x; & & \theta \\
\theta & x \leftarrow 0; & & \iota \\
\iota & w \leftarrow 0; & & \varkappa \\
\varkappa & u \leftarrow (w < v); & & \lambda \\
\lambda & \textbf{Если } u \textbf{ то} & \mu & \textbf{Иначе } \nu \\
\mu & z \leftarrow 0; & & \alpha \\
\zeta & w \leftarrow s(w); & & \varkappa \\
\alpha & \Pi_+ & & \zeta
\end{array}$$

Прежде всего отметим, что после первых трёх присваиваний значения у переменных будут такие: v — исходное значение x, $x = 0$, $w = 0$, текущей меткой будет $\varkappa$. Далее индукцией по $v - w$ покажем, что результатом будет $x + y \times (v - w)$.

При $v = w$ значение u становится нулевым, поэтому результат равен $x = x + y \times (v - w)$. Если $v - w = n > 0$ и для $v - w < n$ утверждение доказано, то значение u равно единице. Поэтому будет выполнена программа Π_+, которая, как мы уже доказали, прибавит к x значение y, затем на единицу увеличится w и текущей меткой снова станет $\varkappa$. Но теперь разность $v - w$ стала на единицу меньше, по индукционному предположению результатом будет $(x + y) + y \times (v - (w + 1)) = x + y \times (v - w)$, что и требуется.

Учитывая, что значение v равно исходному значению x, а значения x и w — нулю, получим, что результат равен $0 + y \times (x - 0) = y \times x$.

Пример 92. *Последний пример, который мы рассмотрим в этом параграфе, — это программа Π_∞, которая вычисляет нигде неопределённую функцию $\Phi_{\Pi_\infty,x}(x) = \infty$ для всех x:*

$$\alpha \quad x \leftarrow 0; \alpha$$

Здесь заключительных конфигураций нет, так как нет заключительных меток.

§ 18.3. Ограниченные программы

Приведённый нами синтаксис и семантика программ довольно просты, однако в дальнейшем для нас будет крайне желательно придать им ещё более простую форму, в частности — уменьшить количество видов операторов. Покажем, что все перечисленные выше типы присваиваний можно заменить всего двумя. Такие программы мы будем называть о г р а н и ч е н н ы м и.

Теорема 139. *Любая программа с метками эквивалентна ограниченной программе, в которой присутствуют только операторы трёх следующих видов:*

1) $\alpha\ x \leftarrow s(x);\ \beta$
2) $\alpha\ x \leftarrow d(x);\ \beta$
3) α **Если** x **то** β **Иначе** γ

где x — это произвольная переменная, а $d(x)$ — это ограниченный декремент (см. задачу 263 на стр. 378).

Доказательство. Покажем последовательно, как можно избавиться от операторов всех других видов. Во всех дальнейших программах предполагается, что все метки, начиная с γ, — новые.

Чтобы выполнить оператор «$\alpha\ x \leftarrow 0;\ \beta$», можно воспользоваться такой программой:

$$\begin{array}{ll} \alpha & \textbf{Если } x \textbf{ то } \gamma \textbf{ Иначе } \beta \\ \gamma & x \leftarrow d(x); \quad \alpha \end{array}$$

Здесь значение переменной x при помощи d будет уменьшаться до тех пор, пока не станет нулевым.

Оператор $\alpha\ x \leftarrow y;\ \beta$ реализуем так. Возьмём новую переменную z, обнулим её. Как это сделать, мы уже показали. Затем одновременно уменьшаем y и увеличиваем z, пока значение y не станет равным нулю. Теперь старое значение y находится в z. Далее обнуляем x. Наконец, последовательно уменьшаем z и одновременно увеличиваем x и y, пока z не обратится в ноль. В результате старое значение

y из z будет одновременно перемещено и в x, и в y. Эти действия реализуются в следующей программе:

$$\begin{array}{ll}
\alpha & z \leftarrow 0; \quad \gamma \\
\gamma & \textbf{Если } y \textbf{ то } \delta \textbf{ Иначе } \eta \\
\delta & y \leftarrow d(y); \quad \zeta \\
\zeta & z \leftarrow s(z); \quad \gamma \\
\eta & x \leftarrow 0; \quad \theta \\
\theta & \textbf{Если } z \textbf{ то } \iota \textbf{ Иначе } \beta \\
\iota & z \leftarrow d(z); \quad \varkappa \\
\varkappa & x \leftarrow s(x); \quad \lambda \\
\lambda & y \leftarrow s(y); \quad \theta
\end{array}$$

Оператор $\alpha\ x \leftarrow s(y);\ \beta$ является простой комбинацией

$$\begin{array}{ll}
\alpha & x \leftarrow y; \quad \gamma \\
\gamma & y \leftarrow s(y);\ \beta
\end{array}$$

Наконец, оператор присваивания вида $\alpha\ x = (y < z);\ \beta$ реализуем следующим способом. Скопируем значения переменных y и z в новые переменные u и v соответственно, после чего будем одновременно уменьшать u и v. Если u станет нулём раньше v, то в x нужно записать единицу, иначе — ноль:

$$\begin{array}{ll}
\alpha & u \leftarrow y; \quad \gamma \\
\gamma & v \leftarrow z; \quad \delta \\
\delta & x \leftarrow 0; \quad \zeta \\
\zeta & \textbf{Если } u \textbf{ то } \eta \textbf{ Иначе } \iota \\
\eta & u \leftarrow d(u); \quad \theta \\
\theta & v \leftarrow d(v); \quad \zeta \\
\iota & \textbf{Если } v \textbf{ то } \varkappa \textbf{ Иначе } \beta \\
\varkappa & x \leftarrow s(x); \quad \beta
\end{array}$$

Таким образом, мы для каждого из четырёх присваиваний старого вида нашли ограниченную программу, которая его реализует. $\square$

Задачи

263. Построить программы, вычисляющие в выходной переменной z следующие функции, и доказать их корректность:

(а) $\mathrm{dec}(x)$ — ограниченный декремент:

$$\mathrm{dec}(x) = \begin{cases} x-1, & \text{при } x > 0, \\ 0, & \text{иначе;} \end{cases}$$

(б) $x!$ — факториал;

(в) $[\sqrt{x}]$ — целая часть квадратного корня;

(г) x^y;

(д) $[\log_2 x]$ — целая часть логарифма;

(е) $x \bmod y$ остаток от деления;

(ж) $[x/y]$ целая часть частного.

264. Пусть П — программа и $|\boldsymbol{V}_\Pi| = \{x_1, \ldots, x_m\}$. Из определений следует, что при различной фиксации входных переменных (считаем, что входные переменные всегда упорядочены по возрастанию номеров) и выходной переменной программа может вычислять различные функции. Для удобства полагаем, что среди допустимых операций имеются четыре арифметических действия.

(а) Каково максимальное число функций от $n \leqslant m$ переменных, которое может вычислять П? Сколько всего разных функций может вычислить П?

(б) Построить программу $\Pi(m, n)$, которая вычисляет максимальное число различных функций от $n \leqslant m$ переменных.

(в) Построить программу $\Pi(m)$ с $|\boldsymbol{V}_\Pi| = m$, которая для каждого $n \leqslant m$ вычисляет максимальное число различных функций от n переменных. ▼

265. Решить предыдущую задачу в случае, когда порядок входных переменных является произвольным. ▼

266. Определить, сколько всего существует попарно неэквивалентных программ с метками, имеющих не более n операторов и только одну переменную, которая является одновременно входной и выходной. ▼

267. Определить, какие всюду определённые функции могут вычисляться ограниченными программами с метками, которые имеют только одну переменную, являющуюся одновременно входной и выходной. ▼

268. Пусть программа с метками содержит только пропозициональные переменные $x_1, \ldots, x_n$ и присваивания с булевыми связками (аналогично линейным программ из параграфа 13.2). Предложить способ, который позволяет определить, какую функцию вычисляет такая программа. ▼

269. Построить программы, вычисляющие в выходной переменной z следующие функции:

(а) $f_1(x,y) = \begin{cases} [(x+y)/2], & \text{если } x < y, \\ x^2y^3 & \text{в противном случае;} \end{cases}$

(б) $f_2(x,y) = \begin{cases} 3^y, & \text{если } \log_2(x+1) \geqslant y, \\ |x-y| & \text{в противном случае;} \end{cases}$

(в) $f_3(x,y) = \begin{cases} [xy/2], & \text{если } \log_2 x \leqslant y+2, \\ x^{[y/2]} & \text{в противном случае;} \end{cases}$

(г) $p(x) = \begin{cases} 1, & \text{если } x \text{ — простое число,} \\ 0 & \text{в противном случае.} \end{cases}$

270. Пусть программа П с одной входной переменной x вычисляет в переменной y некоторую всюду определённую взаимно однозначную функцию f, область значений которой совпадает с множеством всех натуральных чисел ω. Пусть $\boldsymbol{V}_{\text{П}} = \{x, y, z_1, \ldots, z_m\}$. Построить программу, которая вычисляет обратную к f функцию f^{-1}: $f^{-1}(x) = z$, если $f(z) = x$. ▼

271. Построить программу, которая по натуральному числу i вычисляет число Фибоначчи F_i.

Глава 19

Частично рекурсивные функции

Краткое содержание: базисные рекурсивные функции, операции суперпозиции, примитивной рекурсии и минимизации, классы частично рекурсивных и общерекурсивных функций, общерекурсивность некоторых арифметических функций, вычислимость частично рекурсивных функций, функции нумерации n-ок, частичная рекурсивность вычислимых функций.

Ключевые слова: частично рекурсивная функция, общерекурсивная функция, суперпозиция, примитивная рекурсия, минимизация.

В этом разделе мы изучим алгебраический подход к определению класса вычислимых функций. Каждая вычислимая функция будет получаться из некоторых простейших очевидно вычислимых базисных функций с помощью некоторых операций, вычислимость которых также не вызывает сомнения. Операция, которая дала название этому подходу, — р е к у р с и я — это способ задания функции путём определения каждого её значения в терминах ранее определённых её значений и других уже определённых функций.

§ 19.1. Построение частично рекурсивных функций

Сейчас мы будем рассматривать частичные арифметические функции. В этом разделе для нас будет существенна местность функции, поэтому мы при первом введении какой-либо функции всегда будем её указывать тем или иным способом.

Так как нам предстоит иметь дело с частичными функциями, то возможно возникновение формул типа $f(h(x), y)$, где значение функции h неопределено. В этом случае мы будем считать, что и значение всей формулы тоже неопределено. Этот принцип можно сформулировать следующим образом:

> чтобы значение функции было определено, необходимо, чтобы значения всех аргументов были определены, даже если они являются фиктивными.

Теперь перейдём к способу построения функций. Начнём с самых элементарных, из которых впоследствии будем строить все остальные.

Определение 147 (Базисные рекурсивные функции)**.** *Базисными рекурсивными функциями называются:*

1) *функция* 0*, которая не имеет аргументов. Значение этой функции равняется нулю;*
2) *одноместная функция* s*. Значение* $s(x)$ *равняется* $x + 1$ *для всех* x*;*
3) *всевозможные проектирующие* n*-местные функции* id_m^n*,* $n, m \in \omega$*,* $0 < m \leqslant n$*. Значение* $\mathrm{id}_m^n(x_1, \ldots, x_n)$ *равняется* x_m*.*

Чтобы из этих функций можно было строить более сложные, определим три следующие операции над функциями.

Определение 148 (Суперпозиция)**.** *Пусть даны* m*-местная функция* g *и* n*-местные функции* $f_1, \ldots, f_m$*. Функция* h *получена из*

$g, f_1, \ldots, f_m$ *суперпозицией, если*

$$h(x_1, \ldots, x_n) = g(f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)).$$

Суперпозиция обозначается так: $h = g(f_1, \ldots, f_m)$.

Фактически с операцией суперпозиции мы уже неоднократно встречались, например, когда определяли формулы для булевых функций (определение 26 на стр. 66).

Определение 149 (Примитивная рекурсия)**.** *Допустим, что имеются* $(n-1)$*-местная функция* f *и* $(n+1)$*-местная функция* g*. Функция* h *получена из* f *и* g *примитивной рекурсией, если*

1) $h(x_1, \ldots, x_{n-1}, 0) = f(x_1, \ldots, x_{n-1})$;
2) $h(x_1, \ldots, x_{n-1}, i+1) = g(x_1, \ldots, x_{n-1}, i, h(x_1, \ldots, x_{n-1}, i))$ *для каждого* $i \in \omega$.

Функция h *обозначается в этом случае с помощью* $\mathrm{Pr}[f, g]$.

В случае, когда $n = 1$, то есть функция h зависит от одного аргумента i, а аргументов $x_1, \ldots, x_{n-1}$ нет, схема примитивной рекурсии принимает вид

$$h(i) = \begin{cases} c, & \text{если } i = 0, \\ g^{(2)}(i-1, h(i-1)) & \text{иначе,} \end{cases}$$

для всех $i \in \omega$.

Заметим, что если исходные функции в операторах суперпозиции и примитивной рекурсии всюду определены, то и результирующие функции также всюду определены. Следующий оператор позволяет определять частичные функции.

Определение 150 (Минимизация)**.** *Пусть* $f(x_1, \ldots, x_{n-1}, x_n)$ *—* n*-местная функция. Частичная* $(n-1)$*-местная функция* h *получена из* f *с помощью минимизации:* $h = \mu f$*, если* $h(x_1, \ldots, x_{n-1}) = y$ *в том и только том случае, когда*

1) $f(x_1, \ldots, x_{n-1}, z)$ *определено и не равно нулю при всех* $z < y$;

2) $f(x_1, \ldots, x_{n-1}, y) = 0$.

Следовательно, если $f(x_1, \ldots, x_{n-1}, y) \neq 0$ *для всех натуральных* y, *то* $h(x_1, \ldots, x_{n-1})$ *неопределено.*

Операции примитивной рекурсии и минимизации по сути своей напоминают циклы «Для» и «Пока», которые используются в языках программирования. В самом деле, из определения следует, что если $h^{(n)} = \mathrm{Pr}[f, g]$, то значение h на аргументах $x_1, \ldots, x_n$ можно вычислить так:

1: $i \leftarrow 0$
2: $y \leftarrow f(x_1, \ldots, x_{n-1})$
3: **Для всех** $i \leftarrow 0, \ldots, x_n - 1$ **выполнять**
4: $\quad y \leftarrow g(x_1, \ldots, x_{n-1}, i, y)$
5: **Конец Для**
6: **Вернуть** y

а если $h^{(n)} = \mu f$, то так:

1: $y \leftarrow 0$
2: **Пока** $f(x_1, \ldots, x_{n-1}, y) \neq 0$ **выполнять**
3: $\quad y \leftarrow y + 1$
4: **Конец Пока**
5: **Вернуть** y

Функции, которые можно получить из базисных с помощью трёх перечисленных операций, и называются частично рекурсивными.

Определение 151 (Частично рекурсивные и общерекурсивные функции)**.** *Частичная функция, которая может быть получена из базисных рекурсивных функций при помощи операторов суперпозиции, примитивной рекурсии и минимизации называется ч а с т и ч н о р е к у р с и в н о й ф у н к ц и е й (сокращённо: ч. р. ф.). Если частично рекурсивная функция является всюду определённой, то она называется о б щ е р е к у р с и в н о й (сокращённо: о. р. ф.).*

Приведём некоторые примеры ч. р. ф.

Пример 93. *Нульместные константы:* $1^{(0)}, 2^{(0)}, \ldots$ *легко определяются по индукции с помощью суперпозиции функции* s *и предыдущей константы:*

$$1 = s(0); \quad 2 = s(1); \quad \ldots$$

Пример 94. *Чтобы определить одноместные константы* $0^{(1)}(x), 1^{(1)}(x), \ldots,$ *рассмотрим одноместную функцию* $f = \Pr[0, \mathrm{id}_2^2]$. *Тогда*

$$f(0) = 0 \quad \textit{и} \quad f(i+1) = \mathrm{id}_2^2(i, f(i)) = f(i) = 0.$$

Следовательно, $f = 0^{(1)}$. *Аналогично получаются функции-константы для остальных чисел:* $i^{(1)} = \Pr[i^{(0)}, \mathrm{id}_2^2]$.

Индукцией по m *легко построить и функции-константы произвольной местности:* $n^{(m+1)} = \Pr[n^{(m)}, \mathrm{id}_{m+2}^{m+2}]$.

Пример 95. *Характеристическая функция нуля* $\mathrm{not}(x)$, *превращающая нуль в единицу, а остальные числа — в нуль, тоже строится при помощи примитивной рекурсии:* $\mathrm{not}^{(1)} = \Pr[1^{(0)}, 0^{(2)}]$. *В самом деле:*

$$\mathrm{not}(0) = 1, \quad \mathrm{not}(x+1) = 0^{(2)}(x, \mathrm{not}(x)) = 0.$$

Пример 96. *Характеристическая функция множества положительных натуральных чисел* $\mathrm{test}(x)$ *легко получается суперпозицией* not *с собой же:* $\mathrm{test} = \mathrm{not}(\mathrm{not})$.

Пример 97. *Чтобы показать частичную рекурсивность сложения, сначала построим трёхместную вспомогательную функцию* $s^{(3)} = s(\mathrm{id}_3^3)$, *увеличивающую на единицу третий аргумент:* $s^{(3)}(x, y, z) = z + 1$. *Теперь рассмотрим* $\mathrm{sum} = \Pr[\mathrm{id}_1^1, s^{(3)}]$. *Покажем, что* $\mathrm{sum}(x, y) = x + y$ *индукцией по* y. *В самом деле, по определению примитивной рекурсии получаем:*

$$\mathrm{sum}(x, 0) = \mathrm{id}_1^1(x) = x = x + 0.$$

Предположим, что для y *утверждение доказано:* $\mathrm{sum}(x, y) = x + y$, *тогда для* $y + 1$ *получается*

$$\mathrm{sum}(x, y+1) = s^{(3)}(x, y, \mathrm{sum}(x, y)) = \\ = s^{(3)}(x, y, x+y) = (x+y) + 1 = x + (y+1),$$

что и нужно.

Пример 98. *Умножение строится аналогично сложению. Сначала построим вспомогательную функцию с тремя аргументами* $\mathrm{sum}^{(3)} = \mathrm{sum}(\mathrm{id}_1^3, \mathrm{id}_3^3)$, *значением её является сумма первого и последнего:* $\mathrm{sum}^{(3)}(x, y, z) = x + z$. *Пусть теперь* $\mathrm{prod} = \Pr[0^{(1)}, \mathrm{sum}^{(3)}]$. *Покажем, что* $\mathrm{prod}(x, y) = x \times y$ *индукцией по* y. *По определению примитивной рекурсии получаем:*

$$\mathrm{prod}(x, 0) = 0^{(1)}(x) = 0 = x \times 0.$$

Если для y *уже доказано, что* $\mathrm{prod}(x, y) = x \times y$, *то для* $y + 1$ *получается*

$$\begin{aligned}\mathrm{prod}(x, y+1) &= \mathrm{sum}^{(3)}(x, y, \mathrm{prod}(x, y)) = \\ &= \mathrm{sum}^{(3)}(x, y, x \times y) = x \times y + x = x \times (y+1).\end{aligned}$$

Пример 99. *Функция возведения в степень* $\mathrm{pow}^{(2)}(x, y) = x^y$ *строится аналогично двум предыдущим:*

$$\mathrm{prod}^{(3)} = \mathrm{prod}\left(\mathrm{id}_1^3, \mathrm{id}_3^3\right), \quad \mathrm{pow} = \Pr\left[1^{(1)}, \mathrm{prod}^{(3)}\right].$$

Пример 100. *Рассмотрим функцию* $\mathrm{dec}(x)$ *— ограниченного декремента:* $\mathrm{dec}(x+1) = x$, $\mathrm{dec}(0) = 0$. *Построим* $\mathrm{dec} = \Pr\left[0^{(0)}, \mathrm{id}_1^2\right]$. *Тогда*

$$\mathrm{dec}(0) = 0, \quad \mathrm{dec}(x+1) = \mathrm{id}_1^2(x, \mathrm{dec}(x)) = x.$$

Пример 101. *С помощью ограниченного декремента и примитивной рекурсии можно построить функцию ограниченного вычитания:* $\lfloor x - y \rfloor$. *Точно так же как для сложения и умножения легко можно показать, что* $\lfloor - \rfloor = \Pr\left[\mathrm{id}_1^1, \mathrm{dec}\left(\mathrm{id}_3^3\right)\right]$.

Пример 102. *Чтобы построить характеристическую функцию для отношения «меньше»* $\mathrm{less}(x, y)$, *можно прибегнуть к помощи ограниченного вычитания:* $\mathrm{less} = \mathrm{test}\left(\left\lfloor \mathrm{id}_2^2 - \mathrm{id}_1^2 \right\rfloor\right)$. *Тогда* $\mathrm{less}(x, y)$ *будет равно единице, если разность* $y - x$ *положительна, то есть* $x < y$, *или нулю иначе.*

Пример 103. *Характеристическую функцию отношения «меньше или равно»* $\mathrm{leq}(x, y)$ *легко получить суперпозицией отрицания и предыдущей:* $\mathrm{leq} = \mathrm{not}\left(\mathrm{less}\left(\mathrm{id}_2^2, \mathrm{id}_1^2\right)\right)$.

Пример 104. *Характеристическая функция равенства* $\mathrm{eq}(x, y)$ *строится с помощью* leq: $\mathrm{eq} = \mathrm{leq}\left(\mathrm{id}_1^2, \mathrm{id}_2^2\right) \times \mathrm{leq}\left(\mathrm{id}_2^2, \mathrm{id}_1^2\right)$.

Пример 105. *Нигде не определённая функция* $\delta^{(1)}$ *может быть задана при помощи минимизации ненулевой двухместной функции-константы:* $\delta = \mu 1^{(2)}$.

Как видим, действительно, многие вычислимые в интуитивном представлении функции являются ч. р. ф.

Замечание 17 (О значении минимизации)**.** Из приведённых примеров может сложиться впечатление, что единственная задача операции минимизации — это построение частичных функций из всюду определённых. Это не так. Существуют о. р. ф., которые без операции минимизации построить нельзя. Одним из наиболее известных примеров такого рода является двухместная функция, открытая В. Аккерманом, которая определяется индуктивно:

$$\begin{aligned} A(0, y) &= y + 1, \\ A(x + 1, 0) &= A(x, 1), \\ A(x + 1, y + 1) &= A(x, A(x + 1, y)). \end{aligned}$$

Можно показать, что функция Аккермана растёт быстрее, чем любая о. р. ф., построенная без минимизации. Поэтому саму функцию Аккермана без минимизации построить нельзя (сравните этот результат с задачей 277 на стр. 397).

§ 19.2. Вычислимость частично рекурсивных функций

В этом параграфе начнём рассматривать соотношение между программно вычислимыми функциями и ч. р. ф. Основным результатом будет то, что оба класса функций совпадают. Сейчас мы докажем включение в одну сторону.

Прежде всего продемонстрируем, что операции суперпозиции, примитивной рекурсии и минимизации не выводят за рамки класса программно вычислимых функций. Следовательно, класс программно вычислимых функций замкнут относительно этих действий.

Предложение 140. *Если частичные функции* $g^{(m)}$, $f_1^{(n)}$, …, $f_m^{(n)}$ *программно вычислимы, то их суперпозиция* $h^{(n)} = g(f_1, \ldots, f_m)$ *тоже программно вычислима.*

Доказательство. Доказательство настоящего и двух следующих предложений сводится к написанию программы, которая вычисляет

функцию h, если программы, вычисляющие исходные функции, уже даны.

Прежде чем перейти к этому построению введём такое обозначение: $\alpha\ K_{y_1\ldots y_n}^{x_1\ldots x_n}\ \beta$ — это программа с начальной меткой α и заключительной меткой β, которая копирует переменные $x_1, \ldots, x_n$ в $y_1, \ldots, y_n$ соответственно. Такая программа имеет вид:

$$\begin{array}{rl} \alpha & y_1 \leftarrow x_1;\ \alpha_2 \\ \alpha_2 & y_2 \leftarrow x_2;\ \alpha_3 \\ & \ldots \\ \alpha_n & y_n \leftarrow x_n;\ \beta \end{array}$$

Промежуточные метки $\alpha_2, \ldots, \alpha_n$ могут быть произвольными, они выбираются так, чтобы не совпадать с другими метками программы. По аналогии с помощью $\alpha\ Z_{x_1\ldots x_n}\ \beta$ обозначаем программу с начальной меткой α и заключительной меткой β, обнуляющую переменные $x_1, \ldots, x_n$:

$$\begin{array}{rl} \alpha & x_1 \leftarrow 0;\ \alpha_2 \\ \alpha_2 & x_2 \leftarrow 0;\ \alpha_3 \\ & \ldots \\ \alpha_n & x_n \leftarrow 0;\ \beta \end{array}$$

Итак, пусть имеются программы $\Pi_1, \ldots, \Pi_m$, которые вычисляют функции $f_1, \ldots, f_m$. Без ограничения общности полагаем, что начальными их метками являются $\gamma_1, \ldots, \gamma_m$, а заключительными — $\delta_1, \ldots, \delta_m$ соответственно. Будем считать, что все эти программы используют переменные $x_1, \ldots, x_n$, y, $z_1, \ldots, z_k$, из которых $x_1, \ldots, x_n$ — входные, y — выходная, $z_1, \ldots, z_k$ — вспомогательные. Программа Π_g для вычисления g тоже пусть имеет входные переменные $x_1, \ldots, x_m$, выходную y, вспомогательные $z_1, \ldots, z_k$ и начальную метку ζ.

Основная идея вычисления h заключается в следующем: сохраним значения $x_1, \ldots, x_n$ в новых вспомогательных переменных $u_1, \ldots, u_n$, выполним Π_1, сохраним полученный результат y во новой вспомогательной переменной v_1. Далее восстановим значения $x_1, \ldots, x_n$ из $u_1, \ldots, u_n$, обнулим $y, z_1, \ldots, z_k$, выполним Π_2, сохраним полученный

$$
\begin{array}{rll}
\alpha_1 & K^{x_1\ldots x_n}_{u_1\ldots u_n} & \gamma_1 \\
 & \Pi_1 & \\
\delta_1 & v_1 \leftarrow y; & \beta_1 \\
\beta_1 & K^{u_1\ldots u_n}_{x_1\ldots x_n} & \eta_1 \\
\eta_1 & Z_{yz_1\ldots z_k} & \gamma_2 \\
 & \Pi_2 & \\
\delta_2 & v_2 \leftarrow y; & \beta_2 \\
 & \ldots & \\
\beta_{m-1} & K^{u_1\ldots u_n}_{x_1\ldots x_n} & \eta_{m-1} \\
\eta_{m-1} & Z_{yz_1\ldots z_k} & \gamma_m \\
 & \Pi_m & \\
\delta_m & v_m \leftarrow y; & \beta_m \\
\beta_m & K^{v_1\ldots v_m}_{x_1\ldots x_m} & \eta_m \\
\eta_m & Z_{yz_1\ldots z_k} & \zeta \\
 & \Pi_g &
\end{array}
$$

Рис. 114: Программа для суперпозиции, предложение 140.

результат y в новой вспомогательной переменной v_2 и т. д. В результате в переменных $v_1, \ldots, v_m$ хранятся результаты $f_1(x_1, \ldots, x_n)$, $\ldots$, $f_m(x_1, \ldots, x_n)$ соответственно. После этого остаётся скопировать $v_1, \ldots, v_m$ в $x_1, \ldots, x_m$, обнулить $y, z_1, \ldots, z_k$ и выполнить Π_g, чтобы вычислить $g(v_1, \ldots, v_m)$. Последнее и будет значением $h(x_1, \ldots, x_n)$.

Программа Π, реализующая эти действия, показана на рис. 114.□

Предложение 141. *Если частичные функции $f^{(n-1)}$ и $g^{(n+1)}$ программно вычислимы, то полученная с помощью примитивной рекурсии функция $h^{(n)} = \mathrm{Pr}[f, g]$ тоже программно вычислима.*

Доказательство. В доказательстве мы будем использовать программы вида $K^{x_1\ldots x_n}_{y_1\ldots y_n}$ и $Z_{x_1\ldots x_n}$, построенные в доказательстве предыдущего предложения.

$$
\begin{array}{lll}
\alpha & K^{x_1\ldots x_n}_{u_1\ldots u_n} & \gamma \\
& \Pi_f & \\
\delta & v \leftarrow (i < u_n);\ \beta & \\
\beta & \textbf{Если } v \textbf{ то} & \theta \textbf{ Иначе } \mu \\
\theta & K^{u_1\ldots u_{n-1}}_{x_1\ldots x_{n-1}} & \iota \\
\iota & x_n \leftarrow i; & \varkappa \\
\varkappa & x_{n+1} \leftarrow y; & \lambda \\
\lambda & Z_{yz_1\ldots z_k} & \zeta \\
& \Pi_g & \\
\eta & i \leftarrow s(i); & \delta
\end{array}
$$

Рис. 115: Программа для примитивной рекурсии, предложение 141.

Опять у нас имеются программы Π_f и Π_g для вычисления функций f и g соответственно. Полагаем, что выходной переменной у них является y, вспомогательными — $z_1, \ldots, z_k$, а входными — $x_1, \ldots, x_{n-1}$ у первой и $x_1, \ldots, x_{n+1}$ у второй. Начальной и заключительной метками пусть соответственно являются γ, δ у Π_f и ζ, η у Π_g.

Вычислять h будем так. Сохраним значения $x_1, \ldots, x_n$ в новых вспомогательных переменных $u_1, \ldots, u_n$ и выполним Π_f, чтобы найти $f(x_1, \ldots, x_{n-1}) = h(x_1, \ldots, x_{n-1}, 0)$. Далее будем повторять следующую последовательность действий, пока значение переменной i не станет равно u_n (входное значение x_n): восстанавливаем $x_1, \ldots, x_{n-1}$ из $u_1, \ldots, u_{n-1}$, копируем i в x_n и y в x_{n+1}. обнуляем $y, z_1, \ldots, z_k$, выполняем Π_g, увеличиваем i на единицу. Нетрудно видеть, что на каждом шаге значение переменной y будет равно $h(x_1, \ldots, x_{n-1}, i)$, что напрямую вытекает из определения примитивной рекурсии. Следовательно, когда значения i и u_n сравняются, мы получим в y искомый результат $h(x_1, \ldots, x_{n-1}, x_n)$.

Программа Π для вычисления h описанным способом приведена на рис. 115. □

α $K^{x_1 \ldots x_n}_{u_1 \ldots u_n}$ β
β $x_{n+1} \leftarrow i;$ ζ
ζ $Z_{y z_1 \ldots z_k}$ γ
Π_f
δ **Если** y **то** η **Иначе** θ
η $i \leftarrow s(i);$ β
θ $y \leftarrow i;$ ι

Рис. 116: Программа для минимизации, предложение 142.

Предложение 142. *Если частичная функция $f^{(n+1)}$ программно вычислима, то её минимизация $h^{(n)} = \mu f$ тоже программно вычислима.*

Доказательство. Схема доказательства та же, что и в двух предыдущих предложениях.

Снова считаем, что программа Π_f вычисляет f, имеет выходную переменную y, вспомогательные $z_1, \ldots, z_k$, входные $x_1, \ldots, x_{n+1}$, начальную метку γ и заключительную δ.

Чтобы построить программу Π для вычисления h, мы реализуем такие действия. Копируем значения $x_1, \ldots, x_n$ в новые вспомогательные переменные $u_1, \ldots, u_n$. Далее повторяем следующие действия: копируем i в x_{n+1}, обнуляем переменные $y, z_1, \ldots, z_k$, выполняем Π_f, если значение y не равно нулю, то увеличиваем i на единицу и делаем это всё снова. Как только y стал равен нулю, то мы нашли наименьшее значение i, для которого $f(x_1, \ldots, x_n, i) = 0$, оно и будет значением $h(x_1, \ldots, x_n)$.

Описанные действия реализованы в программе Π на рис. 116. □

Из этих трёх предложений получаем следующую теорему.

Теорема 143. *Каждая ч. р. ф. программно вычислима.*

Доказательство. Применим индукцию по построению ч. р. ф. f.

Базис индукции. Базисные рекурсивные функции, очевидно, вычисляются следующими программами, где y — выходная перемен-

ная, x_i — входные. Константа 0:

$$\alpha \quad y \leftarrow 0;\ \beta$$

функция s:

$$\alpha \quad y \leftarrow s(x_1);\ \beta$$

функция id_m^n:

$$\alpha \quad y \leftarrow x_m;\ \beta$$

Индукционный шаг. Пусть функция f построена из каких-то функций $f_1, \ldots, f_k$ при помощи суперпозиции, примитивной рекурсии или минимизации. По индукционному предположению все функции $f_1, \ldots, f_k$ программно вычислимы. Тогда, используя предложения 140 на стр. 386, 141 на стр. 388 и 142 на противоположной странице, делаем вывод, что и функция f программно вычислима. □

§ 19.3. Частичная рекурсивность вычислимых функций

Имеет место и утверждение, обратное теореме 143 на предыдущей странице. Чтобы его доказать, установим частичную рекурсивность некоторых функций, которые нам понадобятся.

Предложение 144. *Если $(m+1)$-местная функция $f(\bar{x}, y)$ является ч. р. ф. (соответственно о. р. ф.), то и следующие функции являются ч. р. ф. (соответственно о. р. ф.):*

$$g(\bar{x}, z) = \sum_{y<z} f(\bar{x}, y); \quad h(\bar{x}, z) = \prod_{y<z} f(\bar{x}, y).$$

Доказательство.

$$g = \mathrm{Pr}\left[0, \mathrm{id}_{m+2}^{m+2} + f\left(\mathrm{id}_1^{m+2}, \ldots, \mathrm{id}_m^{m+2}, \mathrm{id}_{m+1}^{m+2}\right)\right];$$

$$h = \mathrm{Pr}\left[1, \mathrm{id}_{m+2}^{m+2} \times f\left(\mathrm{id}_1^{m+2}, \ldots, \mathrm{id}_m^{m+2}, \mathrm{id}_{m+1}^{m+2}\right)\right]. \qquad \square$$

Предложение 145. *Для любого положительного натурального n следующая функция выбора* $\mathrm{case}^{(n+2)}$ *является о. р. ф.:*

$$\mathrm{case}(x, y_0, y_1, \ldots, y_{n-1}, y_n) = \begin{cases} y_i, & \text{если } x = i \text{ для } i = 0, \ldots, n-1; \\ y_n, & \text{если } x \geqslant n. \end{cases}$$

Доказательство. Нетрудно заметить, что

$$\begin{aligned} \mathrm{case} = \mathrm{eq}\left(\mathrm{id}_1^{n+2}, 0^{(n+2)}\right) \times \mathrm{id}_2^{n+2} + \cdots \\ \cdots + \mathrm{eq}\left(\mathrm{id}_1^{n+2}, (n-1)^{(n+2)}\right) \times \mathrm{id}_{n+1}^{n+2} + \\ + \mathrm{leq}\left(n^{(n+2)}, \mathrm{id}_1^{n+2}\right) \times \mathrm{id}_{n+2}^{n+2}. \end{aligned}$$ □

Следствие 146. *Следующая условная функция* $\mathrm{if}^{(3)}$ *является о. р. ф.:*

$$\mathrm{if}(x, y, z) = \begin{cases} y, & \text{если } x \neq 0; \\ z, & \text{иначе.} \end{cases}$$

Доказательство. $\mathrm{if}(x, y, z) = \mathrm{case}^{(3)}(x, z, y)$. □

Предложение 147. *Следующие функции являются о. р. ф.:*

1) mod — *остаток от деления (для удобства будет полагать, что* $x \bmod 0 = 0$*);*
2) $(x \mid y)$ — *характеристическая функция делимости y на x;*
3) $\left[x/y\right]$ — *результат целочисленного деления x на y (для удобства будет полагать, что* $\left[x/0\right] = 0$*);*
4) $\mathrm{ind}(x, y)$ — *наибольшая степень числа y, на которую делится x.*

Доказательство. 1) Остаток от деления x на y легко определяется индукцией по x:

$$\mathrm{mod}' = \mathrm{Pr}\left[0^{(1)}, \mathrm{if}\left(\mathrm{eq}\left(\mathrm{id}_3^3, \mathrm{dec}\left(\mathrm{id}_1^3\right)\right), 0^{(1)}, s\left(\mathrm{id}_3^3\right)\right)\right].$$

Здесь мы воспользовались тем, что $(x+1) \bmod y$ равно $(x \bmod y) + 1$, если остаток $x \bmod y$ меньше $y - 1$, или 0 в противном случае. Далее остаётся только переставить аргументы:

$$\mathrm{mod} = \mathrm{mod}'(\mathrm{id}_2^2, \mathrm{id}_1^2).$$

2) $(|) = \mathrm{not}(\mathrm{mod}') \times \mathrm{test}\left(\mathrm{id}_1^2\right)$.

3) Чтобы найти частное от деления x на y, можно просуммировать единицы для тех $i = 1, \ldots, x$, которые делятся на y:

$$\left[x/y\right] = \sum_{i<x} (y \mid (i+1)).$$

4) Для нахождения ind аналогично просуммируем единицы для тех $i = 1, \ldots, x$, для которых выполнено $y^i \mid x$:

$$\mathrm{ind}(x, y) = \sum_{i<x} \left(y^{i+1} \mid x\right). \qquad \square$$

Следующая группа функций, которая нас будет интересовать, — это функции нумерации пар и произвольных n-ок натуральных чисел и обратные им.

Предложение 148. *Функция* $\langle x, y\rangle = 2^x(2y+1) - 1$ *взаимно однозначно отображает множество пар натуральных чисел* $\omega \times \omega$ *на само множество натуральных чисел* ω.

Доказательство. Пусть $z = \langle x, y\rangle + 1$. Тогда z — положительное натуральное число, которое можно единственным способом разложить в произведение простых множителей: $z = 2^{a_2}3^{a_3}\cdots p^{a_p}$ для простых чисел $2, 3, \ldots, p$, где все показатели $a_2, \ldots, a_p$ — натуральные числа. Произведение $3^{a_3}\cdots p^{a_p}$ является нечётным числом, а 2^{a_2} — степенью двойки. Следовательно,

$$x = a_2, \quad y = \frac{3^{a_3}\cdots p^{a_p} - 1}{2}.$$

Из этих же рассуждений следует сюръективность функции $\langle \ldots \rangle$. $\square$

По индукции можно определить аналогичную функцию для произвольных n-ок для $n > 2$:

$$\langle x_1, \ldots, x_n, x_{n+1}\rangle = \left\langle \langle x_1, \ldots, x_n\rangle, x_{n+1}\right\rangle,$$

которая будет взаимно однозначно отображать ω^n на ω.

Предложение 149. *Все функции $\langle\ldots\rangle$ являются о. р. ф.*

Доказательство. Функция $\langle x, y\rangle$ является суперпозицией возведения в степень, сложения, умножения и констант. Каждая $(n+1)$-местная функция является суперпозицией n-местной и $\langle x, y\rangle$. □

Покажем, что общерекурсивны и обратные к ним функции.

Предложение 150. *Унарные функции $\langle\ldots\rangle_1$ и $\langle\ldots\rangle_2$ такие, что*

$$x = \langle\langle x, y\rangle\rangle_1, \quad y = \langle\langle x, y\rangle\rangle_2,$$

общерекурсивны.

Доказательство. Функцию $\langle\ldots\rangle_1$ можно построить используя ind: $\langle z\rangle_1 = \mathrm{ind}(z+1, 2)$. Вторая функция строится с помощью первой:

$$\langle z\rangle_2 = \frac{(z+1)/2^{\langle z\rangle_1} - 1}{2}.$$ □

По индукции можно доказать общерекурсивность функций, которые по коду n-ки $\langle x_1, \ldots, x_n\rangle$ возвращают x_i.

Предложение 151. *Общерекурсивны все унарные функции $\langle\ldots\rangle_i^n$ такие, что*

$$\langle\langle x_1, \ldots, x_n\rangle\rangle_i^n = x_i,$$

где $1 \leqslant i \leqslant n$.

Доказательство. Используем индукцию по n. Для $n = 2$ функции построены в предыдущем предложении.

Пусть теперь уже построены функции $\langle\ldots\rangle_i^n$. Согласно определению

$$z = \langle x_1, \ldots, x_n, x_{n+1}\rangle = \langle\langle x_1, \ldots, x_n\rangle, x_{n+1}\rangle,$$

поэтому $\langle z\rangle_1 = \langle x_1, \ldots, x_n\rangle$ и $\langle z\rangle_2 = x_{n+1}$. Для первой полученной n-ки можно применить индукционное предположение. Поэтому получаем

$$\langle z\rangle_i^{n+1} = \langle\langle z\rangle_1\rangle_i^n$$

для $i = 1, \ldots, n$ и

$$\langle z\rangle_{n+1}^{n+1} = \langle z\rangle_2.$$ □

Теперь можно приступить к доказательству основной теоремы.

Теорема 152. *Каждая программно вычислимая функция является ч. р. ф.*

Доказательство. Пусть дана программа П, вычисляющая n-местную функцию $f^{(n)}$. Будем считать, что все переменные программы П — это $x_1, \ldots, x_k$, из них первые n: $x_1, \ldots, x_n$ — являются входными, x_{n+1} — выходной, а остальные — вспомогательными. Также считаем, что метки программы П пронумерованы натуральными числами: $\alpha_0, \ldots, \alpha_m$, причём α_0 является начальной, а α_m — единственной заключительной.

Для начала укажем способ кодирования конфигураций программы П. Напомним, что конфигурация включает в себя метку, то есть одну из $\alpha_0, \ldots, \alpha_m$, и функцию, которая по каждой переменной определяет её значение. Поскольку метки пронумерованы, то каждую α_i можно закодировать натуральным числом i. Количество переменных заранее ограничено, поэтому все значения переменных $x_1, \ldots, x_k$ можно представить набором $(\sigma(x_1), \ldots, \sigma(x_k))$. Таким образом, конфигурация (α_i, σ) программы П может быть закодирована одним натуральным числом $c = \langle i, \sigma(x_1), \ldots, \sigma(x_k)\rangle$. Заметим, что при помощи обратных функций $\langle \ldots \rangle_j^{k+1}$ можно извлечь все составляющие конфигурации: метку $\alpha_{\langle c\rangle_1^{k+1}}$ и значение переменных x_j: $\sigma(x_j) = \langle c\rangle_{j+1}^{k+1}$ для $j = 1, \ldots, k$.

Далее для каждого оператора o с меткой α программы П построим одноместную о. р. ф. g_α, которая по коду предыдущей конфигурации будет давать код следующей. Напомним, что согласно теореме 139 на стр. 376 мы можем предполагать, что программа содержит только три вида операторов:

1) $\alpha\ x_j \leftarrow s(x_j);\ \alpha_i$
2) $\alpha\ x_j \leftarrow d(x_j);\ \alpha_i$
3) α **Если** x_j **то** α_{i_1} **Иначе** α_{i_2}

Покажем, как построить g_α в каждом из этих случаев соответственно:

1) $g_\alpha(c) = \left\langle i, \langle c\rangle_2^{k+1}, \ldots, \langle c\rangle_j^{k+1}, s\big(\langle c\rangle_{j+1}^{k+1}\big), \langle c\rangle_{j+2}^{k+1}, \ldots, \langle c\rangle_{k+1}^{k+1}\right\rangle$;
2) $g_\alpha(c) = \left\langle i, \langle c\rangle_2^{k+1}, \ldots, \langle c\rangle_j^{k+1}, \mathrm{dec}\big(\langle c\rangle_{j+1}^{k+1}\big), \langle c\rangle_{j+2}^{k+1}, \ldots, \langle c\rangle_{k+1}^{k+1}\right\rangle$;

3) $g_\alpha(c) = \left\langle \mathrm{if}(\langle c\rangle_{j+1}^{k+1}, i_1, i_2), \langle c\rangle_2^{k+1}, \ldots, \langle c\rangle_{k+1}^{k+1} \right\rangle$.

Фактически мы просто с помощью функций g_o описали формальное определение перехода программы с метками от одной конфигурации к следующей (определение 145 на стр. 371) в зависимости от вида оператора.

Далее при помощи функции case опишем общую унарную функцию $g^{(1)}$ перехода за один шаг:

$$g(c) = \mathrm{case}(\langle c\rangle_1^{k+1}, g_{\alpha_0}(c), \ldots, g_{\alpha_{m-1}}(c), c).$$

Здесь мы просто определили в зависимости от текущей метки, какой из функций g_α нужно воспользоваться. Теперь можно определить функцию перехода $g^{(2)}(c,p)$ за заданное число шагов p с помощью примитивной рекурсии: $g = \mathrm{Pr}\left[\mathrm{id}_1^1, g^{(1)}\left(\mathrm{id}_3^3\right)\right]$: чтобы вычислить $g^{(2)}(c,p)$ мы просто к c применяем p раз функцию g.

С помощью следующей функции $q^{(2)}(c,p)$ можно проверять, получилась ли у нас через p шагов заключительная конфигурация:

$$q(c,p) = \mathrm{not}\left(\mathrm{eq}\left(\langle g(c,p)\rangle_1^{k+1}, m\right)\right).$$

Она вернёт ноль, если метка через p шагов после конфигурации с кодом c станет заключительной α_m. Далее с помощью минимизации определяем число шагов, когда это произойдёт: $t^{(1)} = \mu q$. Таким образом, $t(c)$ — это количество шагов, которое выполнит программа Π на конфигурации с кодом c прежде, чем остановится. Тогда $g(c,t(c))$ — это код заключительной конфигурации, в которую придёт программа Π, а $\langle g(c,t(c))\rangle_{n+2}^{k+1}$ — это значение выходной переменной x_{n+1} в заключительной конфигурации.

Код начальной конфигурации строится так:

$$g_0(x_1,\ldots,x_n) = \langle 0, x_1, \ldots, x_n, 0, \ldots, 0\rangle.$$

Следовательно, функция f является ч. р. ф.:

$$f(x_1,\ldots,x_n) = \langle g(g_0(x_1,\ldots,x_n), t(g_0(x_1,\ldots,x_n)))\rangle_{n+2}^{k+1}. \qquad \square$$

В качестве следствия теорем 143 на стр. 390 и только что доказанной получаем

Следствие 153. *Классы ч. р. ф. и программно вычислимых функций совпадают.*

Задачи

272. Показать, что следующие функции являются ч. р. ф.:

(а) $\mathrm{fact}(x) = x!$ — факториал;

(б) $\min^{(n)}(x_1, \ldots, x_n)$ — наименьший из аргументов, $n > 0$;

(в) $\max^{(n)}(x_1, \ldots, x_n)$ — наибольший из аргументов, $n > 0$;

(г) $|x - y|$ — модуль разности. ▼

273. Доказать, что если функция $f(x_1, \ldots, x_n)$ является ч. р. ф., то и функция $g(x_1, \ldots, x_n) = f(x_{i_1}, \ldots, x_{i_n})$ является ч. р. ф. для каждой перестановки $(i_1, \ldots, i_n)$ чисел $1, 2, \ldots, n$. ▼

274. Пусть $g(x_1, \ldots, x_{n-1}, x_n)$ — ч. р. ф., a и $b > 0$ — натуральные числа. Доказать, что функция

$$f(x_1, \ldots, x_{n-1}, x_n) = \begin{cases} a, & \text{если } x_n < b \\ g(x_1, \ldots, x_{n-1}, x_n - b), & \text{если } x_n \geqslant b \end{cases}$$

тоже является ч. р. ф. ▼

275. Найти значения функции Аккермана $A(1, y)$ и $A(2, y)$ для всех натуральных y. ▼

276. Показать, что следующие функции являются ч. р. ф.:

(а) $\mathrm{rt}(n, x) = [\sqrt[n]{x}]$ — целая часть корня n-й степени из x;

(б) $\log(i, x) = [\log_i x]$;

(в) $p(x) = 1$, если x — простое число, и $p(x) = 0$ в противном случае;

(г) $\mathrm{pn}(k)$ — k-е простое число в порядке возрастания, $\mathrm{pn}(0) = 2$;

(д) $\tau(x)$ — количество различных делителей числа x, считать, что $\tau(0) = 0$;

(е) $\mathrm{digit}(n, m, i)$ — i-я цифра в m-ичном представлении числа n: то есть если $n = \sum_{i=0}^{\infty} a_i m^i$, где $0 \leqslant a_i \leqslant m - 1$, то $d(n, m, i) = a_i$;

(ж) НОД(x, y) — наибольший общий делитель чисел x и y. ▼

277. О. р. ф., которая может быть построена без использования минимизации, называется примитивно рекурсивной. Пусть функции g и f примитивно рекурсивны, $h = \mu g$, h — о. р. ф., и h мажорируется функцией f: $h(\bar{x}) < f(\bar{x})$ для всех $\bar{x}$. Доказать, что h тоже является примитивно рекурсивной. ▼

278. Показать, что функция $F(x)$ из задачи 271 на стр. 379 является о. р. ф. (Указание: показать сначала, что функция $g(x) = \langle F(x), F(x+1)\rangle$ является о. р. ф.) ▼

279. Доказать, что если значения о. р. ф. $f(x)$ изменить на конечном множестве, то получившаяся функция $f'(x)$ также будет о. р. ф. ▼

280. Доказать, что из константы 0 и функций id_m^n с помощью суперпозиции и примитивной рекурсии нельзя получить функцию $s(x) = x + 1$ и функцию $d(x) = 2 \times x$. ▼

281. Пусть f — взаимно однозначная на ω о. р. ф. Доказать, что обратная функция f^{-1} тоже является о. р. ф., явно её построив. ▼

282. Пусть $g(x_1, \ldots, x_n, y)$ — о. р. ф. Доказать, что функция

$$f(x_1, \ldots, x_n, y, z) = \begin{cases} \sum\limits_{i=0}^{z} g(x_1, \ldots, x_n, y + i), & \text{при } y \leqslant z, \\ 0, & \text{при } y > z \end{cases}$$

общерекурсивна. ▼

283. Доказать, что если функции $f(x_1, \ldots, x_n, y), g(x_1, \ldots, x_n, y)$ и $h(x_1, \ldots, x_n, y)$ общерекурсивны, то функция

$$F(x_1, \ldots, x_n) = \min\{y : f(x_1, \ldots, x_n, y) = 0 \text{ или } g(x_1, \ldots, x_n, y) > h(x_1, \ldots, x_n, y)\}$$

является ч. р. ф. ▼

284. Предположим, что все пары (x, y) натуральных чисел упорядочены по возрастанию суммы $x + y$, а внутри группы пар с одинаковой суммой — по возрастанию координаты x. Этот порядок выглядит так:

$$(0,0), (0,1), (1,0), (0,2), (1,1), (2,0), \ldots$$
$$\ldots, (0, x+y), (1, x+y-1), \ldots, (x, y), \ldots, (x+y, 0), \ldots$$

Пусть $k(x, y)$ — это номер пары (x, y) в этом порядке (будем считать, что пара $(0, 0)$ имеет номер 0). Тогда функция $k^{(2)}$ взаимно однозначно нумерует все пары натуральных чисел.

(а) Доказать, что $k(x, y) = \dfrac{(x+y)(x+y+1)}{2} + x$.

(б) Найти обратные функции $k_1(z)$ и $k_2(z)$ такие, что $k_1(k(x, y)) = x$, $k_2(k(x, y)) = y$ и, следовательно, $k(k_1(z), k_2(z)) = z$.

(в) Показать, что все эти функции общерекурсивны. ▼

Глава 20

Машины Тьюринга

Краткое содержание: определение машин Тьюринга и класса вычислимых ими функций, примеры машин Тьюринга, стандартная заключительная конфигурация, односторонние машины Тьюринга, ограничение алфавита, вычислимость по Тьюрингу и программная, универсальная машина Тьюринга.

Ключевые слова: машина Тьюринга, рабочая лента, головка, команда, программа, конфигурация, начальная и заключительная конфигурации, вычисление, стандартная заключительная конфигурация, односторонняя машина Тьюринга, вычислимость по Тьюрингу, универсальная машина Тьюринга.

§ 20.1. Основные определения

Рассматриваемая в этом разделе модель алгоритмов предложена английским математиком А. Тьюрингом в 1937 г. ещё до создания современных компьютеров.[1] Он исходил из общей идеи моделирования

[1] Аналогичная модель вычислений была примерно в то же время определена Э. Постом. Поэтому иногда такие машины называют машинами Тьюринга-Поста.

работы вычислительного устройства, которое работает в соответствии с некоторым строгим формальным предписанием. В машине Тьюринга расчленение процесса вычисления на элементарные шаги доведено в известном смысле до предела. Элементарным действием является замена одного символа в ячейке на другой и перемещение к соседней ячейке. При таком подходе процесс вычисления значительно удлиняется, но зато логическая структура процесса сильно упрощается и приобретает удобный для теоретического исследования вид.

Рабочее пространство машины Тьюринга состоит из неограниченной в обе стороны ленты, разбитой на отдельные ячейки. Такая «бесконечность» ленты является математической абстракцией, отражающей потенциальную неограниченность памяти вычислителя. Разумеется, в каждом завершающемся вычислении используется только конечная часть этой памяти — конечное число ячеек. В каждой ячейке записан один символ из конечного алфавита Σ ленты. По ячейкам этой ленты передвигается головка машины. Головка машины представляет собой конечный преобразователь, который в каждый момент времени находится в одном из множества состояний Q. На каждом шаге работы головка читает символ из обозреваемой ячейки, в зависимости от своего внутреннего состояния в соответствии с программой изменяет своё внутреннее состояние и содержимое наблюдаемой ячейки и может либо сдвинуться на одну ячейку вправо или влево, либо остаться на месте.

Дадим более формальное определение.

Определение 152 (Машина Тьюринга)**.** *Машина Тьюринга* $\mathfrak{M}$ *— это четвёрка вида* (Q, Σ, P, q_0)*, в которой:*

1) $Q = \{q_0, \ldots, q_{n-1}\}$, $n \geqslant 1$, *— конечное множество состояний;*
2) $\Sigma = \{a_1, \ldots, a_m\}$, $m \geqslant 1$, *— конечный алфавит, при этом одним из символов* Σ *обязательно является* Λ*;*
3) P *— программа машины Тьюринга;*
4) $q_0 \in Q$ *— начальное состояние.*

Символ Λ *называется пустым (или пробельным).*

Программа машины Тьюринга — это конечное множество команд — слов вида «$q, a \to p, b, s$», где $q, p \in Q$ — состояния, $a, b \in \Sigma$ — символы алфавита, $s \in \{+1, -1, 0\}$ — направление сдвига. При этом для всех $q \in Q$ и $a \in \Sigma$ в P должно быть не более одной команды описанного вида.

В дальнейшем мы всегда считаем, что «слово» — это конечная последовательность символов, не содержащая пустого. Пустой символ мы используем как разделитель между словами. Поэтому, например, в последовательности символов $\Lambda aa\Lambda ba\Lambda\Lambda abba$ записаны три слова: aa, ba и $abba$, из которых первые два разделены одним пустым символом, а следующие — двумя.

Как и для конечных автоматов, программу P можно задавать с помощью таблицы размера $n \times m$, строки которой соответствуют состояниям из Q, а столбцы — символам из входного алфавита Σ, в которой на пересечении строки q и столбца a стоит тройка p, b, s — правая часть команды «$q, a \to p, b, s$».

Определение 153 (Конфигурация машины Тьюринга)**.** *Конфигурацией машины Тьюринга $\mathfrak{M} = (Q, \Sigma, P, q_0)$ называется тройка (q, i, α), где $q \in Q$ — текущее состояние, $i \in \mathbb{Z}$ — целое число, положение головки, $\alpha : \mathbb{Z} \to \Sigma$ — функция, определяющая по номеру ячейки ленты её содержимое. Эту функцию мы тоже будем называть лента.*

На рис. 117 на следующей странице изображена конфигурация $(q_2, 1, \alpha)$ некой машины Тьюринга. В этой конфигурации состояние — q_2, головка обозревает ячейку с номером 1 (номера ячеек ленты указаны сверху), а лента устроена так: $\alpha(1) = \alpha(3) = \alpha(6) = b$, $\alpha(2) = \alpha(5) = a$ и $\alpha(i) = \Lambda$ для всех остальных i.

Определение 154 (Слово, записанное на ленте)**.** *Если лента α содержит только пустые символы: $\alpha(i) = \Lambda$ для всех $i \in \mathbb{Z}$, то говорим, что на ленте записано пустое слово. Иначе записанное на ленте слово — это максимальная последовательность непустых символов вида $\alpha(k)\alpha(k+1)\ldots\alpha(\ell-1)\alpha(\ell)$.*

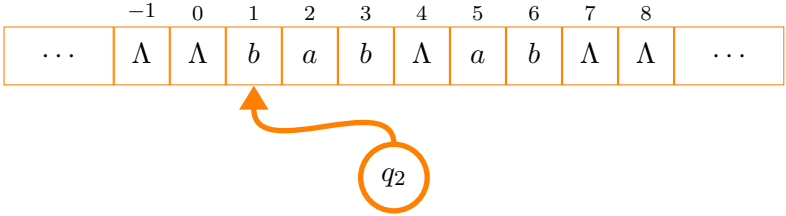


Рис. 117: Пример конфигурации.

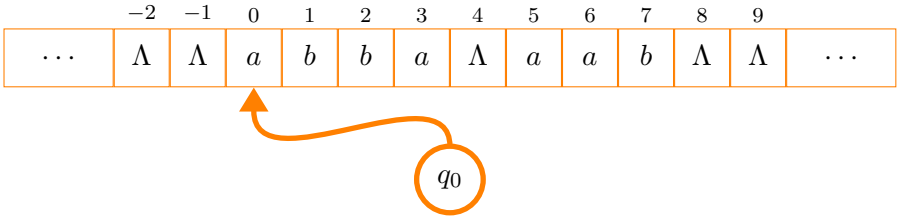


Рис. 118: Пример начальной конфигурации.

Максимальность означает, что эту последовательность нельзя продолжить ни в одну сторону, то есть она окружена пустыми символами. Из определения непосредственно следует, что на ленте может быть записано несколько слов.

Определение 155 (Начальная и заключительная конфигурации)**.** *Пусть* $\mathfrak{M} = (Q, \Sigma, P, q_0)$ *— машина Тьюринга.*

Начальная конфигурация машины $\mathfrak{M}$ *с входными словами* $w_1, \ldots, w_n$ *— это конфигурация вида* $(q_0, 0, \alpha)$*, где на ленте* α*, начиная с нулевой ячейки, записаны слова* $w_1, \ldots, w_n$*, разделённые одним пустым символом.*

Конфигурация вида (q, i, α) *называется заключительной для* $\mathfrak{M}$*, если в программе* P *нет команды вида* $q, \alpha(i) \to \ldots$

На рис. 118 изображена начальная конфигурация $(q_0, 0, \alpha)$ машины $\mathfrak{M} = (Q, \Sigma, P, q_0)$ с двумя входными словами *abba* и *aab*.

Следует заметить, что в начальной конфигурации конец входа легко опознать по двум подряд идущим пустым символам.

Определение 156 (Переход за один шаг)**.** *Пусть дана машина Тьюринга* $\mathfrak{M} = (Q, \Sigma, P, q_0)$*. Конфигурация* $\sigma = (q, i, \alpha)$ *переходит за один шаг в конфигурацию* $\tau = (p, j, \beta)$*, если*

1) *в программе* P *есть команда* $q, a \to p, b, s$*, где* $a = \alpha(i)$*;*

2) $j = i + s$*;*

3) $\beta(i) = b$*,* $\beta(k) = \alpha(k)$ *при* $k \neq i$*.*

Записываем это как обычно: $\sigma \vdash_{\mathfrak{M}} \tau$*.*

Если при выполнении команды некоторый непустой символ заменяется на пустой, то есть команда имеет вид $q, a \to p, \Lambda, s$, то будем говорить, что символ a стирается.

Точно так же как и раньше определяется переход за k шагов $\vdash^k_{\mathfrak{M}}$, за произвольное число шагов $\vdash^*_{\mathfrak{M}}$ и понятие вычисления.

Пример 106. *На рис. 119 на следующей странице сверху представлена следующая конфигурация* σ *некоторой машины* $\mathfrak{M}$*. Состояние машины —* q_3*, головка находится в ячейке с символом* 0*. На ленте записаны слова* 101 *и* 0*, разделённые одним пустым символом.*

Если программа машины $\mathfrak{M}$ *содержит команду* $q_3, 0 \to q_5, 1, +1$*, то после её выполнения конфигурация* σ *перейдёт за один шаг в конфигурацию* τ*:* $\sigma \vdash_{\mathfrak{M}} \tau$*, показанную на этом же рисунке снизу. Состоянием будет* q_5*, символ* 0 *в ранее обозреваемой ячейке поменяется на* 1*, на ленте будут записаны слова* 111 *и* 0*, головка сдвинется на ячейку вправо.*

Если в программе $\mathfrak{M}$ *нет команды вида* $q_5, 1 \to \ldots$*, то последняя конфигурация будет заключительной.*

Определение 157 (Результат вычисления)**.** *Пусть дана машина Тьюринга* $\mathfrak{M} = (Q, \Sigma, P, q_0)$*,* $w_1, \ldots, w_n$ *— слова в алфавите* Σ*.*

Если вычисление $\mathfrak{M}$ *на входе* $w_1, w_2, \ldots, w_n$ *завершается заключительной конфигурацией* τ*, причём в конфигурации* τ *на ленте написано слово* v*, то говорим, что* v *является* результатом вычисления $\mathfrak{M}$ *на входе* $w_1, \ldots, w_n$*.*

В том случае, когда в заключительной конфигурации на ленте записано несколько слов $v_1, \ldots, v_k$*, разделённых одним или несколькими пустыми символами, считаем, что* результатом *является набор* $(v_1, \ldots, v_k)$*.*

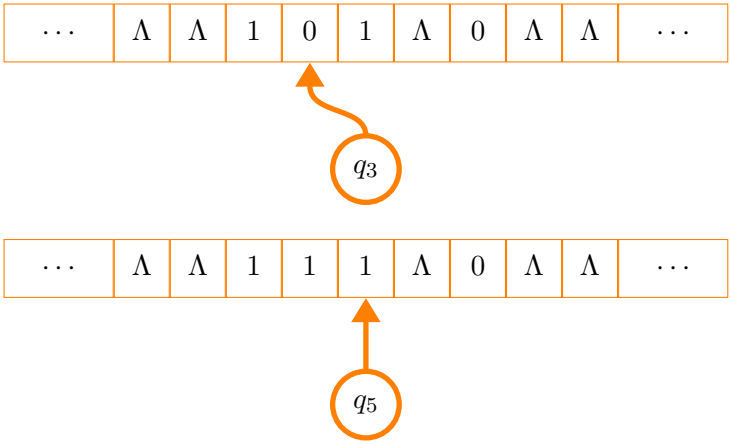


Рис. 119: Выполнение команды $q_3, 0 \to q_5, 1, +1$

Если, например, нижняя конфигурация, показанная на рис. 119 является заключительной, то результат — это пара слов $(111, 0)$.

В силу детерминированности у машины Тьюринга может быть не более одного результата вычисления. Результат может быть неопределён, если в вычислении заключительная конфигурация никогда не образуется. Тогда говорят, что вычисление *зацикливается*.

Определение 158 (Вычислимость по Тьюрингу, эквивалентность машин Тьюринга)**.** *Пусть дана машина Тьюринга* $\mathfrak{M} = (Q, \Sigma, P, q_0)$.

Скажем, что $\mathfrak{M}$ *вычисляет частичную словарную функцию* $f : \Omega_1^* \to \Omega_2^*$ *(с неопределённым числом аргументов), если* $\Omega_1, \Omega_2 \subseteq \Sigma$ *и для всех слов* $w_1, \ldots, w_n \in \Omega_1^*$ *выполнено одно из двух:*

1) $f(w_1, \ldots, w_n)$ *определено и равно* v*, результатом вычисления* $\mathfrak{M}$ *на* $w_1, \ldots, w_n$ *тоже является* v *(*v *может быть одним словом или их набором);*

2) $f(w_1, \ldots, w_n)$ *неопределено и результат вычисления* $\mathfrak{M}$ *на* $w_1, \ldots, w_n$ *неопределён.*

Если словарная функция f *вычисляется некоторой машиной Тьюринга, то она называется вычислимой по Тьюрингу.*

Если машины Тьюринга $\mathfrak{M}$ и $\mathfrak{N}$ вычисляют одну и ту же функцию $f : \Omega_1^* \to \Omega_2^*$, то они Ω_1-эквивалентны.

Из последнего определения сразу получаем.

Следствие 154. *Если машины Тьюринга $\mathfrak{M}$ и $\mathfrak{N}$ являются Ω-эквивалентными, то они будут Ω'-эквивалентными для любого $\Omega' \subseteq \Omega$.*

§ 20.2. Примеры машин Тьюринга

В этом параграфе мы приведём несколько примеров вычисления на машинах Тьюринга словарных функций и проиллюстрируем некоторые общие приёмы, позволяющие создавать машины Тьюринга для решения разнообразных задач.

Пример 107. *Рассмотрим задачу вычисления такой словарной функции в алфавите $\Sigma = \{a, b, c\}$: $f(\varepsilon) = \varepsilon$, $f(xw_1, \ldots, w_n) = x$ для $x \in \Sigma$, то есть её значение равно первому символу первого входного слова. Чтобы выполнить эту задачу, можно реализовать такие действия:*

$$\begin{array}{lll} q_0, a \to q_1, a, +1 & q_1, a \to q_1, \Lambda, +1 & q_2, a \to q_1, \Lambda, +1 \\ q_0, b \to q_1, b, +1 & q_1, b \to q_1, \Lambda, +1 & q_2, b \to q_1, \Lambda, +1 \\ q_0, c \to q_1, c, +1 & q_1, c \to q_1, \Lambda, +1 & q_2, c \to q_1, \Lambda, +1 \\ & q_1, \Lambda \to q_2, \Lambda, +1 & \end{array}$$

Здесь мы пропускаем первый символ (состояние q_0), стираем все символы, начиная со второго (состояние q_1) до тех пор, пока не найдём пустой символ (состояние q_2). Если за ним снова идёт непустой символ, возвращаемся в q_1 и продолжаем стирать, иначе останавливаемся.

Пример 108. *Обращение слова в алфавите $\Sigma = \{a, b, c\}$: $f(w) = w^{-1}$ (значение функции на нескольких аргументах для нас несущественно). Поступим так: будем по одному отделять символы, начиная с конца слова, и составлять из новое слово в обратном порядке (рис. 120 на следующей странице).*

Для этого пропускаем все буквы (состояние q_0). В состоянии q_1 запоминаем последнюю букву и стираем её, для этого мы вводим три отдельных состояния q_2^a, q_2^b и q_2^c. Заметим, что если слово на ленте было пустым, то машина остановится, так как команд вида $q_1, \Lambda \to \ldots$ в программе нет. Записываем эту букву на место образовавшегося пустого символа и сдвигаемся назад к предыдущей букве (состояние q_3). Если буквы закончились,

$$
\begin{array}{lll}
q_0, a \to q_0, a, +1 & q_2^b, a \to q_2^a, b, -1 & q_4^a, a \to q_4^a, a, +1 \\
q_0, b \to q_0, b, +1 & q_2^b, b \to q_2^b, b, -1 & q_4^a, b \to q_4^a, b, +1 \\
q_0, c \to q_0, c, +1 & q_2^b, c \to q_2^c, b, -1 & q_4^a, c \to q_4^a, c, +1 \\
q_0, \Lambda \to q_1, \Lambda, -1 & q_2^b, \Lambda \to q_3, b, -1 & q_4^a, \Lambda \to q_2^a, \Lambda, -1 \\
q_1, a \to q_2^a, \Lambda, 0 & q_2^c, a \to q_2^a, c, -1 & q_4^b, a \to q_4^b, a, +1 \\
q_1, b \to q_2^b, \Lambda, 0 & q_2^c, b \to q_2^b, c, -1 & q_4^b, b \to q_4^b, b, +1 \\
q_1, c \to q_2^c, \Lambda, 0 & q_2^c, c \to q_2^c, c, -1 & q_4^b, c \to q_4^b, c, +1 \\
q_2^a, a \to q_2^a, a, -1 & q_2^c, \Lambda \to q_3, c, -1 & q_4^b, \Lambda \to q_2^b, \Lambda, -1 \\
q_2^a, b \to q_2^b, a, -1 & q_3, a \to q_4^a, \Lambda, +1 & q_4^c, a \to q_4^c, a, +1 \\
q_2^a, c \to q_2^c, a, -1 & q_3, b \to q_4^b, \Lambda, +1 & q_4^c, b \to q_4^c, b, +1 \\
q_2^a, \Lambda \to q_3, a, -1 & q_3, c \to q_4^c, \Lambda, +1 & q_4^c, c \to q_4^c, c, +1 \\
 & & q_4^c, \Lambda \to q_2^c, \Lambda, -1
\end{array}
$$

Рис. 120: Обращение слова

то машина останавливается, иначе эту букву запоминаем (состояния q_4^a, q_4^b и q_4^c) и стираем. Далее переходим к концу нового слова и в последнюю его ячейку записываем запомненную букву, одновременно запоминая то, что мы затёрли (состояния q_2^a, q_2^b и q_2^c). Далее двигаемся назад и продолжаем этот процесс. В результате результат будет сдвинут назад на одну ячейку. Как только мы дойдём до пустого символа, процесс повторится.

Пример 109. *Рассмотрим вычисление такой функции: $f(w_1w_2) = w_1$, если $|w_1| - |w_2| \in \{0, 1\}$ в алфавите $\Sigma = \{a, b, c\}$ (поведение функции на многих входных словах нас по прежнему не интересует). Иначе говоря мы должны стереть вторую половину слова, оставив центральный символ, если длина была нечётной.*

На этом примере мы проиллюстрируем широко применяемый приём: расширение алфавита. В алфавит машины, кроме символов a, b, c и пустого мы включим символы со штрихом: a', b', c'. Машина будет работать так: помечает штрихом первый символ (состояние q_0), идёт к концу слова (состояние q_1), дойдя до пустого символа, она разворачивается и стирает последний символ (состояние q_2), далее движется назад до последнего заштрихованного символа (состояние q_3), делает один шаг вперёд и процесс повторяется. Таким образом, каждый такой цикл штрихует одну букву с начала слова и стирает одну букву с конца. Если очередного символа нет

(в состоянии q_0 обнаруживается Λ или в состоянии q_2 оказывается, что все символы заштрихованы), то машина снимает штрихи со всех оставшихся символов (состояние q_2) и останавливается.

$$
\begin{array}{lll}
q_0, a \to q_1, a', +1 & q_2, a \to q_3, \Lambda, -1 & q_3, a \to q_3, a, -1 \\
q_0, b \to q_1, b', +1 & q_2, b \to q_3, \Lambda, -1 & q_3, b \to q_3, b, -1 \\
q_0, c \to q_1, c', +1 & q_2, c \to q_3, \Lambda, -1 & q_3, c \to q_3, c, -1 \\
q_0, \Lambda \to q_2, \Lambda, -1 & q_2, a' \to q_2, a, -1 & q_3, a' \to q_0, a', +1 \\
q_1, a \to q_1, a, +1 & q_2, b' \to q_2, b, -1 & q_3, b' \to q_0, b', +1 \\
q_1, b \to q_1, b, +1 & q_2, c' \to q_2, c, -1 & q_3, c' \to q_0, c', +1 \\
q_1, c \to q_1, c, +1 & & \\
q_1, \Lambda \to q_2, \Lambda, -1 & &
\end{array}
$$

Пример 110. *Рассмотрим теперь такую задачу: вход содержит двоичную запись двух натуральных чисел, требуется выдать двоичную запись их суммы, то есть сложить два числа в двоичной системе. Считаем, что записи обоих чисел непусты (рис. 121 на следующей странице).*

Здесь мы тоже расширили двоичный алфавит $\Sigma = \{0, 1\}$ штрихованными символами $0'$ и $1'$. Сам алгоритм сложения реализован так. Пометим штрихом последний разряд в первом из чисел (состояния q_0, q_1). Проверим, не закончилось ли второе число (состояния q_2, q_3). Если нет, то дойдём до конца второго числа (состояния q_4, q_5), запомним его последнюю цифру и сотрём её (состояния q_6^0 и q_6^1). Вернёмся к штрихованому разряду первого числа и добавим к нему запомненную цифру. Если возник перенос (состояние q_7), то сдвигаемся назад, прибавляем единицу к предыдущему разряду и продолжаем этот процесс, пока перенос не исчезнет. Затем возвращаемся к штрихованому символу (состояние q_8), сдвигаем штрих на один разряд влево (состояние q_9) и продолжаем процесс. Когда второе число полностью сотрётся (состояние q_{10}), машина сотрёт штрих и остановится.

Одним из часто применяемых способов расширения алфавита является построение многоэтажных символов (см. пример 66 на стр. 287).

Пример 111. *Рассмотрим умножение двоичных чисел. Как и в предыдущей задаче считаем, что изначально на ленте даны в точности два слова.*

Применим метод умножения «столбиком» (рис. 122 на стр. 409). Для этого воспользуемся двухэтажными символами: на нижнем этаже будет накапливаться результат, а на верхнем мы будем постепенно сдвигать первый множитель. Изначально они создаются в состоянии q_0, в этот момент сумма равна нулю, поэтому нижний этаж заполняется одними нулями. Сам алгоритм будет работать так. Проверяем, не закончились ли разряды во

$q_0, 0 \to q_0\ , 0\ , +1$
$q_0, 1 \to q_0\ , 1\ , +1$
$q_0, \Lambda \to q_1\ , \Lambda, -1$
$q_1, 0 \to q_2\ , 0', +1$
$q_1, 1 \to q_2\ , 1', +1$
$q_2, 0 \to q_2\ , 0\ , +1$
$q_2, 1 \to q_2\ , 1\ , +1$
$q_2, \Lambda \to q_3\ , \Lambda, +1$
$q_3, 0 \to q_4\ , 0\ , +1$
$q_3, 1 \to q_4\ , 1\ , +1$
$q_3, \Lambda \to q_{10}, \Lambda, -1$
$q_4, 0 \to q_4\ , 0\ , +1$
$q_4, 1 \to q_4\ , 1\ , +1$
$q_4, \Lambda \to q_5\ , \Lambda, -1$

$q_5, 0 \to q_6^0, \Lambda, -1$
$q_5, 1 \to q_6^1, \Lambda, -1$
$q_6^0, 0 \to q_6^0, 0\ , -1$
$q_6^0, 1 \to q_6^0, 1\ , -1$
$q_6^0, \Lambda \to q_6^0, \Lambda, -1$
$q_6^0, 0' \to q_8, 0', 0$
$q_6^0, 1' \to q_8, 1', 0$
$q_6^1, 0 \to q_6^1, 0\ , -1$
$q_6^1, 1 \to q_6^1, 1\ , -1$
$q_6^1, \Lambda \to q_6^1, \Lambda, -1$
$q_6^1, 0' \to q_8, 1', 0$
$q_6^1, 1' \to q_7, 0', -1$

$q_7, 1 \to q_7\ , 0\ , -1$
$q_7, 0 \to q_8\ , 1\ , +1$
$q_7, \Lambda \to q_8\ , 1\ , +1$
$q_8, 0 \to q_8\ , 0\ , +1$
$q_8, 1 \to q_8\ , 1\ , +1$
$q_8, 0' \to q_9\ , 0\ , -1$
$q_8, 1' \to q_9\ , 1\ , -1$
$q_9, 0 \to q_2\ , 0', +1$
$q_9, 1 \to q_2\ , 1', +1$
$q_9, \Lambda \to q_2\ , 0', +1$
$q_{10}, 0 \to q_{10}, 0\ , -1$
$q_{10}, 1 \to q_{10}, 1\ , -1$
$q_{10}, \Lambda \to q_{10}, \Lambda, -1$
$q_{10}, 0' \to q_{11}, 0\ , 0$
$q_{10}, 1' \to q_{11}, 1\ , 0$

Рис. 121: Сложение двоичных чисел.

втором множителе (состояния q_1 и q_2). Если нет, то ищем самый младший из них и стираем его (состояния q_2 и q_3). Если он равен нулю (состояние q_4^0), то просто сдвигаем первый множитель на один разряд влево (состояния q_5, q_5^0 и q_5^1). Если единице (состояние q_4^0) — добавляем его к сумме (состояния q_6 и q_7), а затем (состояние q_8) тоже сдвигаем.

Когда все разряды второго множителя закончатся (состояние q_9), остаётся только переписать полученный результат с нижнего этажа в виде обычных, «одноэтажных», символов (состояние q_{10}).

Для краткости записи мы применили следующий приём. С помощью букв a, b, c обозначены произвольные символы из множества $\{0, 1\}$. Поэтому, например, с помощью $q_5^a, \left[\!\!\begin{bmatrix} b \\ c \end{bmatrix}\!\!\right] \to q_5^b, \left[\!\!\begin{bmatrix} a \\ c \end{bmatrix}\!\!\right], -1$ записано сразу восемь различных команд, получающихся подстановкой вместо a, b, c всевозможных комбинаций нулей и единиц. При этом, естественно, всюду одна и та же буква в этой команде должна быть заменена одним и тем же символом.

$$
\begin{array}{llll}
q_0, & a & \to q_0, & \left[\!\!\begin{smallmatrix} a \\ 0 \end{smallmatrix}\!\!\right], +1 \\
q_0, & \left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right] & \to q_0, & \left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right], +1 \\
q_0, & \Lambda & \to q_1, & \Lambda, +1 \\
q_1, & a & \to q_2, & a, +1 \\
q_1, & \Lambda & \to q_9, & \Lambda, -1 \\
q_2, & a & \to q_2, & a, +1 \\
q_2, & \Lambda & \to q_3, & \Lambda, -1 \\
q_3, & a & \to q_4^a, & \Lambda, -1 \\
q_4^a, & b & \to q_4^a, & b, -1 \\
q_4^0, & \Lambda & \to q_5, & \Lambda, -1 \\
q_4^1, & \Lambda & \to q_6, & \Lambda, -1
\end{array}
\qquad
\begin{array}{llll}
q_5, & \left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right] & \to q_5^a, & \left[\!\!\begin{smallmatrix} 0 \\ b \end{smallmatrix}\!\!\right], -1 \\
q_5^a, & \left[\!\!\begin{smallmatrix} b \\ c \end{smallmatrix}\!\!\right] & \to q_5^b, & \left[\!\!\begin{smallmatrix} a \\ c \end{smallmatrix}\!\!\right], -1 \\
q_5^0, & \Lambda & \to q_0, & \Lambda, +1 \\
q_5^1, & \Lambda & \to q_0, & \left[\!\!\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\!\!\right], +1 \\
q_8, & \left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right] & \to q_8, & \left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right], +1 \\
q_8, & \Lambda & \to q_5, & \Lambda, -1 \\
q_9, & \Lambda & \to q_{10}, & \Lambda, -1 \\
q_{10}, & \left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right] & \to q_{10}, & b, -1
\end{array}
\qquad
\begin{array}{llll}
q_6, & \left[\!\!\begin{smallmatrix} 0 \\ a \end{smallmatrix}\!\!\right] & \to q_6, & \left[\!\!\begin{smallmatrix} 0 \\ a \end{smallmatrix}\!\!\right], -1 \\
q_6, & \left[\!\!\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\!\!\right] & \to q_6, & \left[\!\!\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\!\!\right], -1 \\
q_6, & \left[\!\!\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}\!\!\right] & \to q_7, & \left[\!\!\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\!\!\right], -1 \\
q_6, & \Lambda & \to q_8, & \Lambda, +1 \\
q_7, & \left[\!\!\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\!\!\right] & \to q_6, & \left[\!\!\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\!\!\right], -1 \\
q_7, & \left[\!\!\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\!\!\right] & \to q_7, & \left[\!\!\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\!\!\right], -1 \\
q_7, & \left[\!\!\begin{smallmatrix} 1 \\ a \end{smallmatrix}\!\!\right] & \to q_7, & \left[\!\!\begin{smallmatrix} 1 \\ a \end{smallmatrix}\!\!\right], -1 \\
q_7, & \Lambda & \to q_8, & \left[\!\!\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\!\!\right], +1
\end{array}
$$

Рис. 122: Умножение двоичных чисел.

§ 20.3. Стандартная заключительная конфигурация

В дальнейшем нам часто придётся выполнять такую операцию: запустить одну машину Тьюринга $\mathfrak{M}$, а после того, как $\mathfrak{M}$ остановится, — запустить вторую $\mathfrak{N}$, используя в качестве входа для $\mathfrak{N}$ выход $\mathfrak{M}$. Чтобы это можно было делать, заключительная конфигурация машины $\mathfrak{M}$ должна быть применима в качестве начальной конфигурации машины $\mathfrak{N}$: головка располагалась бы в нулевой ячейке и, начиная с нулевой же ячейки, был записан результат. Если результат состоит из нескольких слов, то они должны быть разделены в точности одним пустым символом. Эти условия называются стандартной заключительной конфигурацией.

Определение 159 (Стандартная заключительная конфигурация)**.** *Назовём с т а н д а р т н о й заключительную конфигурацию вида*

$(q, 0, \alpha)$, если слова на ленте α записаны, начиная с нулевой ячейки, и разделены в точности одним пустым символом.

Говорим, что машина Тьюринга $\mathfrak{M}$ имеет стандартную заключительную конфигурацию, если для всех начальных конфигураций σ, на которых машина $\mathfrak{M}$ останавливается, заключительная конфигурация τ, в которую σ перейдёт, будет стандартной.

Таким образом, определение означает, что если машина $\mathfrak{M}$ остановится, то обязательно в стандартной заключительной конфигурации.

Теорема 155. *Для всякой машины Тьюринга $\mathfrak{M} = (Q, \Sigma, P, q_0)$ можно построить Σ-эквивалентную машину Тьюринга $\mathfrak{N}$, имеющую стандартную заключительную конфигурацию.*

Доказательство. Основная идея доказательства будет заключаться в следующем. Мы отметим символ в нулевой ячейке при помощи звёздочки, а во всякой другой ячейке, где побывала головка машины, — штрихом. По завершении работы мы должны будем просмотреть всю заштрихованную область и при необходимости сместить результат так, чтобы он начинался в ячейке со звёздочкой.

Приступим к построению машины $\mathfrak{N} = (Q', \Sigma', P', q_0')$. Её алфавит, как мы уже отметили, кроме старых символов будет заключать их копии со штрихами и звёздочками:

$$\Sigma' = \{a, a', a^* : a \in \Sigma\}.$$

Первая группа команд программы P' новой машины $\mathfrak{N}$ предназначена для расстановки начальных меток: звёздочки в первой ячейке и штрихов — во всех остальных ячейках входных слов:

$$\begin{array}{lll} q_0', a \to q_1', a^*, +1 & q_2', b \to q_1', b', +1 & q_3', a' \to q_3', a', -1 \\ q_1', b \to q_1', b', +1 & q_2', \Lambda \to q_3', \Lambda, -1 & q_3', a^* \to q_0, a^*, 0 \\ q_1', \Lambda \to q_2', \Lambda', +1 & & \end{array}$$

Здесь $a \in \Sigma$ — любой символ исходного алфавита Σ, а $b \in \Sigma \setminus \{\Lambda\}$ — любой непустой символ Σ. В этом фрагменте программы мы помечаем нулевую ячейку звёздочкой, после движемся вперёд, помечая все символы штрихом, пока не дойдём до двух пустых символов подряд,

что, как мы уже отмечали, является признаком конца входных данных. Проделав все эти манипуляции, мы возвращаемся в начальную ячейку и переходим в старое начальное состояние машины $\mathfrak{M}$.

Чтобы промоделировать работу старой машины $\mathfrak{M}$ с описанными в начале доказательства изменениями, мы для каждой команды $\mathfrak{M}$ вида $q, a \to p, b, s$ добавим в программу P' новой машины следующие три команды:

$$q, a \to p, b', s; \quad q, a' \to p, b', s; \quad q, a^* \to p, b^*, s.$$

Первая из них отмечает штрихом ячейку, если она ещё не заштрихована, то есть делает пометку, что головка эту ячейку посетила. Вторая — оставляет заштрихованную ячейку заштрихованной, а третья — сохраняет звёздочку.

Нетрудно понять, что благодаря этим командам новая машина будет работать так же как и старая, если не считать новых пометок.

После того, как старая машина $\mathfrak{M}$ остановилась, новая должна будет заняться сдвигом результата. Для этого в программу P' добавляем всевозможные команды вида

$$q, a \to p_1, a, 0; \quad q, a' \to p_1, a', 0; \quad q, a^* \to p_1, a^*, 0,$$

если команды вида $q, a \to \ldots$ не было в программе $\mathfrak{M}$. Далее нужно перейти к началу заштрихованной части ленты (состояние p_1) и, двигаясь вправо, искать положение выходного слова, параллельно стирая штрихи с пустых символов (состояние p_2). Возможны три варианта: первое выходное слово начинается левее (состояния p_3^a), правее (состояние p_6) или в точности в начальной ячейке (состояние p_5). В первом случае мы сдвигаем содержимое ленты вправо, проверяя, не оказался ли первый непустой символ в ячейке со звёздочкой, после чего переходим в состояние p_5. Третий вариант (начиная с состояния

p_6) предлагается рассмотреть самостоятельно (задача 289 на стр. 433):

$$
\begin{array}{ll}
p_1, a' \to p_1, a', -1 & p_3^a, c' \to p_3^c, a', +1 \\
p_1, a^* \to p_1, a^*, -1 & p_3^a, c^* \to p_3^c, a^*, +1 \\
p_1, \Lambda \to p_2, \Lambda, +1 & p_3^a, \Lambda \to p_4, a', -1 \\
p_2, \Lambda' \to p_2, \Lambda, +1 & p_4, a' \to p_4, a', -1 \\
p_2, b^* \to p_5, b^*, +1 & p_4, a^* \to p_4, a^*, -1 \\
p_2, \Lambda^* \to p_6, \Lambda^*, +1 & p_4, a \to p_2, a, +1 \\
p_2, b' \to p_3^b, \Lambda, +1 &
\end{array}
$$

Как и раньше с помощью a и c обозначены любые символы Σ (они могут и совпадать друг с другом), а с помощью b — любой непустой.

После того, как начало выхода совпало с нулевой ячейкой, необходимо перенести остальные выходные слова, чтобы они были разделены только одним пустым символом. Для этого мы движемся вперёд по ленте (состояния p_5 и p_7) и, если обнаруживаем два пустых символа подряд (состояние p_8), то сдвигаем правую часть ленты на одну ячейку влево (состояния p_9 и p_9^a), возвращаемся к началу ленты (состояние p_{10}) и продолжаем этот процесс заново. Если таких мест на ленте не обнаружено, то переходим к последнему этапу: стираем штрихи, звёздочку (состояние p_{11}) и останавливаемся.

$$
\begin{array}{ll}
p_5, b' \to p_5, b', +1 & p_9, a' \to p_9^a, \Lambda, -1 \\
p_5, \Lambda' \to p_7, \Lambda', +1 & p_9^a, b' \to p_9^b, a', -1 \\
p_5, \Lambda \to p_{11}, \Lambda, -1 & p_9^b, \Lambda' \to p_9^\Lambda, b', -1 \\
p_7, b' \to p_5, b', +1 & p_9^\Lambda, \Lambda' \to p_{10}, \Lambda', -1 \\
p_7, \Lambda' \to p_8, \Lambda', +1 & p_{10}, a' \to p_{10}, a', -1 \\
p_7, \Lambda \to p_{11}, \Lambda, -1 & p_{10}, a^* \to p_5, a^*, +1 \\
p_8, a' \to p_8, a', +1 & p_{11}, a' \to p_{11}, a, -1 \\
p_8, \Lambda \to p_9, \Lambda, -1 & p_{11}, a^* \to p_{11}, a, 0
\end{array}
$$

Поскольку головка остановилась в той ячейке, где стояла звёздочка, то есть в начальной, то теперь машины Тьюринга оказалась в стандартной заключительной конфигурации. □

§ 20.4. Односторонние машины Тьюринга

С интуитивной точки зрения односторонней считается машина Тьюринга, головка которой никогда ни на каком входе не может сдвинуться в область отрицательных ячеек.

К сожалению, определить, обладает ли машина Тьюринга таким свойством алгоритмически невозможно (это будет доказано в следующей главе, теорема 170 на стр. 444). Поэтому мы будем называть односторонними машины Тьюринга, в которых такое ограничение явно прописано.

В любых машинах Тьюринга алфавит ленты обязательно содержит пустой символ Λ. Для односторонних машин мы добавим ещё один специальный символ # — *ограничитель*, которым будет отмечаться нулевая ячейка.

Определение 160 (Односторонняя машина Тьюринга)**.** *Машина Тьюринга* $\mathfrak{M} = (Q, \Sigma, P, q_0)$ *называется односторонней, если*

1) *алфавит ленты* Σ *содержит символ* #*;*
2) *в программе* P *нет команд ни одного из трёх следующих видов:* «$q, \# \to p, \#, -1$»*;* «$q, \# \to p, a, s$»*;* «$q, a \to p, \#, s$» *ни для каких* $q, p \in Q$, $a \in \Sigma \setminus \{\#\}$, $s \in \{+1, -1, 0\}$.

Таким образом, односторонняя машина не может сдвинуть головку левее #*, не может стереть* # *и не может записать его в ячейку, где его не было.*

Подобно тому, как при рассмотрении произвольных машин Тьюринга мы считали, что слова не могут содержать пустого символа Λ, теперь мы добавим ещё одно ограничение: слова не могут содержать символ #. Таким образом, слово, записанное на ленте, должно быть ограничено с обеих сторон символами Λ или #.

В начальной конфигурации входные слова должны быть записаны сразу после #, то есть начиная с ячейки с первым номером. Например, начальная конфигурация с входными словами *aab* и *bbb* выглядит, как показано на рис. 123 на следующей странице. К стандартной заключительной конфигурации односторонней машины предъявляются те

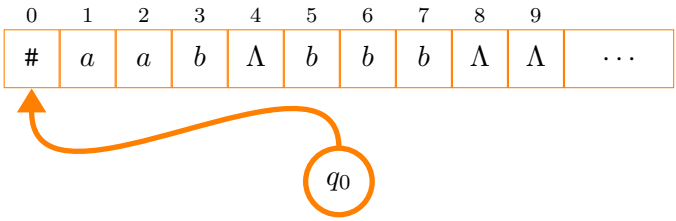


Рис. 123: Начальная конфигурация односторонней машины Тьюринга.

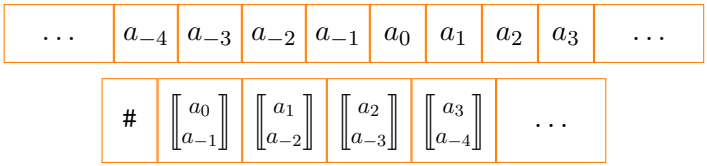


Рис. 124: «Упаковка» двусторонней ленты в двухэтажную

же требования: выходные слова должны быть записаны сразу после #, разделены одним пустым символом, а головка должна находиться в нулевой ячейке.

Теорема 156. *Для всякой машины Тьюринга* $\mathfrak{M} = (Q, \Sigma, P, q_0)$ *существует* Σ*-эквивалентная односторонняя машина Тьюринга* $\mathfrak{N}$*, имеющая стандартную заключительную конфигурацию.*

Доказательство. Согласно теореме 155 на стр. 410 мы можем без ограничения общности полагать, что исходная машина $\mathfrak{M}$ имеет стандартную заключительную конфигурацию. Для построения машины $\mathfrak{N} = (Q', \Sigma', P', q_0)$ применим механизм многоэтажных символов. Мы «сложим» ленту исходной машины пополам так, что ячейки с отрицательными номерами окажутся на нижнем этаже, а с неотрицательными на верхнем (рис. 124). Таким образом, ячейка с номером i будет содержать пару символов: сверху — из ячейки с номером $i + 1$, снизу — из ячейки $-i$.

Алфавит Σ' будет состоять из символов вида $\left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right]$, где $a, b \in \Sigma$:

$$\Sigma' = \left\{ \left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right] : a, b \in \Sigma \right\} \cup \{\#\}.$$

При этом мы отождествим двухэтажный символ с нижней «пустышкой» с верхним этажом: $\left[\!\!\begin{smallmatrix} a \\ \Lambda \end{smallmatrix}\!\!\right] = a$. В частности, $\left[\!\!\begin{smallmatrix} \Lambda \\ \Lambda \end{smallmatrix}\!\!\right] = \Lambda$. Это означает, что когда мы, например, пишем «$q, \left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right] \to p, \left[\!\!\begin{smallmatrix} d \\ c \end{smallmatrix}\!\!\right], 0$», то подразумеваем сразу четыре типа команд: «$q, \left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right] \to p, \left[\!\!\begin{smallmatrix} d \\ c \end{smallmatrix}\!\!\right], 0$» для непустых b и c, «$q, \left[\!\!\begin{smallmatrix} a \\ b \end{smallmatrix}\!\!\right] \to p, d, 0$» для пустого c и непустого b, «$q, a \to p, \left[\!\!\begin{smallmatrix} d \\ c \end{smallmatrix}\!\!\right], 0$» для пустого b и непустого c, «$q, a \to p, d, 0$» для пустых b и c.

Каждое состояние $q \in Q$ старой машины «раздвоится»: одно состояние q^+ будет предназначено для работы в ячейках с неотрицательными номерами (то есть с верхним этажом), второе состояние q^- — с отрицательными (с нижним этажом):

$$Q' = \{q^+, q^- : q \in Q\} \cup \{q_0\}.$$

Таким образом, в состояниях q^+ мы должны рассматривать верхний этаж, а в состояниях q^- — нижний.

Рассмотрим, как будут строиться команды новой машины $\mathfrak{N}$.

Если P содержит команду $q, a \to p, b, 0$, то в новую программу P' добавляем две группы команд:

$$q^+, \left[\!\!\begin{smallmatrix} a \\ c \end{smallmatrix}\!\!\right] \to p^+, \left[\!\!\begin{smallmatrix} b \\ c \end{smallmatrix}\!\!\right], 0; \quad q^-, \left[\!\!\begin{smallmatrix} c \\ a \end{smallmatrix}\!\!\right] \to p^-, \left[\!\!\begin{smallmatrix} c \\ b \end{smallmatrix}\!\!\right], 0$$

для всех символов $c \in \Sigma$.

Если P содержит команду $q, a \to p, b, +1$, то в новую программу P' добавляем три группы команд:

$$q^+, \left[\!\!\begin{smallmatrix} a \\ c \end{smallmatrix}\!\!\right] \to p^+, \left[\!\!\begin{smallmatrix} b \\ c \end{smallmatrix}\!\!\right], +1; \quad q^-, \left[\!\!\begin{smallmatrix} c \\ a \end{smallmatrix}\!\!\right] \to p^-, \left[\!\!\begin{smallmatrix} c \\ b \end{smallmatrix}\!\!\right], -1; \quad p^-, \# \to p^+, \#, +1$$

для всех символов $c \in \Sigma$. Третья команда проверяет, не вышла ли головка из отрицательной части ленты в положительную, если это

произошло, то состояние меняет знак: мы от работы с нижним этажом переходим к работе в верхним. Заметим, что при работе с нижним этажом направление сдвига меняется на противоположное, так как нижний этаж содержит отрицательную часть ленты, записанную в обратном порядке.

По аналогии строятся команды программы P', если в исходной программе P был сдвиг влево $q, a \to p, b, -1$:

$$q^+, \left[\!\!\begin{matrix} a \\ c \end{matrix}\!\!\right] \to p^+, \left[\!\!\begin{matrix} b \\ c \end{matrix}\!\!\right], -1; \quad q^-, \left[\!\!\begin{matrix} c \\ a \end{matrix}\!\!\right] \to p^-, \left[\!\!\begin{matrix} c \\ b \end{matrix}\!\!\right], +1; \quad p^+, \# \to p^-, \#, +1$$

для всех символов $c \in \Sigma$. Теперь мы проверяем, не вышла ли головка из положительной части ленты в отрицательную, и меняем знак в случае необходимости, чтобы перейти от верхнего этажа к нижнему.

Кроме того, нужно будет добавить команду, «запускающую» работу с двумя этажами: $q_0, \# \to q_0^+, \#, +1$. Заметим, что поскольку мы отождествили двухэтажные символы с пустым нижним этажом с одноэтажными, то в начальной конфигурации можно считать входные слова расположенными на верхнем этаже, что соответствует их расположению на двусторонней ленте в ячейках с неотрицательными номерами, отрицательная часть ленты при этом пуста.

Аналогично с заключительной конфигурацией, при остановке машины $\mathfrak{M}$ левая часть ленты пуста, поэтому все двухэтажные символы содержат пустой нижний этаж, можно считать, что это обычные символы, которые образуют выходное слово. Так как машина $\mathfrak{M}$ имела стандартную заключительную конфигурацию, то выходные слова записаны, начиная с нулевой ячейки, и разделены одним пустым символом. Но тогда в машине $\mathfrak{N}$ выходные слова записаны сразу после # и тоже разделены одним пустым символом. Следовательно, $\mathfrak{N}$ имеет стандартную заключительную конфигурацию. □

Мы уже отмечали, что стандартная заключительная конфигурация позволяет последовательно применять несколько машин Тьюринга: результат работы предыдущей является исходными данными для следующей. Односторонние машины Тьюринга можно использовать, когда нужно вычислить что-то, не изменяя входные данные.

Пример 112. *Допустим, что у нас есть односторонние машины Тьюринга* $\mathfrak{M}_+$ *и* $\mathfrak{M}_\times$*, которые выполняют сложение и умножение двух чисел. Покажем, как из них построить машину Тьюринга* $\mathfrak{M}$ *для вычисления значения выражения* $xy + 2x + y$.

Сама машина $\mathfrak{M}$ *односторонней не будет, поэтому символ* #*, который мы будем использовать не имеет для* $\mathfrak{M}$ *какого-либо «особого» значения. Последовательность действий* $\mathfrak{M}$ *будет следующей:*

1) *Пусть в начальный момент лента имеет вид* $\ldots \Lambda x \Lambda y \Lambda \ldots$.
2) *Создадим копии* x *и* y*, отделив их от исходных* x *и* y *символом* #*:* $\ldots \Lambda x \Lambda y \# x \Lambda y \Lambda \ldots$.
3) *В правой части ленты запустим машину* $\mathfrak{M}_\times$*. В силу её односторонности она не сможет зайти левее* #*, поэтому исходные* x *и* y *не будут испорчены. Следовательно, при завершении работы* $\mathfrak{M}$ *лента будет иметь вид* $\ldots \Lambda x \Lambda y \# z \Lambda \ldots$*, где* $z = x \times y$.
4) *Перенесём* # *на место предыдущего пробела:* $\ldots \Lambda x \# y \Lambda z \Lambda \ldots$*, где* $z = x \times y$.
5) *В правой части ленты запустим машину* $\mathfrak{M}_+$*. Так как она односторонняя, то* x *левее* # *не будет в ходе её работы испорчен, а после завершения работы мы получим* $\ldots \Lambda x \# u \Lambda \ldots$*, где* $u = y + xy$.
6) *Ещё раз скопируем* x*:* $\ldots \Lambda x \# u \Lambda x \Lambda \ldots$*, и ещё раз запустим* $\mathfrak{M}_+$*, получим* $\ldots \Lambda x \# v \Lambda \ldots$*, где* $v = xy + x + y$.
7) *Теперь передвинем* # *перед* x*:* $\ldots \# x \Lambda v \Lambda \ldots$ *и снова запустим* $\mathfrak{M}_+$*, в результате получим* $\ldots \# w \Lambda \ldots$*, где* $w = xy + 2x + y$.
8) *Последнее, что остаётся — стереть* #.

Используя описанную в примере технику, на машинах Тьюринга можно реализовать классические программистские конструкции: ветвление и цикл.

Пусть, например, требуется выполнить **Если** A **то** B **Иначе** C. Для этого мы можем сначала сконструировать одностороннюю машину Тьюринга $\mathfrak{M}_A$, проверяющую условие A. Всё ветвление тогда реализуется следующим образом: создаём справа копию данных, отделяя её от оригинала символом #, запускаем на этой копии машину $\mathfrak{M}_A$. Так как она односторонняя, то оригинальные данные, расположенные левее # не будут изменены. После остановки $\mathfrak{M}$ нужно запомнить её

результат (например, 0 или 1), стереть его, и в зависимости от этого результата запустить $\mathfrak{M}_B$ или $\mathfrak{M}_C$ на оригинальных данных.

Аналогично реализуется цикл типа **Пока** A **выполнять** B. Создаём справа копию данных, отделяя её символом #, запускаем на этой копии одностороннюю машину $\mathfrak{M}_A$, после её остановки запоминаем результат и стираем его. Если результат равен 1, то запускаем $\mathfrak{M}_B$ и всё повторяем заново, иначе останавливаемся.

§ 20.5. Вычислимость: программная и по Тьюрингу

Сравним возможности программ с метками и машин Тьюринга. Основным результатом данного параграфа будет то, что обе этих модели алгоритма позволяют вычислять одни и те же функции, то есть их возможности совпадают. В качестве следствия будет получен ещё один существенный результат об ограничении алфавита.

Сначала покажем, как вычислить на машине Тьюринга функцию, которую можно вычислить при помощи какой-то программы. Сложность заключается в том, что программы работают с числами, а машины Тьюринга — со словами. Поэтому чтобы преобразовать программу в машину Тьюринга потребуется какой-то способ кодирования чисел словами.

Мы будем считать, что зафиксирована некоторая система счисления, то есть разнозначная функция, отображающая натуральные числа в слова какого-то алфавита. Эта система может быть k-ичной, где $k \geqslant 2$ (например, двоичной, десятичной и т. д.), часто применяемой в повседневной жизни и в программировании, либо, в простейшем случае, — унарной.

Унарная система счисления — это способ записи числа n соответствующим количеством палочек. Трудность заключается в том, что число 0 при таком способе будет выглядеть как пустое слово ε, что может породить неоднозначность. Поэтому мы условимся считать, что в унарной системе натуральное число n представляется в виде слова из $n + 1$ палочки, например, число 5 кодируется словом «||||||».

Определение 161 (Вычислимость по Тьюрингу арифметических функций)**.** *Пусть зафиксирована некоторая система счисления π. Скажем, что машина Тьюринга $\mathfrak{M}$ вычисляет частичную арифметическую функцию $f : \omega^n \to \omega$, если для всех наборов значений аргументов $a_1, \ldots, a_n \in \omega$ выполнено одно из двух:*

1) *$f(a_1, \ldots, a_n)$ определено и равно b, и $\mathfrak{M}$ останавливается на входе $(\pi(a_1), \ldots, \pi(a_n))$, а выходным словом при этом является $\pi(b)$;*
2) *$f(a_1, \ldots, a_n)$ неопределено и $\mathfrak{M}$ на входе $(\pi(a_1), \ldots, \pi(a_n))$ не останавливается.*

Арифметическая (частичная) функция $f(x_1, \ldots, x_n)$ вычислима по Тьюрингу, если существует машина Тьюринга, которая эту функцию вычисляет.

Теорема 157. *Каждая программно вычислимая функция f вычислима по Тьюрингу без расширения алфавита.*

Доказательство. Пусть программа Π вычисляет функцию f. Будем считать, что эта программа содержит переменные $x_1, \ldots, x_k$, из которых $x_1, \ldots, x_n$ являются входными, а x_1 — ещё и выходной. Начальной меткой Π пусть является α_0, а единственной заключительной — α_m.

Схема доказательства будет заключаться в следующем. Напомним, что каждая конфигурация (α, σ) программы Π содержит текущую метку α и функцию σ, определяющую значение переменных. Мы создадим такую одностороннюю машину Тьюринга $\mathfrak{M}$, что каждой конфигурации (α, σ) программы Π будет соответствовать конфигурация машины Тьюринга вида $(q_\alpha, 0, \varkappa)$, где $\varkappa$ — лента, на которой последовательно записаны слова $\pi(\sigma(x_1))$, …, $\pi(\sigma(x_k))$, разделённые одним пустым символом. Иначе говоря, после # идёт последовательная запись значений переменных x_i, разделённых с помощью Λ.

В качестве системы счисления мы выберем унарную, как более простую. Однако и использование других систем больши́х сложностей не вызывает (задача 294 на стр. 433).

Работа машины Тьюринга будет состоять из трёх этапов: инициализация, моделирование программы Π и завершение работы.

На первом этапе у нас имеется вход, где записаны n чисел. Поскольку конфигурация должна содержать k чисел, то мы должны дописать $r = k - n$ нулей:

$$
\begin{array}{lll}
q_0, \# \to q_0, \#, +1 & q_1', \Lambda \to q_2, \Lambda, +1 & \dots \\
q_0, | \to q_0, |, +1 & q_2, \Lambda \to q_2', |, +1 & q_r, \Lambda \to q_r', |, -1 \\
q_0, \Lambda \to q_1, \Lambda, +1 & q_2', \Lambda \to q_3, \Lambda, +1 & q_r', \Lambda \to q_r', \Lambda, -1 \\
q_1, | \to q_0, |, +1 & q_3, \Lambda \to q_3', |, +1 & q_r', | \to q_r', |, -1 \\
q_1, \Lambda \to q_1', |, +1 & \dots & q_r', \# \to q_{\alpha_0}, |, -1
\end{array}
$$

Здесь мы в состояниях q_i добавляем очередную i-ю палочку, а в состояниях q_i' — пропускаем один пустой символ. Добавив r палочек, мы возвращаемся в начальную ячейку, чтобы приступить ко второму этапу.

Чтобы промоделировать работу программы Π, мы для каждого оператора o программы с метками создадим часть P_o программы машины Тьюринга $\mathfrak{M}$, которая будет моделировать работу o. Напомним, что, согласно теореме 139 на стр. 376, мы можем считать, что оператор o имеет один из трёх видов. Покажем, как в каждом из этих случаев построить P_o.

Если $o = \alpha\ x_i \leftarrow s(x_i);\ \beta$, то мы из начальной ячейки должны сдвинуться к началу i-го числа, раздвинуть ленту, чтобы освободить одну ячейку, и записать в эту ячейку палочку. Следующий фрагмент $P_{\alpha i}$ предназначен для поиска начала записи i-го числа:

$$
\begin{array}{ll}
q_\alpha, \# \to q_\alpha^1, \#, +1 & q_\alpha^2, | \to q_\alpha^2, |, +1 \\
q_\alpha^1, | \to q_\alpha^1, |, +1 & q_\alpha^2, \Lambda \to q_\alpha^3, \Lambda, +1 \\
q_\alpha^1, \Lambda \to q_\alpha^2, \Lambda, +1 & \dots \\
 & q_\alpha^{i-1}, \Lambda \to q_\alpha^i, \Lambda, +1
\end{array}
$$

Далее вставляем палочку, сдвигаем остаток ленты вправо и возвращаемся назад в начальную ячейку:

$$
\begin{array}{ll}
q^i_\alpha, | \ \to q^{|}_\alpha, | \ , +1 & q^\Lambda_\alpha, \Lambda \to q^\#_\alpha, \Lambda, -1 \\
q^{|}_\alpha, | \ \to q^{|}_\alpha, | \ , +1 & q^\#_\alpha, \Lambda \to q^\#_\alpha, \Lambda, -1 \\
q^{|}_\alpha, \Lambda \to q^\Lambda_\alpha, | \ , +1 & q^\#_\alpha, | \ \to q^\#_\alpha, | \ , -1 \\
q^\Lambda_\alpha, | \ \to q^{|}_\alpha, \Lambda, +1 & q^\#_\alpha, \# \to q_\beta, \# \, , 0
\end{array}
$$

Для оператора $o = \alpha\ x_i \leftarrow d(x_i);\ \beta$ поступаем аналогично, только не добавляем, а удаляем палочку. Сначала добавляем такой же как и раньше фрагмент $P_{\alpha i}$ для перехода к записи i-го числа. После этого удаляем палочку и переходим к следующей ячейке. Если палочка была единственной, то возвращаем всё на место, а в противном случае сдвигаем ленту влево:

$$
\begin{array}{lll}
q^i_\alpha, | \ \to q'_\alpha, \Lambda, +1 & q^r_\alpha, \Lambda \to q^e_\alpha, \Lambda, +1 & q^\Lambda_\alpha, | \ \to q^{|}_\alpha, \Lambda, -1 \\
q'_\alpha, \Lambda \to q''_\alpha, \Lambda, -1 & q^e_\alpha, | \ \to q^r_\alpha, | \ , +1 & q^\Lambda_\alpha, \Lambda \to q^\#_\alpha, \Lambda, -1 \\
q'_\alpha, | \ \to q^r_\alpha, | \ , +1 & q^e_\alpha, \Lambda \to q^\Lambda_\alpha, \Lambda, -1 & q^\Lambda_\alpha, \# \to q_\beta, \# \, , 0 \\
q''_\alpha, \Lambda \to q^\#_\alpha, | \ , -1 & q^{|}_\alpha, | \ \to q^{|}_\alpha, | \ , -1 & q^\#_\alpha, \Lambda \to q^\#_\alpha, \Lambda, -1 \\
q^r_\alpha, | \ \to q^r_\alpha, | \ , +1 & q^{|}_\alpha, \Lambda \to q^\Lambda_\alpha, | \ , -1 & q^\#_\alpha, | \ \to q^\#_\alpha, | \ , -1 \\
 & & q^\#_\alpha, \# \to q_\beta, \# \, , 0
\end{array}
$$

Наконец, для оператора $o = \alpha$ **Если** x_i **то** β **Иначе** γ нужно только проверить значение i-й переменной и перейти в одно из состояний q_β или q_γ. В начале добавляется такой же фрагмент $P_{\alpha i}$, а затем выполняем проверку:

$$
\begin{array}{lll}
q^i_\alpha, | \ \to q'_\alpha, | \ , +1 & q^\beta_\alpha, \Lambda \to q^\beta_\alpha, \Lambda, -1 & q^\gamma_\alpha, \Lambda \to q^\gamma_\alpha, \Lambda, -1 \\
q'_\alpha, | \ \to q^\beta_\alpha, | \ , -1 & q^\beta_\alpha, | \ \to q^\beta_\alpha, | \ , -1 & q^\gamma_\alpha, | \ \to q^\gamma_\alpha, | \ , -1 \\
q'_\alpha, \Lambda \to q^\gamma_\alpha, \Lambda, -1 & q^\beta_\alpha, \# \to q_\beta, \# \, , 0 & q^\gamma_\alpha, \# \to q_\gamma, \# \, , 0
\end{array}
$$

В результате выполнения команд из P_o на ленте машины Тьюринга будут записаны значения переменных $x_1, \ldots, x_k$ в следующей конфигурации программы и состояние будет соответствовать метке следующей конфигурации.

Последний этап работы машины Тьюринга будет заключаться в подготовке выходного слова. Поскольку в программе Π выходной является переменная x_1, то результат на ленте — это первое же слово. Поэтому единственное, что остаётся, — стереть остальные слова:

$$\begin{array}{ll} q_{\alpha_m}, \# \to q_z^1, \#, +1 & q_z^3, | \ \to q_z^2, \Lambda, +1 \\ q_z^1, | \ \to q_z^1, | \ , +1 & q_z^3, \Lambda \to q_z^4, \Lambda, -1 \\ q_z^1, \Lambda \to q_z^2, \Lambda, +1 & q_z^4, | \ \to q_z^4, | \ , -1 \\ q_z^2, | \ \to q_z^2, \Lambda, +1 & q_z^4, \Lambda \to q_z^4, \Lambda, -1 \\ q_z^2, \Lambda \to q_z^3, \Lambda, +1 & \end{array}$$

Таким образом, на ленте останется только запись значения переменной x_1, которое и является значением функции f. $\square$

Теперь займёмся доказательством обратного утверждения: каждая вычислимая по Тьюрингу функция вычислима программно. Здесь перед нами встаёт обратная проблема: машины Тьюринга работают со словами, поэтому, чтобы сравнить машину Тьюринга с программой, нужно разнозначно закодировать слова числами.

Для этого используем такой метод. Будем считать, что все символы алфавита Σ пронумерованы подряд натуральными числами: $\Sigma = \{a_0, a_1, a_2, \ldots, a_{k-1}, a_k\}$. При этом будем считать, что $\alpha_0 = \Lambda$, $\alpha_k = \#$. Символ начала ленты # учитывать не будем, так как ни в какой ячейке кроме нулевой он встречаться не может. Тогда любое слово можно рассматривать как запись какого-то числа в k-ичной системе счисления, если считать, что $a_0, \ldots, a_{k-1}$ — это k-ичные цифры $0, \ldots, k-1$ соответственно. Для удобства полагаем, что эти цифры идут слева направо в порядке возрастания разрядов. Таким образом, кодом слова $a_{i_0} a_{i_1} a_{i_2} \ldots a_{i_n}$ является число, k-ичная запись которого имеет вид $i_n \ldots i_2 i_1 i_0$, а именно $\sum\limits_{j=0}^{n} i_j k^j$. Обозначим такой способ кодирования с помощью ρ: $\rho(w)$ — код слова w.

Для двух слов такие коды совпадут, только если хотя бы одно из них оканчивается символом $a_0 = \Lambda$. Но поскольку мы условились считать, что слова пустых символов не содержат, то такое кодирование слов будет разнозначным.

Также заметим, что этот способ позволяет взаимно однозначно закодировать содержимое ленты α односторонней машины Тьюринга. В самом деле, лента в каждый момент времени содержит последовательность символов такого вида:

$$\#a_{i_1}a_{i_2}a_{i_3}\dots a_{i_n}\Lambda\Lambda\dots$$

Поскольку мы условились, что $\Lambda = a_0$, а # не рассматривается, то код содержимого ленты в k-ичной системе имеет вид $\dots 00i_n\dots i_3i_2i_1$. Ведущие нули в записи числа роли не играют, поэтому получаем число $\sum\limits_{j=1}^{n} i_j k^{j-1}$. Таким образом, $\rho(\alpha) = \sum\limits_{j=1}^{\infty} \alpha(j) \times k^{j-1}$, причём в этой сумме ненулевыми являются только конечно много слагаемых, поэтому она определена.

Определение 162 (Программная вычислимость словарной функции)**.** *Пусть зафиксирован алфавит $\Sigma = \{a_0, a_1, a_2, \dots, a_k\}$, причём $a_0 = \Lambda$ и $a_k =$ #. Скажем, что программа Π вычисляет частичную словарную n-местную функцию f, если для всех наборов слов $w_1, \dots, w_n$ выполнено одно из двух:*

1) *$f(w_1, \dots, w_n)$ определено и равно u, Π останавливается на входе $(\rho(w_1), \dots, \rho(w_n))$, а результатом при этом является $\rho(u)$;*
2) *$f(w_1, \dots, w_n)$ неопределено и Π на входе $(\rho(w_1), \dots, \rho(w_n))$ не останавливается.*

Словарная (частичная) функция $f(w_1, \dots, w_n)$ программно вычислима, если существует вычисляющая её программа с метками.

Теорема 158. *Каждая вычислимая по Тьюрингу словарная функция является программно вычислимой.*

Доказательство. Пусть дана словарная функция f и вычисляющая её машина Тьюринга $\mathfrak{M} = (Q, \Sigma, P, q_0)$. Без ограничения общности будем считать, что эта машины является односторонней. Чтобы промоделировать работу машины Тьюринга $\mathfrak{M}$ с помощью программы Π, мы должны будем хранить её конфигурацию (q, i, α) в переменных программы Π и текущих метках. Для этого выделим

переменную h, в которой будет находится номер обозреваемой ячейки, и переменную r, где будет храниться код ленты α. Текущее состояние машины Тьюринга будет соответствовать текущей метке программы.

Работа программы будет состоять из трёх этапов: начальный — инициализация нужных переменных, второй — непосредственно моделирование $\mathfrak{M}$, и третий — получение результата.

Пусть коды слов передаются во входных переменных $x_1, \ldots, x_n$. Тогда содержимое ленты α_0 в начальный момент выглядит так:

$$\#w_1\Lambda w_2\Lambda \ldots w_n\Lambda\Lambda \ldots$$

Код такого содержимого ленты можно итеративно вычислить по формуле:

$$\rho(w_j\Lambda \ldots \Lambda w_n) = \rho(w_j) + \rho(w_{j+1}\Lambda \ldots \Lambda w_n) \times k^{|w_j|+1}.$$

Длина $|w_j|$ слова w_j совпадает с количеством цифр в k-ичной записи его кода. Это последнее равно $[\log_k \rho(w_j)] + 1$. Таким образом, процедура вычисления начального значения переменной r может быть представлена так:

1: $r \leftarrow x_n$
2: **Для** $i \leftarrow n-1, \ldots, 1$ **выполнять**
3: $\quad \ell \leftarrow [\log_k x_i] + 1$
4: $\quad r \leftarrow x_i + r \times k^{\ell}$
5: **Конец Для**
6: **Вернуть** r

что нетрудно переделать в программу с метками $\alpha\ \Pi_r\ \beta$, так как все используемые здесь действия (сложение, умножение, возведение в степень) программно вычислимы. Значение переменной h, очевидно, должно быть равно 0, поэтому первый этап работы программы Π заключается в выполнении такого фрагмента Π_0:

$$\begin{array}{ll} \alpha & \Pi_r \qquad \beta \\ \beta & h \leftarrow 0;\ q_0 \end{array}$$

Второй этап работы программы Π будет заключаться в моделировании команд машины Тьюринга $\mathfrak{M}$. Напомним, что i-я ячейка ленты α соответствует i-й цифре числа $\rho(\alpha)$ в его k-ичной записи. Поэтому изменение ленты заключается в изменении соответствующей цифры числа, хранимого в переменной r.

Прежде всего, для любого состояния $q \in Q$ машины Тьюринга $\mathfrak{M}$ нужно определить, какой символ a_j обозревает головка в текущей конфигурации. Для этого нужно записать программу с метками $q\ \Pi_q\ q'$, соответствующую следующим действиям:

1: **Если** $h = 0$ **то**
2: $\quad c \leftarrow k$
3: **Иначе**
4: $\quad d \leftarrow k^h$
5: $\quad c \leftarrow [r/d]$
6: $\quad c \leftarrow c \bmod k$
7: **Конец Если**

после чего в переменной c будет находится j. Далее последовательно нужно проверять полученное значение c, чтобы определить какую именно из команд требуется выполнить:

$$
\begin{array}{rll}
q' & u \leftarrow 0; & q'_0 \\
q'_0 & u \leftarrow s(u); & q''_0 \\
q''_0 & v \leftarrow (c < u); & q'''_0 \\
q'''_0 & \textbf{Если } v \textbf{ то} & q_{a_0} \textbf{ Иначе } q'_1 \\
q'_1 & u \leftarrow s(u); & q''_1 \\
q''_1 & v \leftarrow (c < u); & q'''_1 \\
q'''_1 & \textbf{Если } v \textbf{ то} & q_{a_1} \textbf{ Иначе } q'_2 \\
 & \dots & \\
q'_{k-1} & u \leftarrow s(u); & q''_{k-1} \\
q''_{k-1} & v \leftarrow (c < u); & q'''_{k-1} \\
q'''_{k-1} & \textbf{Если } v \textbf{ то} & q_{a_{k-1}} \textbf{ Иначе } q_{a_k}
\end{array}
$$

В результате текущей меткой станет q_{a_i}, если обозревался символ a_i.

Далее определим часть программы $q_{a_i}\ \Pi_{qa_i}\ p$, соответствующую команде $q, a_i \to p, a_j, s$.

Если $i < k$, то в числе, которое хранится в переменной r, нужно заменить h-ю цифру на j и изменить значение самой переменной h:

1: $r_1 \leftarrow r \bmod d$
2: $r_2 \leftarrow [r/d]$
3: $r_2 \leftarrow [r_2/k]$
4: $r \leftarrow r_2 \times d \times k + j \times d + r_1$
5: $h \leftarrow h + s$

Если $i = k$, то содержимое ленты измениться не может, меняем только h:

1: $h \leftarrow h + s$

Если в программе P машины Тьюринга отсутствует команда вида $q, a_i \to \ldots$, то это означает остановку машины. В таком случае наша программа П должна будет выполнить последнее действие: расшифровать содержимое ленты, чтобы получить результат. Если вычисляется функция, значением которой является одно слово, то ничего делать не нужно, так как код ленты будет совпадать с кодом этого слова, следовательно, r будет результатом вычисления. Если же предполагается, что значением функции может быть набор из нескольких выходных слов $v_1, \ldots, v_t$, то их нужно по очереди извлечь из переменной r (задача 292 на стр. 433). □

Из последней теоремы можно извлечь ещё одно существенное утверждение.

Теорема 159. *Если словарная функция $f : \Omega^* \to \Omega^*$ вычислима на машине Тьюринга $\mathfrak{M} = (Q, \Sigma, P, q_0)$, $\Omega \subseteq \Sigma$, то она вычислима на односторонней машине Тьюринга $\mathfrak{N} = (Q', \Sigma', P', q_0')$ без расширения алфавита: $\Sigma' = \Omega \cup \{\Lambda, \#\}$.*

Доказательство. Будем считать, что алфавит Ω содержит символ a.

Рассмотрим программу с метками П, реализующую второй этап моделирования машины Тьюринга $\mathfrak{M}$ (теорема 158 на стр. 423). Входной и выходной переменной будем считать r. Такая программа в качестве входа получает код ленты в какой-либо конфигурации машины $\mathfrak{M}$ и возвращает код ленты в заключительной конфигурации

машины $\mathfrak{M}$ (если $\mathfrak{M}$ останавливается). Пусть машина Тьюринга $\mathfrak{N}_1$ построена по теореме 157 на стр. 419 и вычисляет ту же арифметическую функцию, что и П в унарной записи, где в качестве «палочки» используется буква a. Ввиду использования унарной записи алфавит машины $\mathfrak{N}$ будет таким: $\{\Lambda, \#, a\}$.

Тогда работа машины $\mathfrak{N}$ будет заключаться в следующем: сначала по входу построить унарную запись числа $\rho(\alpha_0)$, затем — запустить $\mathfrak{N}_1$, наконец, если $\mathfrak{N}_1$ остановится, по унарной записи выхода $\rho(\alpha)$ построить сам выход (задача 293 на стр. 433). □

§ 20.6. Универсальная машина Тьюринга

Пусть дана некоторая формализация $\mathcal{A}$ понятия алгоритм: машина Тьюринга, программа с метками, ч. р. ф. или любая другая. Назовём алгоритм $\mathfrak{U}$ универсальным в классе формализаций $\mathcal{A}$, если для всякого алгоритма $\mathfrak{M}$ из $\mathcal{A}$ существует такой вход $c_{\mathfrak{M}}$, что для всех входов x выполнено $\mathfrak{U}(c_{\mathfrak{M}}, x) = \mathfrak{M}(x)$. Иными словами, если первой частью входа алгоритма $\mathfrak{U}$ является $c_{\mathfrak{M}}$, то алгоритм $\mathfrak{U}$ работает как $\mathfrak{M}$, и для каждого $\mathfrak{M}$ такое $c_{\mathfrak{M}}$ найдётся.

Дадим более точное определение для машин Тьюринга.

Определение 163 (Универсальная машина Тьюринга)**.** *Пусть Σ — некий алфавит машин Тьюринга. Скажем, что машина Тьюринга $\mathfrak{U} = (Q^{\mathfrak{U}}, \Sigma, P^{\mathfrak{U}}, q_0^{\mathfrak{U}})$ является Σ-универсальной, если для каждой машины Тьюринга $\mathfrak{M} = (Q, \Sigma, P, q_0)$ существует слово $c_{\mathfrak{M}} \in \Sigma^*$ такое, что для всех слов $w_1, \ldots, w_n \in \Sigma^*$ выполнено $\mathfrak{U}(c_{\mathfrak{M}}, w_1, \ldots, w_n) = \mathfrak{M}(w_1, \ldots, w_n)$.*

Один из фундаментальных фактов заключается в существовании универсальных алгоритмов.

Теорема 160. *Для любого алфавита Σ существует Σ-универсальная машина Тьюринга $\mathfrak{U}$.*

Доказательство. Прежде всего отметим, что в силу теоремы 159 на предыдущей странице при построении универсальной машины $\mathfrak{U}$

допустимо применять расширение алфавита, так как лишние символы потом можно будет убрать.

В качестве Σ-универсальной машины мы построим интерпретатор $\mathfrak{U}$. Первым аргументом машины $\mathfrak{U}$ будет программа P произвольной машины $\mathfrak{M}$, а работа $\mathfrak{U}$ будет заключаться в пошаговом моделировании работы этой программы. Следовательно, для любой машины $\mathfrak{M} = (Q, \Sigma, P, q_0)$ в качестве первого слова $c_{\mathfrak{M}}$ можно будет взять её программу P. В силу теоремы 156 на стр. 414 можно ограничиться рассмотрением в качестве $\mathfrak{M}$ только односторонних машин, поэтому считаем, что $\Lambda, \# \in \Sigma$. Кроме того, всегда можно считать, что $\mathfrak{M}$ имеет стандартную заключительную конфигурацию: в момент остановки головка всегда оказывается в начальной ячейке.

Прежде всего нужно решить вопрос о том, как записывать программы P на ленте, то есть — построение кодирования программ. Напомним, что программа P машины Тьюринга $\mathfrak{N}$ состоит из команд вида «$q, a \to p, b, s$», где $q, p \in Q$, $a, b \in \Sigma$, $s \in \{+1, -1, 0\}$. Всевозможные варианты для a, b, s заранее известны (напомним, что алфавит Σ мы заранее зафиксировали). Этого нельзя сказать о Q, оно может содержать сколько угодно много состояний. Поэтому мы условимся о следующей записи программы P машины $\mathfrak{M}$. Будем считать, что все состояния Q пронумерованы натуральными числами, причём q_0 — начальное состояние. Алфавит универсальной машины $\mathfrak{U}$ будет в себя включать:

1) все символы алфавита Σ;

2) вспомогательные символы для записи программы машины $\mathfrak{M}$: «q», «$|$», «$\to$», «$+1$», «-1», «0», «h», «λ» (мы считаем, что «$+1$» и «-1» — это единые символы, а не два отдельных). Предполагаем, что все эти символы отсутствуют в Σ;

3) двухэтажные символы $\left[\!\!\begin{matrix} x \\ y \end{matrix}\!\!\right]$, составленные из элементов двух предыдущих множеств. Как и в доказательстве теоремы 156 на стр. 414 мы будем отождествлять двухэтажный символ $\left[\!\!\begin{matrix} x \\ \Lambda \end{matrix}\!\!\right]$ с символом x.

Сама универсальная машина $\mathfrak{U}$ будет двусторонней, поэтому # для неё — обычный символ, как любой другой непустой. Каждую команду машины $\mathfrak{N}$, которая имеет вид $q_i, a \to q_j, b, s$ мы будем представлять таким словом: «$q|^{i+1}a \to q|^{j+1}bs$». Таким образом, после q мы записываем в унарной системе номер состояния. Конкатенация всех этих слов и даст нам запись C_P программы P машины $\mathfrak{M}$. Заметим, что проблем с отделением одной команды от другой не возникает: каждый из символов «+1», «−1», «0» однозначно определяет конец очередной команды. Точно так же «→» отделяет левую и правую части команды.

Поскольку в командах может встречаться пустой символ Λ, то возникает проблема, как определить, является ли пустой символ частью команды или пробелом между входными словами? Чтобы такого вопроса не возникало, мы в командах вместо Λ будем записывать λ: например, команда вида «$q_1, a \to q_3, \Lambda, 0$» будет выглядеть так: «$q||a \to q||||\lambda 0$».

Заметим, что при записи программ мы уже использовали символы расширенного алфавита. Чтобы впоследствии эти символы можно было элиминировать, входные слова их содержать не должны. Поэтому используем тот же приём, что в доказательстве теоремы 159 на стр. 426: само слово C_P рассматриваем как число в k-ичной системе счисления, а в качестве входного слова берём его унарный эквивалент c_P, записанный при помощи какого-то фиксированного символа из Σ.

При моделировании работы машины $\mathfrak{M}$ универсальная машины $\mathfrak{U}$ будет иметь такую базовую конфигурацию. Пусть конфигурация машины $\mathfrak{M}$ имела вид (q_j, ℓ, α). Тогда у машины $\mathfrak{U}$ в ℓ-й ячейке ленты вместо символа a_ℓ будет записан двухэтажный символ $\left[\!\!\begin{smallmatrix} a_\ell \\ h \end{smallmatrix}\!\!\right]$, левее # на верхнем этаже будет записано C_P, а в самом начале C_P на нижнем этаже будет записан код текущего состояния $q|^{j+1}$. Там же будет находится и головка машины $\mathfrak{U}$ (рис. 125 на следующей странице).

Мы не будем писать непосредственно текст программы $P^{\mathfrak{U}}$ универсальной машины, вместо этого постараемся максимально подробно его описать. У читателя не должно возникнуть затруднения в реализации $P^{\mathfrak{U}}$ по этому описанию (задача 299 на стр. 434).

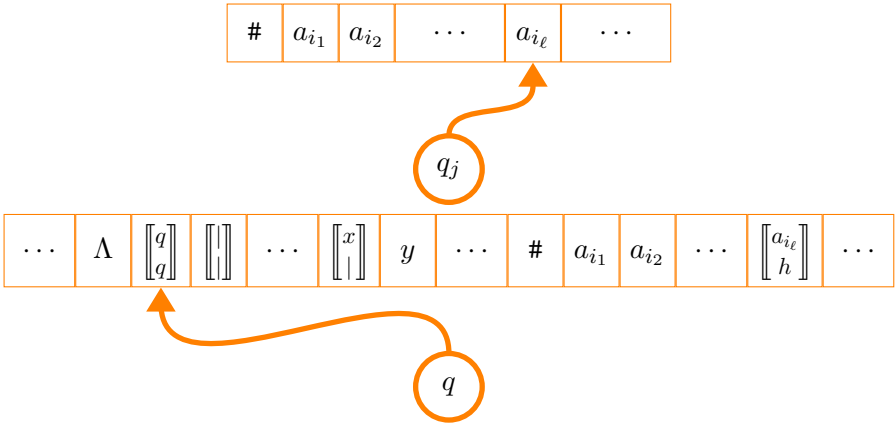


Рис. 125: Конфигурации машин $\mathfrak{M}$ и $\mathfrak{U}$.

П е р в ы й э т а п работы машины $\mathfrak{U}$ будет заключаться в создании нужной конфигурации. Для этого достаточно выполнить такие действия:

1) Преобразуем первый аргумент c_P в C_P, как описано выше.
2) В начале слова C_P на нижнем этаже приписываем слово $q|$.
3) Движемся вперёд до первого пустого символа и заменяем его на $\left[\!\left[\begin{smallmatrix} \# \\ h \end{smallmatrix}\right]\!\right]$.
4) Возвращаемся к началу C_P.

В т о р о й э т а п работы — это непосредственно выполнение команд программы. Выполнение каждой такой команды начинается, когда головка располагается в начале C_P.

5) Движемся вперёд, пока не обнаружим h на нижнем этаже — текущее положение головки моделируемой машины $\mathfrak{M}$.
6) Запоминаем в состоянии символ a, стоящий на верхнем этаже, — обозреваемый в текущий момент.
7) Возвращаемся назад до «|» на нижнем этаже и дописываем символ a, если $a \neq \Lambda$. Если $a = \Lambda$, то вместо a записываем λ.

8) Дописываем на нижнем этаже символ $\to$.

Теперь у нас на нижнем этаже записана левая часть команды, которую нужно выполнить.

9) Начинаем сдвигать слово на нижнем этаже вперёд по одной ячейке, и каждый раз проверяем, не совпало ли оно с верхним.

Возможны два варианта. Первый — в некоторый момент нижнее слово совпало с верхним тогда переходим к следующему пункту. Второй — мы дошли до двухэтажного символа $\left[\!\!\begin{smallmatrix}\#\\ h\end{smallmatrix}\!\!\right]$. Это означает, что в программе машины $\mathfrak{M}$ нет команды для выполнения, она должна остановиться. Тогда переходим к третьему этапу моделирования. Следует заметить, что машина $\mathfrak{M}$ может остановиться только при наличии такого символа, поскольку имеет стандартную заключительную конфигурацию.

10) Движемся вперёд, пропуская на верхнем этаже символы «q» и «$|$».

11) Запоминаем новый символ b и направление сдвига s в состоянии.

12) Движемся вперёд, пока не обнаружим h на нижнем этаже — текущее положение головки моделируемой машины $\mathfrak{M}$.

13) Заменяем на верхнем этаже a на b, если $b \neq \lambda$. Если $b = \lambda$, то заменяем a на Λ.

14) Сдвигаем на нижнем этаже h в соответствии с s: вправо при $s = +1$, влево при $s = -1$ или не меняем положение h при $s = 0$.

15) Возвращаемся назад, до символа $\to$ на нижнем этаже.

16) Движемся вперёд и копируем символы q и $|$ с верхнего этажа на нижний, пока не дойдём до иного символа.

17) Возвращаемся назад, до символа $\to$ на нижнем этаже.

18) Движемся назад на нижнем этаже и стираем его содержимое до «q» включительно.

19) Движемся вперёд, пока не найдём q на нижнем этаже.

20) Начинаем сдвигать слово на нижнем этаже назад, пока не встретим Λ.

21) Переходим к пункту 5), чтобы начать выполнять следующую команду.

Третий этап заключается в удалении с ленты всего лишнего. Так как моделируемая машина $\mathfrak{M}$ имеет стандартную заключительную конфигурацию, то её головка располагается в ячейке с #. У машины $\mathfrak{U}$ тогда в этой ячейке будет стоять символ $\left[\!\!\begin{smallmatrix}\#\\ h\end{smallmatrix}\!\!\right]$, а правее него на нижнем этаже стоят только пустые символы. Так как мы условились, что символы вида $\left[\!\!\begin{smallmatrix}a\\ \Lambda\end{smallmatrix}\!\!\right]$ совпадают с a, то достаточно только стереть символ $\left[\!\!\begin{smallmatrix}\#\\ h\end{smallmatrix}\!\!\right]$ и все символы левее.

Итак, мы построили универсальную машину Тьюринга с расширенным алфавитом. Но поскольку входные и выходные слова в алфавите Σ, то, используя теоремы 156 на стр. 414 и 159 на стр. 426, её можно преобразовать в одностороннюю машину со стандартной заключительной конфигурацией и алфавитом Σ. □

Задачи

285. Построить программы машин Тьюринга, вычисляющих следующие словарные функции в алфавите $\Sigma = \{a, b, c\}$:

(а) циклическая перестановка букв: $f(xw) = wx$, $x \in \Sigma$;

(б) циклическая перестановка слов: $f(w_1, \ldots, w_n) = (w_2, \ldots, w_n, w_1)$;

(в) «переворачивание» последовательности слов: $f(w_1, \ldots, w_n) = (w_n, \ldots, w_1)$;

(г) удвоение каждой буквы: $f(x_1 \ldots x_n) = x_1 x_1 \ldots x_n x_n$;

(д) нахождение образа при гомоморфизме φ: $\varphi(a) = a$, $\varphi(b) = cb$, $\varphi(c) = \varepsilon$;

(е) удаление всех одиночных букв a, стоящих на чётных позициях;

(ж) проверка, содержат ли два слова одно и то же количество букв a (результат равен 1, если ответ «да», 0, если «нет»);

(з) проверка, является ли слово палиндромом (то есть симметрично);

(и) проверка, есть ли в слове две последовательности букв a одной и той же длины;

(к) проверка, образуют ли в слове количества букв a, b и c возрастающую арифметическую прогрессию. ▼

286. Построить машину Тьюринга для перевода записи числа из двоичной системы в унарную, входной алфавит $\{0, 1\}$. ▼

287. Построить машину Тьюринга, выполняющую следующую задачу: по входу, состоящему из одного или нескольких слов $w_1\Lambda\ldots\Lambda w_n$ в алфавите Σ (возможно, что $n = 1$), построить выход, удвоив последнее из слов: $w_1\Lambda\ldots\Lambda w_n\Lambda w_n$. ▼

288. Реализовать машину Тьюринга из примеров 109 на стр. 406 и 110 на стр. 407 без расширения алфавита. ▼

289. Завершить построение машины Тьюринга из теоремы 155 на стр. 410. ▼

290. Доказать, что односторонняя машина Тьюринга $\mathfrak{N}$, построенная в теореме 156 на стр. 414, корректно моделирует исходную машину $\mathfrak{M}$. ▼

291. Другой, по сравнению с конструкцией теоремы 156 на стр. 414, подход к моделированию двухсторонней ленты на односторонней заключается в том, чтобы содержимое неотрицательной половины ленты $\mathfrak{M}$ хранить в ячейках с нечётными номерами, а содержимое левой половины — с чётными. То есть новая лента будет иметь вид $\beta(0) = \#$, $\beta(2x + 1) = \alpha(x)$ и $\beta(2x) = \alpha(-x)$ при $x > 0$. Построить программу односторонней машины $\mathfrak{N}$, реализующую этот подход. ▼

292. Показать, как извлечь из кода ленты $\rho(\alpha)$ выходные слова (теорема 158 на стр. 423). ▼

293. Показать, как на машине Тьюринга построить унарную запись входа и по унарной записи восстановить выход (теорема 159 на стр. 426) без расширения алфавита. ▼

294. Показать, как промоделировать на машине Тьюринга работу программы с метками в двоичной системе (теорема 157 на стр. 419). ▼

295. Построить программы односторонних машин Тьюринга, вычисляющих следующие арифметические функции в унарной системе:

(а) $x + y$;
(б) $\lfloor x - y\rfloor$;
(в) $x \times y$;
(г) сравнение $x < y$;
(д) возведение в степень: n^m;
(е) квадратный корень: $[\sqrt{n}]$;
(ж) нахождение логарифма: $[\log_2 n]$;
(з) деление нацело: $[n/m]$;
(и) остаток от деления: $n \bmod m$;
(к) функция выбора аргумента: id^n_m. ▼

296. Используя машины Тьюринга из предыдущей задачи, построить программы машин Тьюринга, вычисляющих следующие функции:

(а) $f_1(x, y) = \begin{cases} x^2y^3, & \text{если } x < y, \\ [(x + y)/2], & \text{в противном случае;} \end{cases}$

(б) $f_2(x, y) = \begin{cases} [\sqrt{x}], & \text{если } x + 1 \geqslant y, \\ 2(x + y), & \text{в противном случае;} \end{cases}$

(в) $f_3(x, y) = \begin{cases} [\log_2 (1 + [x/y])], & \text{если } x > 2y, \\ xy, & \text{в противном случае;} \end{cases}$

(г) $f_4(x,y) = \begin{cases} [\sqrt{x}], & \text{если } 2x \geqslant y, \\ x \bmod y, & \text{в противном случае.} \end{cases}$

297. Построить машину Тьюринга, сравнивающую два входных слова в алфавите $\{a,b,c\}$ лексикографически. Эта машина Тьюринга должна вычислять словарную функцию:

$$f(x,y) = \begin{cases} a, & \text{если } x < y, \\ b, & \text{если } x = y, \\ c, & \text{если } y < x. \end{cases}$$

298. Показать, как на обычной машине Тьюринга можно организовать выполнение следующих команд:

(а) $q, a \to p, b, \text{return}$ — возврат головки в ту из соседних ячеек, из которой она пришла в текущую;

(б) $q, a \to p, b, \text{zero}$ — возврат головки в нулевую ячейку;

(в) $q, a \to p, b, \text{mirror}$ — сдвиг головку в ячейку $-i$, если она была в ячейке с номером i;

(г) $q, a \to p, b, \text{double}$ — сдвиг головку в ячейку $2i$, если она была в ячейке с номером i;

(д) $q, a \to p, b, \text{next}$ — сдвиг головки в ближайшую справа к текущей ячейку, содержащую a (если она есть, иначе головка остаётся на месте);

(е) $q, a \to p, \text{insert}\, b$ — сдвиг текущей и всех ячеек справа от неё на одну позицию вправо и запись символа b в освободившуюся ячейку, головка оказывается в новой ячейке;

(ж) $q, a \to p, \text{restore}, s$ — замена символа a на тот, который находился в ячейке в начальной конфигурации.

299. Построить универсальную машину Тьюринга, реализовав пункты 1)–21) из теоремы 160 на стр. 427.

Глава 21

Вычислимость и неразрешимые проблемы

Краткое содержание: тезис Тьюринга-Чёрча, алгоритмически разрешимые и неразрешимые проблемы, неразрешимость проблем самоприменимости, остановки, тотальности и односторонности, невычислимость функций «усердного бобра» и оптимального сжатия.

Ключевые слова: тезис Тьюринга-Чёрча, разрешимые и неразрешимые множества, алгоритмическая сводимость проблем.

§ 21.1. Тезис Тьюринга-Чёрча

Мы рассмотрели три математические модели для описания алгоритмов и вычисляемых ими функций, отражающие различные аспекты и представления о работе абстрактного вычислителя. Из теорем 143 на стр. 390, 152 на стр. 395, 157 на стр. 419 и 158 на стр. 423 непосредственно получаем

Теорема 161. *Классы функций программно вычислимых, частично рекурсивных и вычислимых по Тьюрингу совпадают.*

Естественно, возникает вопрос о том, насколько общим является этот результат? Верно ли, что каждый алгоритм может быть задан одним из рассмотренных способов? На эти вопросы теория алгоритмов отвечает следующей гипотезой, которую называют т е з и с о м Т ь ю р и н г а - Ч ё р ч а:

> всякий алгоритм (в интуитивном смысле) может быть формализован в виде соответствующей машины Тьюринга (частично рекурсивного определения), а класс вычислимых (в интуитивном смысле) функций совпадает с классом функций, вычислимых на машинах Тьюринга (с классом частично рекурсивных функций).

Значение этого тезиса заключается в том, что он уточняет общее неформальное определение «всякого алгоритма» и «вычислимой функции» через точные формальные понятия машины Тьюринга, частично рекурсивного определения и соответствующих им классов функций. После этого можно осмысленно ставить вопрос о существовании или не существовании алгоритма, решающего тот или иной класс задач. Теперь этот вопрос следует понимать как вопрос о существовании или не существовании соответствующей машины Тьюринга или (что эквивалентно) программы с метками или частично рекурсивного определения соответствующей функции.

Можно ли доказать этот тезис как теорему? Нет, поскольку в его формулировке речь идёт о неточных, интуитивных, понятиях «алгоритм» и «вычислимая функция», которые не могут быть объектами математических рассуждений. На чём же тогда основана уверенность в справедливости тезиса Тьюринга-Чёрча? В первую очередь, на опыте. Все известные алгоритмы, придуманные за многие века математиками, могут быть заданы с помощью машин Тьюринга. Для всех многочисленных моделей алгоритмов, появившихся в XX–XXI вв. (некоторые из них мы упоминали в начале главы 18), была доказана их равносильность машинам Тьюринга. В качестве доводов в пользу

тезиса Тьюринга-Чёрча можно также рассматривать замкнутость класса машин Тьюринга и ч. р. ф. относительно многочисленных естественных операций над алгоритмами и функциями.

Отметим также, что тезис Тьюринга-Чёрча обращён и в будущее: он предполагает, что какие бы новые формальные определения алгоритмов ни были предложены (а таковыми, например, являются новые языки программирования), все они не выйдут из класса алгоритмов, задаваемых машинами Тьюринга.

§ 21.2. Алгоритмически неразрешимые проблемы

Доказать существование невычислимых функций можно, даже не приводя ни одного конкретного примера.

Теорема 162. *Существуют невычислимые по Тьюрингу арифметические функции.*

Доказательство. Мы уже видели, что любая вычислимая функция может быть вычислена на машине Тьюринга, имеющей алфавит $\Sigma = \{\Lambda, \#, |\}$. Каждая такая машина может быть однозначно описана словом или натуральным числом, как это мы делали в теореме 160 на стр. 427. Мы уже знаем, что множество всех слов в конечном алфавите счётно. Следовательно, существует счётно много машин Тьюринга с алфавитом Σ и счётно много вычислимых функций.

С другой стороны общее количество арифметических функций несчётно, так как несчётно множество уже характеристических функций (следствие 5 на стр. 34). Следовательно, существуют невычислимые арифметические функции. □

Тем не менее, конечно, хотелось бы иметь конкретные примеры, чтобы получить представление о природе невычислимых функций. В этом и следующем параграфе мы такие примеры и рассмотрим.

Прежде всего нужно отметить, что большую часть изучаемых задач составляют задачи типа: для свойства S построить алгоритм $\mathfrak{M}$, который по любому объекту x определяет, обладает x свойством

S или нет. Например, определить по натуральному числу x, является ли оно простым? Естественно, ответ обычно возвращается в формате 1–0: 1 — да, обладает, 0 — нет, не обладает. Для нашего примера это будет функция p из задачи 276 на стр. 397. Как мы уже знаем, каждая функция f, область значений которых состоит из нуля и единицы, является характеристической функций некого множества, в нашем случае — множества простых чисел. Поэтому обычно алгоритмическую проблему отождествляют с множеством, а задачей является проверка вхождения объекта в это множество.

Определение 164 (Алгоритмическая проблема)**.** *Алгоритмической проблемой называется произвольное множество A (натуральных чисел, слов и т. д.) или, что эквивалентно, характеристическая функция этого множества.*

Решением алгоритмической проблемы называется алгоритм $\mathfrak{M}$ (машина Тьюринга, программа с метками, ч. р. ф. и т. д.), вычисляющий характеристическую функцию этого множества:

$$\mathfrak{M}(x) = \begin{cases} 1, \text{если } x \in A, \\ 0, \text{если } x \notin A. \end{cases}$$

Проблема, для которой существует решение, называется (алгоритмически) разрешимой, в противном случае — (алгоритмически) неразрешимой.

Например, проблема палиндромов в алфавите Σ — это множество всех слов-палиндромов в алфавите Σ. Решением этой проблемы является машина Тьюринга, построенная в задаче 285 на стр. 432. Значит, эта проблема алгоритмически разрешима. Разрешимой является и проблема простоты натуральных чисел: упомянутая выше характеристическая функция p множества простых чисел общерекурсивна.

Из определения легко получить полезное следствие.

Следствие 163. *Если проблема A алгоритмически неразрешима, то её дополнение $\bar{A}$ тоже неразрешимо.*

Доказательство. Для характеристических функций множества A и его дополнения $\bar{A}$ выполнено соотношение $I_A = 1 - I_{\bar{A}}$. Следовательно, если функция $I_{\bar{A}}$ вычислима, то и функция I_A вычислима. □

Чтобы привести примеры неразрешимых проблем, начнём с классической проблемы, которая служит отправной точкой для доказательства неразрешимости многих других.

Прежде всего зафиксируем некоторый способ π представления машин Тьюринга с алфавитом Σ словами этого же алфавита. Например, можно взять слова c_P, построенные в теореме 160 на стр. 427, то есть считать, что $\pi(\mathfrak{M}) = c_P$, где P — программа машины $\mathfrak{M}$.

Определение 165 (Проблема самоприменимости)**.** *Машина Тьюринга $\mathfrak{M} = (Q, \Sigma, P, q_0)$ называется самоприменимой, если она останавливается на входном слове $\pi(\mathfrak{M})$.*

Проблемой самоприменимости в этом случае называется следующая словарная функция SELF*:*

$$\mathrm{SELF}(\pi(\mathfrak{M})) = \begin{cases} 1, & \text{если } \mathfrak{M} \text{ самоприменима,} \\ 0, & \text{если } \mathfrak{M} \text{ несамоприменима.} \end{cases}$$

Следовательно, SELF — это характеристическая функция множества самоприменимых машин (точнее — их кодов). Нетрудно понять, что вместо 1 и 0 можно взять любые другие представления ответов, например, символы a и b или произвольное непустое слово и ε. Но мы для единообразия всегда будем ориентироваться на представление ответа именно в форме 1–0.

Теорема 164. *Проблема самоприменимости неразрешима.*

Доказательство. Будем рассуждать от противного. Предположим, что существует машина Тьюринга $\mathfrak{M}_{\mathrm{SELF}}$, вычисляющая функцию SELF. Мы можем считать, что $\mathfrak{M}_{\mathrm{SELF}}$ имеет стандартную заключительную конфигурацию. В нашем случае это будет означать, что при остановке машины $\mathfrak{M}_{\mathrm{SELF}}$ головка находится в той же ячейке, где записан единственный символ результата — 0 или 1.

Построим новую машину $\mathfrak{M}'_{\mathrm{SELF}}$ из машины $\mathfrak{M}_{\mathrm{SELF}}$ следующим способом: если в программе машины $\mathfrak{M}_{\mathrm{SELF}}$ не было команды вида

$q, 1 \to \ldots$ (то есть машина $\mathfrak{M}_{\text{SELF}}$ должна была в этом случае остановиться), то в программу $\mathfrak{M}'_{\text{SELF}}$ добавляем команду $q, 1 \to q, 1, 0$. Нетрудно видеть, что если такая команда сработает хоть один раз, то дальше она будет выполняться до бесконечности, машина зациклится.

Так как машина $\mathfrak{M}_{\text{SELF}}$ вычисляла SELF, то новая машина $\mathfrak{M}'_{\text{SELF}}$ вычисляет следующую функцию:

$$\text{SELF}'(\pi(\mathfrak{N})) = \begin{cases} \infty, & \text{если } \mathfrak{N} \text{ самоприменима,} \\ 0, & \text{если } \mathfrak{N} \text{ несамоприменима.} \end{cases} \tag{28}$$

Выясним, будет ли самоприменимой машина $\mathfrak{M}'_{\text{SELF}}$.

Допустим, что $\mathfrak{M}'_{\text{SELF}}$ самоприменима. Тогда из (28) сразу получаем, что $\text{SELF}'(\pi(\mathfrak{M}'_{\text{SELF}})) = \infty$, это означает, что $\mathfrak{M}'_{\text{SELF}}$ при работе на входе $\pi(\mathfrak{M}'_{\text{SELF}})$ зацикливается, а это означает, что $\mathfrak{M}'_{\text{SELF}}$ несамоприменима. Получили противоречие.

Допустим теперь, что $\mathfrak{M}'_{\text{SELF}}$ несамоприменима. Тогда из (28) следует $\text{SELF}'(\pi(\mathfrak{M}'_{\text{SELF}})) = 0$, что означает, что $\mathfrak{M}'_{\text{SELF}}$ при работе на входе $\pi(\mathfrak{M}'_{\text{SELF}})$ выдаёт результат 0, в частности, она останавливается. Но это означает её самоприменимость. Снова получили противоречие.

Итак, оба варианта ведут к противоречию, значит, неверным было исходное предположение о вычислимости функции SELF. □

Заметим, что на самом деле мы доказали отсутствие машины Тьюринга для вычисления функции SELF. Но тезис Тьюринга-Чёрча и эквивалентность машин Тьюринга программам и ч. р. ф. позволяют сделать вывод об алгоритмической неразрешимости рассматриваемой проблемы независимо от используемой модели алгоритма.

Проблема самоприменимости выглядит очень специфической и не очень интересной с практической, «программистской», точки зрения. Но оказывается, что её можно использовать для доказательства алгоритмической неразрешимости многих других алгоритмических проблем, более тесно связанных с практикой программирования. Метод, который позволяет это делать, называется алгоритмической сводимостью.

Определение 166 (Алгоритмическая сводимость)**.** *Множество (или проблема) A (алгоритмически) сводится к множеству (проблеме) B, если существует вычислимая всюду определённая (то есть общерекурсивная) функция f такая, что для всех x*

$$x \in A \iff f(x) \in B.$$

В этом случае будем писать $A \leqslant_m B$ посредством f.

Содержательно, «A сводится к B посредством f» означает, что для выяснения, входит ли x в A, можно эффективно преобразовать x в такие входные данные $y = f(x)$ проблемы B, что при $y \in B$ имеем $x \in A$, а если $y \notin B$, то и $x \notin A$. Таким образом, умение решать проблему B гарантирует возможность решения проблемы A.

Разрешимые множества сводятся к любым нетривиальным множествам.

Лемма 165. *Если A разрешимо, а $B \subseteq \omega$ не совпадает с $\varnothing$ и ω, то $A \leqslant_m B$.*

Доказательство. Согласно условию леммы найдутся такие b и d, что $b \in B$, а $d \notin B$. Построим о. р. ф. $f(x) = \mathrm{if}(I_A(x), b, d)$, где функция if определена в следствии 146 на стр. 392, а I_A — характеристическая функция множества A. Тогда при $x \in A$ имеем $f(x) = b \in B$, а при $x \notin A$ получаем $f(x) = d \notin B$. Таким образом, $A \leqslant_m B$ посредством функции f. □

Доказательство неразрешимости многих проблем основано на следующем утверждении.

Лемма 166. *Если A сводится к B и проблема A неразрешима, то и проблема B неразрешима.*

Доказательство. Пусть $A \leqslant_m B$ посредством о. р. ф. f. Тогда из определения сводимости следует, что для всех x имеет место равенство $I_A(x) = I_B(f(x))$. Поэтому, если бы B была разрешима, то её характеристическая функция I_B была бы общерекурсивна и I_A также была бы общерекурсивна. Но это противоречит неразрешимости проблемы A. □

Используем метод сводимости, чтобы доказать неразрешимость следующей, более насущной проблемы.

Определение 167 (Проблема остановки)**.** *Проблемой остановки называется двухместная функция* HALT*:*

$$\mathrm{HALT}(\pi(\mathfrak{M}), x) = \begin{cases} 1, & \text{если } \mathfrak{M} \text{ останавливается на входе } x, \\ 0, & \text{если } \mathfrak{M} \text{ не останавливается на входе } x. \end{cases}$$

Неформально говоря, проблема остановки заключается в том, чтобы по программе $\mathfrak{M}$ и её входу определить, остановится ли программа на этом входе.

Теорема 167. *Проблема остановки неразрешима.*

Доказательство. Согласно предыдущей лемме, достаточно свести неразрешимую проблему SELF к HALT. Для этого построим функцию $f(x) = (x, x)$, то есть функцию, удваивающую входное слово (задача 287 на стр. 433). Тогда получаем

$$\pi(\mathfrak{N}) \in \mathrm{SELF} \iff \mathfrak{N}(\pi(\mathfrak{N})) < \infty \iff (\pi(\mathfrak{N}), \pi(\mathfrak{N})) \in \mathrm{HALT},$$

что и требуется от сводящей функции. □

Может возникнуть такое предположение: да, для произвольного, заранее неизвестного, алгоритма $\mathfrak{N}$ проблему остановки решить нельзя. Но, может быть, для каждого конкретного алгоритма $\mathfrak{N}$ эта задача разрешима? Например, если алгоритм $\mathfrak{N}$ вычисляет о. р. ф., то он останавливается на любом входе, поэтому ответ на вопрос об остановке $\mathfrak{N}$ тривиален — всегда «да».

Оказывается, что и для заранее зафиксированной машины $\mathfrak{N}$ проблема остановки $\mathrm{HALT}(\pi(\mathfrak{N}), x)$ может быть неразрешимой.

Теорема 168. *Существует машина Тьюринга $\mathfrak{N}$, для которой проблема остановки $\mathrm{HALT}_{\mathfrak{N}}(x) = \mathrm{HALT}(\pi(\mathfrak{N}), x)$ является алгоритмически неразрешимой.*

Доказательство. Чтобы построить такую машину $\mathfrak{N}$, вспомним о существовании универсальной машины Тьюринга $\mathfrak{U}$ (теорема 160 на стр. 427): $\mathfrak{U}(\pi(\mathfrak{M}), x) = \mathfrak{M}(x)$ для любой машины $\mathfrak{M}$ и любого

входа x. Тогда получаем $\mathfrak{U}(\pi(\mathfrak{M}), \pi(\mathfrak{M})) = \mathfrak{M}(\pi(\mathfrak{M}))$. Пусть машина $\mathfrak{N}$ работает так: сначала удваивает входное слово (задача 287 на стр. 433), а затем запускает универсальную машину $\mathfrak{U}$. Тогда

$$\mathfrak{N}(\pi(\mathfrak{M})) = \mathfrak{U}(\pi(\mathfrak{M}), \pi(\mathfrak{M})) = \mathfrak{M}(\pi(\mathfrak{M})).$$

Следовательно, $\mathfrak{N}$ останавливается на входе $\pi(\mathfrak{M})$ тогда и только тогда, когда $\mathfrak{M}$ останавливается на входе $\pi(\mathfrak{M})$, то есть $\mathfrak{M}$ самоприменима. Поэтому тождественная функция id_1^1 сводит проблему самоприменимости к проблеме остановки для машины $\mathfrak{N}$. □

Одним из способов построения сводящей функции является какая-то небольшая модификация кода программы. Рассмотрим такой пример. С точки зрения практики программирования ещё более важной, чем проблема остановки, является проблема тотальности: верно ли, что программа останавливается на любых входных данных?

Определение 168 (Проблема тотальности)**.** *Проблемой тотальности называется одноместная функция* TOTAL*:*

$$\mathrm{TOTAL}(\pi(\mathfrak{M})) = \begin{cases} 1, & \text{если } \mathfrak{M} \text{ останавливается на всех входах,} \\ 0, & \text{иначе.} \end{cases}$$

Таким образом, множество TOTAL состоит из кодов машин Тьюринга, которые никогда не зацикливаются.

Теорема 169. *Проблема* TOTAL *неразрешима.*

Доказательство. Сведём проблему остановки HALT к проблеме TOTAL.

Для этого построим двухместную функцию f, которая будет вычисляться следующим способом. Пусть аргументами функции f являются два слова $\pi(\mathfrak{M}) = c_P$ и $x = a_0 \dots a_k$. Сначала преобразуем $\pi(\mathfrak{M}) = c_P$ в C_P (см. теорему 160 на стр. 427). В записи программы C_P увеличим номера всех состояний на $k+6$, в результате чего бывшее начальное состояние q_0 станет q_{k+6}. Следующим шагом добавим к слову C_P коды таких команд: «$q_i, a_i \to q_{i+1}, a_i, +1$» для $i = 0, \dots, k$, то есть

припишем к C_P слова вида «$q|^{i+1}a_i \to q|^{i+2}a_i + 1$» для $i = 0, \ldots, k$. После этого к C_P припишем коды следующих команд:

$$\begin{array}{ll} q_{k+1}, \Lambda \to q_{k+2}, \Lambda, +1 & q_{k+4}, a \to q_{k+3}, a\,, -1 \\ q_{k+2}, \Lambda \to q_{k+3}, \Lambda, -1 & q_{k+4}, \Lambda \to q_{k+5}, \Lambda, +1 \\ q_{k+3}, a \to q_{k+3}, a\,, -1 & q_{k+5}, \Lambda \to q_{k+6}, \Lambda, +1 \\ q_{k+3}, \Lambda \to q_{k+4}, \Lambda, -1 & \end{array}$$

для всех непустых $a \in \Sigma$, то есть слова «$q|^{k+2}\lambda \to q|^{k+3}\lambda + 1$»,…, «$q|^{k+6}\lambda \to q|^{k+7}\lambda + 1$». Полученное слово C'_P снова преобразуем в унарный формат c'_P, это и будет значением функции f.

Покажем, что построенная функция f сводит HALT к TOTAL. Прежде всего отметим, что машина $\mathfrak{M}'$, код которой мы построили с помощью функции f: $\pi(\mathfrak{M}') = f(\pi(\mathfrak{M}), x)$, работает следующим образом. Сначала проверяется, не совпадает ли вход с x. Если нет, то машина сразу останавливается (в программе нет соответствующих команд). В противном случае головка возвращается в начальную ячейку, после чего начинает работать машина $\mathfrak{M}$, код программы которой является первым аргументом. Иначе говоря,

$$\mathfrak{M}'(y) = \begin{cases} y, & \text{если } y \neq x, \\ \mathfrak{M}(x), & \text{если } y = x. \end{cases}$$

Пусть $(\pi(\mathfrak{M}), x) \in$ HALT. Тогда $\mathfrak{M}(x)$ определено. Следовательно, результат $\mathfrak{M}'(y)$ определён для всех входов y, иначе говоря, $f(\pi(\mathfrak{M}), x) \in$ TOTAL. Если $(\pi(\mathfrak{M}), x) \notin$ HALT, то $\mathfrak{M}(x)$ неопределено. Поэтому $\mathfrak{M}'(x)$ неопределено и $f(\pi(\mathfrak{M}), x) \notin$ TOTAL. □

При определении односторонних машин мы упоминали, что для произвольной машины $\mathfrak{M}$ сказать, заходит ли головка в область отрицательных ячеек, нельзя, поэтому и приходилось вводить специальное определение. Сейчас мы докажем неразрешимость этой проблемы.

Теорема 170. *Пусть* SIDE 1 *— это множество пар* $(\pi(\mathfrak{M}), x)$ *таких, что машина Тьюринга* $\mathfrak{M}$ *при работе на входе* x *никогда не заходит в область отрицательных ячеек.*

Тогда проблема SIDE 1 *алгоритмически неразрешима.*

Доказательство. Согласно следствию 163 на стр. 438, дополнение $\overline{\text{HALT}}$ проблемы остановки неразрешимо. Сведём $\overline{\text{HALT}}$ к проблеме SIDE 1.

Рассмотрим функцию f, которая вычисляется следующим образом: получив на вход пару $(\pi(\mathfrak{M}), x)$, она преобразует $\pi(\mathfrak{M}) = c_P$ в C_P (см. теорему 160 на стр. 427). Затем программа C_P преобразуется в программу C'_P односторонней машины методом из задачи 291 на стр. 433. После этого находится k — наибольший из номеров состояний в C'_P. Далее к C'_P приписываются команды вида «$q|^n a \to q|^{k+2} a - 1$», если команд вида «$q|^n a \to \ldots$» в C'_P не было, а также команды вида «$q|^{k+2} a \to q|^{k+2} a - 1$» для всех $a \in \Sigma$. Полученное слово C''_P снова преобразуется в унарный формат c''_P и пара $(c''_P, \#x)$ возвращается в качестве результата функции f.

Покажем, что такая функция f и является сводящей $\overline{\text{HALT}}$ к проблеме SIDE 1. Если $(\pi(\mathfrak{M}), x) \in \overline{\text{HALT}}$, то машина $\mathfrak{M}$ на входе x не останавливается. Тогда односторонняя машина $\mathfrak{M}'$ с программой P' на входе $\#x$ тоже не останавливается, значит, новые команды в P'' никогда не выполняются и машина $\mathfrak{M}''$ никогда не выйдет в область отрицательных ячеек. Следовательно, $(c''_P, x) \in \text{SIDE 1}$.

Если же $(\pi(\mathfrak{M}), x) \notin \overline{\text{HALT}}$, то машина $\mathfrak{M}$ на входе x остановится. Тогда остановится и односторонняя машина $\mathfrak{M}'$ на входе $\#x$. Это означает, что вычисление достигнет конфигурации (q_j, i, α) такой, что в программе P' не будет команды вида $q_j, \alpha(i) \to \ldots$. Но тогда в машине $\mathfrak{M}''$ будет выполнена команда $q_j, \alpha(i) \to q_{k+1}, \alpha(i), -1$ и дальше продолжатся выполняться команды типа $q_{k+1}, a \to q_{k+1}, a, -1$. В результате головка машины рано или поздно окажется в области отрицательных ячеек, следовательно, $(c''_P, x) \notin \text{SIDE 1}$. □

§ 21.3. Невычислимые функции

Теперь приведём два примера невычислимых функций, которые не являются характеристическими.

Пусть алфавит машины Тьюринга состоит из двух символов $\{\Lambda, |\}$. Тогда нетрудно заметить, что количество всех машин Тьюринга,

имеющих не больше n состояний, конечно. В самом деле, каждая программа может рассматриваться как частичная функция из множества $Q \times \Sigma$ в $Q \times \Sigma \times S$, где $S = \{+1, -1, 0\}$. Так как оба множества конечны, то и количество частичных функций конечно (задача 300 на стр. 450).

Определение 169 (Функция «усердного бобра»). *Если запустить произвольную машину Тьюринга $\mathfrak{M}$ на пустой ленте, то она либо остановится, либо зациклится. В первом случае* потенциалом машины $\mathfrak{M}$ *назовём количество палочек на ленте в момент остановки, во втором — считаем, что потенциал равен нулю.*

Функция «усердного бобра» bb *определяется так:* $\mathrm{bb}(n)$ *— это самый высокий потенциал среди машин, у которых не больше n состояний.*

Таким образом, $\mathrm{bb}(n)$ — это наибольшее количество палочек, которое можно получить на машине Тьюринга «из ничего» с использованием не более чем n состояний. Непосредственно из определения следует, что функция bb монотонно не убывает: при увеличении n множество рассматриваемых машин только увеличивается, поэтому максимум из их потенциалов не уменьшается.

Мы докажем невычислимость функции bb. На самом деле мы покажем большее: функции bb растёт настолько быстро, что обгоняет в своём росте любую вычислимую функцию.

Теорема 171. *Функция* bb *не мажорируется никакой вычислимой функцией.*

Доказательство. Будем рассуждать от противного: пусть есть вычислимая функция f, мажорирующая bb: $\mathrm{bb}(n) \leqslant f(n)$ для всех натуральных n. Предположим, что функция f вычисляется машиной Тьюринга $\mathfrak{M}_f$ в унарной записи. Нам также понадобятся машины $\mathfrak{M}_1$, которая записывает на пустую ленту число 1 в унарной записи, и $\mathfrak{M}_2$, которая удваивает число в унарной записи. Считаем, что все три машины имеют стандартную заключительную конфигурацию. Пусть также a, b и c — это количества состояний машин $\mathfrak{M}_1$, $\mathfrak{M}_2$ и $\mathfrak{M}_f$ соответственно.

Рассмотрим машину $\mathfrak{N}_k$ устроенную следующим образом: сначала запускается $\mathfrak{M}_1$, затем — k раз запускается $\mathfrak{M}_2$, наконец — $\mathfrak{M}_f$. Тогда общее число состояний машины $\mathfrak{N}_k$ равно $a + kb + c$.

Заметим, что перед запуском машины $\mathfrak{M}_f$ на ленте будет находиться число 2^k в унарной записи, поэтому потенциал машины $\mathfrak{N}_k$ равен $f(2^k) + 1$ (напомним, количество палочек в унарной записи на единицу больше самого числа). Поскольку машина $\mathfrak{N}_k$ имеет $a + kb + c$ состояний и потенциал $f(2^k) + 1$, то можно сделать такой вывод: $\mathrm{bb}(a + kb + c) \geqslant f(2^k) + 1$.

Поскольку при росте k показательная функция 2^k растёт быстрее линейной $a+kb+c$, то существует такое k_0, при котором $2^{k_0} \geqslant a+k_0b+c$. Тогда получаем

$$\mathrm{bb}(2^{k_0}) \geqslant \mathrm{bb}(a + k_0b + c) \geqslant f(2^{k_0}) + 1 \geqslant \mathrm{bb}(2^{k_0}) + 1 > \mathrm{bb}(2^{k_0}).$$

Пришли к противоречию. Значит, наше предположение о существовании вычислимой функции f, мажорирующей bb, неверно. □

Рассмотрим другой пример невычислимой функции. Зафиксируем некоторый алфавит Σ и будем рассматривать работу машин Тьюринга с алфавитом Σ на пустой ленте. Если машина Тьюринга $\mathfrak{M}$ на пустой ленте останавливается и записывает выход $w \in \Sigma^*$, то будем говорить, что $\mathfrak{M}$ порождает w.

Определение 170 (Функция оптимального сжатия)**.** *Функция оптимального сжатия* $\mathrm{arc}(w)$ *равна натуральному числу* n*, если* n *— это наименьшее число состояний машины Тьюринга, порождающей слово (или набор слов)* w*.*

Таким образом, вычисление функции оптимального сжатия фактически заключается в нахождении наименьшего числа состояний машины Тьюринга, достаточного для порождения слова.

Эту задачу можно проинтерпретировать так. Программа машины Тьюринга — это сжатое представление слова w, а выполнив его (можно сказать, «разархивировав»), мы получим само слово w. Тогда функция $\mathrm{arc}(w)$ определяет, до какого размера можно сжать w, поскольку при фиксированном алфавите размер программы машины Тьюринга определяется исключительно количеством состояний.

Теорема 172. *Функция оптимального сжатия* arc *невычислима.*

Доказательство. Снова применим метод рассуждений от противного. Допустим, что функция arc вычислима и $\mathfrak{M}_{\text{arc}}$ — односторонняя машина Тьюринга, которая её вычисляет.

Рассмотрим новую машину Тьюринга $\mathfrak{M}$, которая принимает на вход число n, записанное каким-либо образом, а дальше реализует следующий алгоритм действий:

1: $i \leftarrow 1$
2: **Пока** $i \geqslant 0$ **выполнять**
3: **Для всех** $w \in \Sigma^i$ **выполнять**
4: **Если** $\text{arc}(w) > n$ **то**
5: **Вернуть** w
6: **Конец Если**
7: **Конец Для**
8: $i \leftarrow i + 1$
9: **Конец Пока**

Содержательно, здесь перебираются все слова алфавита Σ: сначала длины 1, затем 2 и т. д. Для каждого слова проверяется условие $\text{arc}(w) > n$, как только будет найдено какое-то слово, удовлетворяющее этому условию, алгоритм остановится, а найденное слово будет его результатом. Таким образом, $\text{arc}(\mathfrak{M}(n)) > n$.

Реализовать эту машину можно так. Пусть $\Sigma = \{a_1, \ldots, a_k\}$. Записываем после входа $|^{n+1}$ слово $w = a_1$ длины $\ell = 1$. Далее выполняем такие действия: создаём копию w (задача 287 на стр. 433), ставим перед этой копией #, запускаем одностороннюю машину $\mathfrak{M}_{\text{arc}}$. В результате после # находится значение $|^{m+1}$ функции arc на слове w. Теперь нужно сравнить $|^{m+1}$ с входом $|^{n+1}$. Если $m > n$, то стираем всё кроме w и останавливаемся. Иначе строим следующее лексикографически слово w длины ℓ. Если w уже является самым больши́м, то строим слово $a_1^{\ell+1}$ на единицу большей длины.

Прежде всего докажем, что $\mathfrak{M}$ останавливается для любого n. В самом деле, существует конечно много (пусть, например, N) машин Тьюринга с не более чем n состояниями и алфавитом Σ. Каждая из них пишет на пустой ленте не более одного слова (некоторые могут

не останавливаться). Следовательно, не больше чем для N слов выполняется условие $\mathrm{arc}(w) \leqslant n$. Поскольку всех слов бесконечно много, то найдутся и те, для которых это условие $\mathrm{arc}(w) \leqslant n$ не выполнено. Как только это произойдёт (а алгоритм $\mathfrak{M}$ перебирает все слова подряд), то цикл завершится.

Как и в предыдущей теореме, пусть $\mathfrak{M}_1$ — это машина, которая имеет a состояний и пишет на пустой ленте единицу, а $\mathfrak{M}_2$ имеет b состояний и удваивает входное число. Предположим, что построенная нами машина $\mathfrak{M}$ имеет d состояний. Также можно считать, что все три машины имеют стандартную заключительную конфигурацию.

Построим машину $\mathfrak{N}_k$ по аналогии с предыдущей теоремой: сначала машина $\mathfrak{N}_k$ запускает $\mathfrak{M}_1$, потом k раз запускает $\mathfrak{M}_2$ и, наконец, $\mathfrak{M}$. Тогда машина $\mathfrak{N}_k$ имеет $a+kb+d$ состояний. Перед запуском $\mathfrak{M}$ на ленте будет записано число 2^k. Выберем k_0 так, чтобы выполнялось неравенство $2^{k_0} > a + k_0 b + d$.

Рассмотрим работу машины $\mathfrak{N}_{k_0}$ на пустой ленте. Перед запуском $\mathfrak{M}$ на ленте записано число 2^{k_0}, поэтому машина $\mathfrak{M}$ должна записать какое-то слово w, для которого выполнено $\mathrm{arc}(w) > 2^{k_0}$. Такое неравенство означает, что никакая машина, которая имеет меньше или равно чем 2^{k_0} состояний слово w породить не может. Но мы только что продемонстрировали, что сама машина $\mathfrak{N}_{k_0}$ имеющая $a+k_0 b+d$ состояний (что меньше 2^{k_0}) это слово порождает. Получили противоречие, значит, наше исходное предположение о вычислимости функции arc неверно. □

* * *

Какой же вывод можно сделать из того, что некоторая алгоритмическая проблема оказалась неразрешимой? Для программистов из такого утверждения извлекаются «две новости: плохая и хорошая». «Плохая» новость заключается в том, что невозможно построить алгоритм (программу) для автоматического решения такой проблемы. Например, из теоремы 167 на стр. 442 следует, что невозможно автоматически проверить, остановится написанная программа или

нет, а из теоремы 168 на стр. 442 — что эта цель может оказаться недостижимой даже для одной конкретной программы. Таким образом, невозможно автоматически проверить корректность построенной программы.

Но неразрешимость проблемы в общем случае не означает, что она не может быть решена для некоторых отдельных входных данных. Например, в предыдущих разделах мы построили достаточно много программ и доказали их корректность. Другой подход заключается в том, чтобы как можно точнее приблизиться к решению такой задачи. Например, общая задача оптимального сжатия неразрешима, но с помощью метода Хаффмана можно построить оптимальный гомоморфизм.

Поэтому «хорошая новость» для программистов и математиков состоит в том, что их труд при решении алгоритмических проблем в каждом отдельном случае является творческим — никакой программой их не заменить. Выявление каждой новой содержательно интересной неразрешимой проблемы только расширяет область их творчества, заставляет искать всё более и более широкие алгоритмы, которые позволяют решать всё более обширные подклассы индивидуальных задач, относящихся к этой проблеме.

Задачи

300. Найти точное количество машин Тьюринга вида $\mathfrak{M} = (Q, \Sigma, P, q_0)$ с заранее зафиксированными множеством состояний Q и алфавитом ленты Σ. ▼

301. Реализовать машину $\mathfrak{M}$ из доказательства теоремы 172 на стр. 448 в случае алфавита $\Sigma = \{a, b\}$.

302. Реализовать машину, вычисляющую сводящую функцию f из доказательства теоремы 169 на стр. 443 в случае алфавита $\Sigma = \{a, b\}$.

303. Найти значение функции «усердного бобра» $\mathrm{bb}(n)$ для $n = 1, 2$. ▼

304. Доказать, что отношение алгоритмической сводимости $\leqslant_m$ является рефлексивным и транзитивным. ▼

305. Доказать алгоритмическую неразрешимость проблему полноты тестовых данных. Проблема состоит из троек $(\pi(\mathfrak{M}), x, q)$ таких, что в вычислении машины $\mathfrak{M}$ на входе x встречается состояние q. ▼

306. Доказать алгоритмическую неразрешимость проблемы нуля ZERO. Проблема ZERO состоит из кодов $\pi(\mathfrak{M})$ таких, что $\mathfrak{M}(x) = \varepsilon$ для всех x. ▼

307. Доказать алгоритмическую неразрешимость проблемы эквивалентности EQU. Проблема EQU состоит из пар $(\pi(\mathfrak{M}), \pi(\mathfrak{N}))$ таких, что машины $\mathfrak{M}$ и $\mathfrak{N}$ эквивалентны. ▼

308. Доказать, что

(а) пересечение двух разрешимых множеств является разрешимым множеством;

(б) объединение двух разрешимых множеств является разрешимым множеством. ▼

309. Доказать, что для двух разрешимых множеств A и B натуральных чисел их «сумма» $A + B = \{x + y : x \in A, y \in B\}$ и «произведение» (не декартово!) $A \cdot B = \{x \cdot y : x \in A, y \in B\}$ также являются разрешимыми множествами. ▼

310. Доказать, что для двух разрешимых языков L и K в алфавите Σ их конкатенация LK и итерация L^* тоже будут разрешимыми языками. ▼

311. Пусть A — разрешимое множество, а $g(x)$ и $h(x)$ являются о. р. ф. Доказать, что функция

$$F(x) = \begin{cases} g(x), & \text{если } x \in A, \\ h(x), & \text{в противном случае} \end{cases}$$

также является общерекурсивной. ▼

312. Доказать, что проблема ограниченной остановки разрешима. Проблема состоит из троек вида $(\pi(\mathfrak{M}), x, t)$ таких, что вычисление машины Тьюринга $\mathfrak{M}$ на входе x останавливается не более чем за t шагов. ▼

313. Показать, что при построении проекции язык из разрешимого может стать неразрешимым. ▼

314. Показать, что функция arc растёт медленнее каждой вычислимой функции: если для о. р. ф. f выполнено $f(x) \leqslant \mathrm{arc}(x)$ для всех x, то f ограничена. ▼

Ответы и решения

1. (а) Если $a \in A \cap B$, то $a \in A$; если $a \in A$, то $a \in A \cup B$. (б) Если $a \in A \setminus B$, то $a \in A$. **2.** (в) Если $x \in (A \cup B) \cap A$, то $x \in A$. Если $x \in A$, то $x \in (A \cap B) \cup A$. Если $x \in (A \cap B) \cup A$, то $x \in A \cap B$ или $x \in A$, но из первого также следует, что $x \in A$, поэтому $x \in A$ в любом случае. Но тогда $x \in A \cup B$ и $x \in (A \cup B) \cap A$. (г) Если $x \in A \setminus (B \cup C)$, то $x \in A$ и $x \notin B \cup C$. Последнее означает $x \notin B$ и $x \notin C$, следовательно, $x \in A \setminus B$ и $x \in A \setminus C$. Получаем, $x \in (A \setminus B) \cap (A \setminus C)$. Если $x \in (A \setminus B) \cap (A \setminus C)$, то $x \in A \setminus B$ и $x \in A \setminus C$, то есть $x \in A$, $x \notin B$ и $x \notin C$. Следовательно, $x \notin B \cup C$ и $x \in A \setminus (B \cup C)$. (д) Если $x \in A \setminus (B \setminus C)$, то $x \in A$ и $x \notin B \setminus C$. Последнее означает, что $x \notin B$ или $x \in C$. В первом случае $x \in A \setminus B$, во втором — $x \in A \cap C$, то есть $x \in (A \setminus B) \cup (A \cap C)$ в любом случае. Если $x \in (A \setminus B) \cup (A \cap C)$, то $x \in A \setminus B$ или $x \in A \cap C$. В первом случае $x \in A$ и $x \notin B$, во втором — $x \in A$ и $x \in C$. В любом случае выполнено $x \in A$, а также — $x \notin B \setminus C$. Следовательно, $x \in A \setminus (B \setminus C)$. (к) Если $x \in A \dot{-} B$, то $x \in A \setminus B$ или $x \in B \setminus A$. Если $x \in A \setminus B$, то $x \in A$ и $x \notin B$, следовательно, $x \in A \cup B$ и $x \notin A \cap B$, то есть $x \in (A \cup B) \setminus (A \cap B)$. Аналогично в случае $x \in B \setminus A$. Если $x \in (A \cup B) \setminus (A \cap B)$, то $x \in A \cup B$ и $x \notin A \cap B$. Если $x \in A$, то $x \notin B$, следовательно, $x \in A \setminus B$ и $x \in A \dot{-} B$. Аналогично в случае $x \in B$. **3.** $\varnothing$; $\varnothing$, $\{\varnothing\}$; $\varnothing$, $\{1\}$, $\{2\}$, $\{3\}$, $\{1,2\}$, $\{1,3\}$, $\{2,3\}$, $\{1,2,3\}$; $\varnothing$, $\{a\}$, $\{\{1,2\}\}$, $\{\varnothing\}$, $\{a,\{1,2\}\}$, $\{a,\varnothing\}$, $\{\{1,2\},\varnothing\}$, $\{a,\{1,2\},\varnothing\}$. **4.** Пусть $A \subseteq B$ и $x \in \mathsf{P}(A)$, тогда $x \subseteq A$, следовательно, $x \subseteq B$ и $x \in \mathsf{P}(B)$. Если $\mathsf{P}(A) \subseteq \mathsf{P}(B)$ и $x \in A$, то $\{x\} \subseteq A$, поэтому $\{x\} \subseteq B$ и $x \in B$. **5.** $A \times B = \{(0,a),(0,b),(0,c),(1,a),(1,b),(1,c)\}$; $B \times A = \{(a,0),(b,0),(c,0),(a,1),(b,1),(c,1)\}$. **6.** (г) Если $(x,y) \in A \times C$, то

$x \in A$, $y \in C$, следовательно, $x \in B$, $y \in D$. Тогда $(x, y) \in A \times D$, $(x, y) \in B \times C$, поэтому $(x, y) \in (A \times D) \cap (B \times C)$. Если $(x, y) \in (A \times D) \cap (B \times C)$, то $(x, y) \in A \times D$, $(x, y) \in B \times C$. Из первого получаем $x \in A$, из второго — $y \in C$. Следовательно, $(x, y) \in A \times C$. **7.** (а) $\operatorname{dom} R = \omega$, $\operatorname{rng} R = \omega$, $R^{-1} = \{(x, y) : x, y \in \omega \text{ и } y \mid x\}$, $R \circ \circ R = R$, $R \circ R^{-1} = \omega^2$; (б) $\operatorname{dom} R = [0; 10]$, $\operatorname{rng} R = [0; 10]$, $R^{-1} = R$, $R \circ R = R \circ R^{-1} = \{(x, y) : x, y \in \omega \text{ и } x + y \leqslant 20\}$; (в) $\operatorname{dom} R = \omega$, $\operatorname{rng} R = \{y \in \omega : y \equiv 1 \pmod 3\}$, $R^{-1} = \{(x, y) : x, y \in \omega \text{ и } x = 3y + 1\}$, $R \circ \circ R = \{(x, y) : x, y \in \omega \text{ и } y = 9x^2 + 3x + 1\}$, $R \circ R^{-1} = \mathrm{id}_\omega$; (г) $\operatorname{dom} R = [0; 10]$, $\operatorname{rng} R = \{0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100\}$, $R^{-1} = \{(x^2, x) : x \in \omega \text{ и } x \leqslant 10\}$, $R \circ R = \{(0, 0), (1, 1), (2, 16), (3, 81)\}$, $R \circ R^{-1} = \mathrm{id}_{[0;10]}$; (д) $\operatorname{dom} R = \{a, b, c, d\}$, $\operatorname{rng} R = \{b, c, d\}$, $R^{-1} = \{(b, a), (c, b), (d, b), (d, c), (b, d)\}$, $R \circ R = \{(a, c), (a, d), (b, d), (b, b), (c, b), (d, c), (d, d))\}$, $R \circ R^{-1} = \{(a, a), (a, d), (b, b), (b, c), (c, c), (c, b), (d, a), (d, d)\}$. **8.** $L = \{((x, y), (u, v)) : x = u \text{ или } y = v, 1 \leqslant x, y, u, v \leqslant 8\}$, $K = \{((x, y), (x + i, y + j)) : |ij| = 2, i, j \in \mathbb{Z}, 1 \leqslant x, y, x + i, y + j \leqslant 8\}$, не будут — не транзитивны, $L \circ L = S^2$. **9.** (а) будет, (б) нет. **10.** Будут (а), (б), (д), (ж). **11.** (а) строгий частичный порядок, линейный только при $m = 1$: слова вида $a_1 a_2$ и $a_2 a_1$ несравнимы; (б) строгий линейный порядок: указанный мтод даёт ответ для любых неравеных слов. **12.** Если $(v, w) \in R_1$, то в словах v и w отсутствует место, где было бы выполнено $v_i > w_i$, Следовательно, не может выполняться $(w, v) \in R_2$, в силу линейнойсти $(v, w) \in R_2$. **13.** Пусть n — самый большой элемент Y. Тогда $X \subseteq Y$ соответствует покоординатному порядку на $(n + 1)$-разрядном двоичном представлении натуральных чисел, а $\rho(X) \leqslant \rho(Y)$ — лексикографическому (см. предыдущую задачу). $\rho(\{0\}) < \rho(\{1\})$. **14.** (а) Для каждого $a \in A$ в $A \times B$ есть в точности $|B|$ пар вида (a, b), $b \in B$. Поэтому общее количество пар $|A| \cdot |B|$. (б) Чтобы найти количество элементов $A \cup B$ нужно к $|A|$ добавить количество элементов B, отсутствующих в A. Последнее равно $|B| - |A \cap B|$. **15.** Построить функцию такую, что в десятичной записи числа $f(x, y)$ чередуются цифры чисел x и y. **16.** Использовать индукцию по n и предыдущую задачу. **17.** Пусть A_i, $i \in \omega$ — счётные множества, $f_i : A_i \to \omega$ — разнозначные функции, $A = \bigcup\limits_i A_i$. Рассмотреть функцию $g : A \to \omega^2$, где $g(x) = (i, f_i(x))$, если $x \in A_i$. **18.** Представить Q как объединение счётных множеств вида

$Q_i = \{x/(i+1) : x \in bZ\}$. **19.** Для многочленов вида $a_0 + a_1x + \dots + a_nx^n$ воспользоваться задачей 16. **20.** Воспользоваться задачами 19 и 17. **21.** Воспользоваться задачами 20, 17 и фактом, что ненулевой многочлен имеет конечно много корней. **22.** $|\mathbb{R}| \leqslant |S|$ тривиально: $g(r) = (r)_{i\in\omega}$. Чтобы построить $h : S \to \mathbb{R}$, воспользуемся функцией f из теоремы 6. Если $f(a_i) = 0, a_{i1}a_{i2}a_{i3}\dots$ в десятичной записи, то $h((a_i)_{i\in\omega})$ можно построить так: на нечётных позициях ставим 0, на делящихся на 2, но не на 4 — цифры a_{0i} числа $f(a_0)$, на делящихся на 4, но не на 8 — цифры a_{1i} числа $f(a_1)$ и т. д. **23.** Аналогично задаче 20 на стр. 41. **24.** $L_1 \cap L_2 = \{a^ib^i : i \in \omega\}$; $L_2 \setminus L_1 = \{a^ib^j : i, j \in \omega, i \neq j\}$; $\bar{L}_1 = \{w \in \Sigma^* :$ количества букв a и b в w различаются$\}$. **25.** Префиксы: ε, a, a^2, a^2b, a^2b^2, a^2b^2c; суффиксы: ε, c, bc, b^2c, ab^2c, a^2b^2c. **26.** Базис: $n = 0$, $0 = 0$. Если для n доказано, то $1^4 + 2^4 + \dots + n^4 + (n+1)^4 = n(n+1)(2n+1)(3n^2+3n-1)/30 + (n+1)^4 = (n+1)(n(2n+1)(3n^2+3n-1)+30(n+1)^3)/30 = (n+1)(6n^4+6n^3-2n^2+3n^3+3n^2-n+30n^3+90n^2+90n+30)/30(n+1)(6n^4+39n^3+91n^2+89n+30)/30$ и $(n+1)((n+1)+1)(2(n+1)+1)(3(n+1)^2+3(n+1)-1)/30 = (n+1)(n+2)(2n+3)(3n^2+9n+5)/30 = (n+1)(2n^2+7n+6)(3n^2+9n+5)/30 = (n+1)(6n^4+18n^3+10n^2+21n^3+63n^2+35n+18n^2+54n+30)/30 = (n+1)(6n^4+39n^3+91n^2+89n+30)/30$. **27.** Базис: $n = 0$, $0 = 0$. Если для n доказано, то $1\cdot 2 + 2\cdot 3 + \dots + n(n+1) + (n+1)(n+2) = n(n+1)(n+2)/3 + (n+1)(n+2) = (n(n+1)(n+2)+3(n+1)(n+2))/3 = (n+1)(n+2)(n+3)/3$. **28.** Базис: $n = 0$, $1 = (x-1)/(x-1)$. Если для n доказано, то $1 + x + x^2 + x^3 + \dots + x^n + x^{n+1} = (x^{n+1}-1)/(x-1) + x^{n+1} = ((x^{n+1}-1) + x^{n+1}(x-1))/(x-1) = (x^{n+1}-1+x^{n+2}-x^{n-1})/(x-1) = (x^{n+2}-1)/(x-1)$. **29.** Для $n = 0$ всю плоскость закрашиваем одним цветом. После проведения $(n+1)$-й прямой в одной из половин, на которые эта прямая делит плоскость, меняем чёрный и белый цвета местами. **30.** Ошибка: нет базиса индукции. Неравенство справедливо при $k \geqslant 3$. **31.** Неверен индукционный шаг при переходе от $n = 1$ к $n = 2$: в стаде не будет коров кроме x и y, поэтому их не с кем сравнивать. **32.** Базис: $F_0 = 0$ — чётное, $F_1 = F_2 = 1$ — нечётные. Если для всех $j < i$ доказано и i делится на 3, то $i-1$ и $i-2$ не делятся на 3, F_{i-1} и F_{i-2} — нечётные, F_i — чётное. Если i не делится на 3, то одно и только одно из чисел $i-1$ или $i-2$ делится на 3, поэтому одно из чисел

F_{i-1}, F_{i-2} чётное, другое — нечётное, следовательно, F_i — нечётное. **33.** При $i=1$: $F_{1-1}F_{1+1}=0\cdot 1=0=1-1=F_1^2+(-1)^1$. Если для i доказано, то $F_{i+1-1}F_{i+1+1}=F_iF_{i+2}=F_i(F_{i+1}+F_i)=F_iF_{i+1}+F_i^2=F_iF_{i+1}+F_{i-1}F_{i+1}-(-1)^i=(F_i+F_{i-1})F_{i+1}-(-1)^i=F_{i+1}^2+(-1)^{i+1}$. **34.** Получаем $A_n=\sum\limits_{m=0}^{n}A_n^m=\sum\limits_{m=0}^{n}n!/(n-m)!=n!\sum\limits_{m=0}^{n}1/(n-m)!=n!\sum\limits_{m=0}^{n}1/m!=e_nn!$. Так как e_n стремится к e при возрастании n, то при больших n получаем $A_n\approx en!$. **35.** (а) $C_{n-1}^k+C_{n-1}^{k-1}=(n-1)!/k!(n-1-k)!+(n-1)!/(k-1)!(n-1-(k-1))!=(n-1)!(n-k)/k!(n-k)!+(n-1)!k/k!(n-k)!=(n-1)!(n-k+k)/k!(n-k)!=n!/k!(n-k)!=C_n^k$; (б) $nC_{n-1}^{k-1}=n\times(n-1)!/(k-1)!(n-1-(k-1))!=n!/(k-1)!(n-k)!=n!k/k!(n-k)!=kC_n^k$; (в) $C_n^kC_{n-k}^{m-k}=n!/k!(n-k)!\times(n-k)!/(m-k)!(n-k-(m-k))!=n!/k!(m-k)!(n-m)!=n!m!/k!m!(m-k)!(n-m)!=m!/k!(m-k)!\times n!/m!(n-m)!=C_m^kC_n^m$. **36.** Для $n=0$: $(x+y)^0=1=C_0^0x^0y^0$. Если для n доказано, то $(x+y)^{n+1}=(x+y)^n(x+y)=(\sum\limits_{i=0}^{n}C_n^ix^iy^{n-i})(x+y)=(\sum\limits_{i=0}^{n}C_n^ix^{i+1}y^{n-i})+(\sum\limits_{i=0}^{n}C_n^ix^iy^{n-i+1})=(\sum\limits_{i=0}^{n+1}C_n^{i-1}x^{i-1+1}y^{n-(i-1)})+(\sum\limits_{i=0}^{n+1}C_n^ix^iy^{n+1-i})=(\sum\limits_{i=0}^{n+1}C_n^{i-1}x^iy^{n+1-i})+(\sum\limits_{i=0}^{n+1}C_n^ix^iy^{n+1-i})=(\sum\limits_{i=0}^{n+1}(C_n^{i-1}+C_n^i)x^iy^{n+1-i})=(\sum\limits_{i=0}^{n+1}C_n^ix^iy^{n+1-i})$. **37.** При $i<[n/2]$ получаем $(n-i)/(i+1)>1$, поэтому $C_n^i=n!/i!(n-i)!<n!/i!(n-i)!\times(n-i)/i+1=n!/(i+1)!(n-i-1)!=C_n^{i+1}$. Для $i>[n/2]$ используем равенство $C_n^i=C_n^{n-i}$. **38.** Базис: $k=1$, тогда существует одно разбиение, $n_1=n$ и $n!/n_1!=1$. Если для k доказано, то любое разбиение на $k+1$ множеств можно получить так: сначала отделить $(k+1)$-е множество от остального, а затем, разбить остаток на k множеств. Количество вариантов для $(k+1)$-го: $C_n^{n_{k+1}}=n!/n_{k+1}!(n-n_{k+1})!$, количество варинтов для оставшегося разбиения по индукционному предположению равно $(n-n_{k+1})!/n_1!\cdots n_k!$. Всего вариантов: $n!/n_{k+1}!(n-n_{k+1})!\times(n-n_{k+1})!/n_1!\cdots n_k!=n!/n_1!\cdots n_k!n_{k+1}!$. **39.** Нужно найти наименьшее n такое, что $C_n^3\geqslant 20$. Следовательно, $n=6$: $C_6^3=20$. **40.** 2^{30} вариантов: (а)–(г) да, (д) нет. **41.** Индукция по m. При $m=0$: $C_{n+0}^k=C_n^kC_0^0=\sum\limits_{i=0}^{i=k}C_n^iC_0^{k-i}$, так как при $k\neq i$ будет $C_0^{k-i}=0$. Если для m

доказано, то $C^k_{n+(m+1)} = C^k_{n+m} + C^{k-1}_{n+m} = \sum\limits_{i=0}^{i=k} C^i_n C^{k-i}_m + \sum\limits_{i=0}^{i=k-1} C^i_n C^{k-1-i}_m =$ $C^k_n C^0_m + \sum\limits_{i=0}^{i=k-1} C^i_n C^{k-i}_m + \sum\limits_{i=0}^{i=k-1} C^i_n C^{k-1-i}_m = C^k_n C^0_{m+1} + \sum\limits_{i=0}^{i=k-1} C^i_n (C^{k-i}_m +$ $C^{k-1-i}_m) = C^k_n C^{k-k}_{m+1} + \sum\limits_{i=0}^{i=k-1} C^i_n C^{k-i}_{m+1} = \sum\limits_{i=0}^{i=k} C^i_n C^{k-i}_{m+1}$. Другой способ доказательства: обе части этого равенства задают количество вариантов выбора k человек из группы, состоящей из n женщин и m мужчин (правая часть равенства — всевозможные варианты распределения количеств мужчин и женщин в этой группе). **42.** Базис: $n = 0$, $\rho^{-1}(k) \subseteq \rho^{-1}(0)$ выполнено только при $k = 0$, при этом $C^0_0 = 1$ и $C^k_0 = 0$ при других k. Аналогично для $n = 1$, $\rho^{-1}(k) \subseteq \rho^{-1}(1)$ выполнено при $k \in \{0, 1\}$, при этом $C^0_1 = C^1_1 = 1$ и $C^k_0 = 0$ при других k. Пусть для всех чисел меньших n доказано и $n = 2^u + m$, где $1 \leqslant m \leqslant 2^u$. Используем тождество Коши: $C^k_n = C^k_{2^u+m} = \sum\limits_{i=0}^{k} C^i_{2^u} C^{k-i}_m$. Так как 2^u и m меньше n, то для них можно пользоваться индукционным предположением. Если $k < 2^u$, то условие $\rho^{-1}(i) \subseteq \rho^{-1}(2^u)$ выполнено только при $i = 0$, поэтому единственное слагаемое, которое может быть нечётным, — это $C^0_{2^u} C^k_m = C^k_m$. Оно нечётно тогда и только тогда, когда $\rho^{-1}(k) \subseteq \rho^{-1}(m)$, то есть $\rho^{-1}(k) \subseteq \rho^{-1}(n)$, поскольку единственный элемент $\rho^{-1}(n) \setminus \rho^{-1}(m)$ равен u и в $\rho^{-1}(k)$ не содержится. Если $k = 2^u$, то условие $\rho^{-1}(i) \subseteq \rho^{-1}(2^u)$ выполнено при $i = 0$ и при $i = 2^u$, поэтому есть только два слагаемых, которые могут быть нечётными, — это $C^0_{2^u} C^k_m = C^k_m$ и $C^{2^u}_{2^u} C^0_m = 1$. Второе нечётно всегда, а первое — при $m = 2^u$. Таким образом, для $k = 2^u$ число C^k_n чётно при $n = 2^{u+1}$ и нечётно при $n < 2^{u+1}$. Последнее условие эквивалентно $\rho^{-1}(k) \subseteq \rho^{-1}(n)$. Если $k > 2^u$, то условие $\rho^{-1}(i) \subseteq \rho^{-1}(2^u)$ также выполнено при $i = 0$ и при $i = 2^u$, поэтому есть только два слагаемых, которые могут быть нечётными, — это $C^0_{2^u} C^k_m = C^k_m$ и $C^{2^u}_{2^u} C^\ell_m = C^\ell_m$, где $\ell = k - 2^u$. Но $m \leqslant 2^u < k$, поэтому первое из них равно нулю, следовательно, C^k_n нечётно тогда и только тогда, когда нечётно C^ℓ_m, что по индукционному предположению эквивалентно $\rho^{-1}(\ell) \subseteq \rho^{-1}(m)$, то есть $\rho^{-1}(k) = \rho^{-1}(\ell) \cup \{u\} \subseteq \rho^{-1}(m) \cup \{u\} = \rho^{-1}(n)$. **43.** Пусть мультимножество составляются из натуральных чисел $1, \ldots, n$. Индукция по $k + n$. Базис: $k + n = 1$. Если $k = 0$, $n = 1$, то существует только одно, пустое, мультимножество и $C^0_{1+0-1} = 1$. Если $k = 1$, $n = 0$, то таких мульти-

множеств нет и $C^1_{0+1-1} = 0$. Пусть для всех $k+n < m$ доказано, рассмотрим случай $k + n = m$. Все k-элементные мультимножества можно разделить на два типа: содержащие n и не содержащие n. Если из первых убрать одно n то останутся $(k-1)$-элементные мультимножества, составленные из $1, \ldots, n$, по индукционному предположению их $C^{k-1}_{n+k-1-1} = C^{k-1}_{n+k-2}$. Вторые состоавлены из элементов $1, \ldots, n-1$ их по индукционному предположению $C^k_{n-1+k-1} = C^k_{n+k-2}$. Всего получаем $C^{k-1}_{n+k-2} + C^k_{n+k-2} = C^k_{n+k-1}$. **44.** C^6_{4+6-1}. **45.** $A_1 6^3 \times C^2_{13}$. **46.** Поставить белые шары, а затем чёрные на $n-m+1$ место между ними и по краям: C^m_{n-m+1}. **47.** Если пронумеровать места за столом натуральными числами $1, \ldots, n$, то требуется из n выбрать k мест так, чтобы не были выбраны одновременно первое и последнее, а также — любые два числа подряд. Используем предыдущую задачу: если выбрать 1, то останется выбрать $k-1$ число из $3, \ldots, n-1$, то есть C^{k-1}_{n-k-1}. Если 1 не выбирать, то останется выбрать k чисел из $2, \ldots, n$, то есть C^k_{n-k}. Итого: $C^{k-1}_{n-k-1} + C^k_{n-k}$. **48.** Взять в качестве множества объектов $\bigcup\limits_{i=1}^{n} A_i$, а в качестве p_i свойство «не принадлежать A_i». Тогда $\left|\bigcap\limits_{i\in I} A_i\right| = N\{p_i : i \in I\}$ и $N\{p'_1, \ldots, p'_n\} = 0$. Из теоремы 15 получаем $0 = \sum\limits_{I\subseteq\{1,\ldots,n\}} (-1)^{|I|} \left|\bigcap\limits_{i\in I} A_i\right|$. Выделяя из суммы слагаемое для $I = \varnothing$, получаем требуемое. **49.** $100 - [100/2] - [100/3] - [100/5] + [100/6] + [100/10] + [100/15] - [100/30] = 26$, 266. **50.** $x + y = 11 - z$, $x + y \geqslant 4$, $4 - x \leqslant y \leqslant 3$. При $x = 1$ — одно решение, при $x = 2$ — два, при $x = 3$ — три, при $x = 4$ — четыре. Всего — десять. **51.** Общее число перестановок $n!$. Число перестановок, при котором выбранные k элементов остаются неподвижными: $(n-k)!$. Получаем: $n! - C^1_n(n-1)! + C^2_n(n-2)! - C^3_n(n-3)! \cdots = n! - n!/1!(n-1)! \times (n-1)! + n!/2!(n-2)! \times (n-2)! - n!/3!(n-3)! \times (n-3)! \cdots = n! - n!/1! + n!/2! - n!/3! \cdots = n!(1 - 1/1! + 1/2! - 1/3! \ldots) \approx n!/e$. **52.** (а) $\{(0,0,0,0), (0,0,0,1), (0,0,1,0), (0,0,1,1), (0,1,0,0), (0,1,0,1), (0,1,1,0), (0,1,1,1)\}$; (б) $\{(0,0,0,1), (0,0,1,1), (0,1,0,1), (0,1,1,1), (1,0,0,1), (1,0,1,1), (1,1,0,1), (1,1,1,1)\}$; (в) $\{(0,0,0,0), (0,1,0,0), (1,0,0,0), (1,1,0,0), (0,1,0,1), (1,0,0,1), (1,1,0,1), (0,1,1,0), (1,0,1,0), (1,1,1,0), (1,1,1,1)\}$; (г) $\mathbb{B}^4 \setminus \{(1,1,0,0), (1,1,0,1), (1,1,1,0)\}$. **53.** (а) 0 только в строке $(0,1,0)$; (б) 1 в строках $(0,0,0)$, $(0,1,1)$, $(1,0,1)$, $(1,1,0)$; (в) 0 в строках $(0,1,1)$,

$(1,0,0)$; (г) 1 в строках $(0,0,1)$, $(0,1,1)$, $(1,1,0)$, $(1,1,1)$. **54.** (а) 0 в строке $(1,1,1)$; (б) 1 в строках $(0,0)$, $(1,0)$; (в) 1 в строках $(0,0,0)$, $(0,1,1)$, $(1,0,0)$. **55.** (а) Две константы; (б) две: x_n и $\neg x_n$. **56.** $2^{2^{n-1}}$. **57.** Существует $2^{n/2}$ парных наборов. Всего $2^{2^n-2^{n/2}} \times 2$. **58.** a — A корректна, b — B корректна, c — C корректна. $(a \to \neg b) \wedge (b \vee c) \wedge (a \vee c) \wedge (\neg a \to (b \oplus c))$. B ошибочна, единственность установить невозможно. **59.** a — «$x < 0$», b — «$y < 0$», c — «$z < 0$». $(c \to (a \vee \neg b)) \wedge (\neg a \to b) \wedge (b \to (a \oplus c))$. Нельзя. **60.** $y \wedge ((x_1 \wedge (x_2 \vee x_3 \vee x_4)) \vee x_2 \wedge (x_3 \vee x_4) \vee x_3 \wedge x_4)$. **61.** (а) a — «сделать ремонт», b — «сменить обои», c — «покрасить пол», d — «сменить ламинат», e — «сменить дверь», f — «сменить стеклопакет», g — «установить натяжной потолок», h — «сменить люстру», i — «сменить мебель», $a \wedge (b \vee c) \wedge (b \to d \wedge e \wedge f) \wedge (c \to g \wedge h \wedge i) \wedge (d \to i) \wedge \neg(\neg b \wedge \neg i) \wedge \neg(\neg i \wedge \neg h)$. (б) a — «будет мороз», b — «будет солнечно», c — «пойдёт снег», d — «пойдёт дождь», e — «будет сыро», f — «будет грязно», $(a \to (b \vee (\neg b \wedge c))) \wedge (\neg a \to (b \vee (\neg b \wedge d))) \wedge ((c \vee d) \to e) \wedge (d \to f) \wedge (b \to (\neg e \wedge \neg f))$. (в) a — «Вова дома», b — «кошка ест», c — «кошка спит», d — «кошка играет», e — «купить корм», f — «купить мебель», g — «купить обои», h — «съездить в зоомагазин», i — «съездить в хозяйственный магазин», j — «поезка на такси», $(a \to (b \vee c)) \wedge (\neg a \to d) \wedge (b \to e) \wedge (d \to (f \vee g)) \wedge (e \to h) \wedge ((f \vee g) \to i) \wedge ((h \vee i) \to j) \wedge \neg(j \wedge i)$. **62.** Всегда выполнено $\Phi \wedge \Psi \Rightarrow \Phi$. Если $\Phi \Rightarrow \Psi$, то $\Phi \Rightarrow \Phi \wedge \Psi$. Следовательно, $\Phi \wedge \Psi \equiv \Phi$. Если $\Phi \wedge \Psi \equiv \Phi$, то $\Phi \Rightarrow \Phi \wedge \Psi$. Это означает, что $\Phi \Rightarrow \Psi$. Всегда выполнено $\Psi \Rightarrow \Phi \vee \Psi$. Если $\Phi \Rightarrow \Psi$, то $\Phi \vee \Psi \Rightarrow \Psi$. Следовательно, $\Phi \vee \Psi \equiv \Psi$. Если $\Phi \vee \Psi \equiv \Psi$, то $\Phi \vee \Psi \Rightarrow \Psi$. Последнее означает $\Phi \Rightarrow \Psi$. **64.** Базис: $\operatorname{depth}(x)^x_\Psi = \operatorname{depth}(\Psi) = 0 + \operatorname{depth}(\Psi) = \operatorname{depth}\Phi + \operatorname{depth}\Psi$; $\operatorname{depth}(y)^x_\Psi = \operatorname{depth}(y) = 0 \leqslant \operatorname{depth}\Phi + \operatorname{depth}\Psi$. Шаг для конъюнкции: $\operatorname{depth}(\Phi_1 \wedge \Phi_2)^x_\Psi = \operatorname{depth}((\Phi_1)^x_\Psi \wedge (\Phi_2)^x_\Psi) = \max\{\operatorname{depth}(\Phi_1)^x_\Psi, \operatorname{depth}(\Phi_2)^x_\Psi\} + 1 \leqslant \max\{\operatorname{depth}(\Phi_1) + \operatorname{depth}(\Psi), \operatorname{depth}(\Phi_2) + \operatorname{depth}(\Psi)\} + 1 = \max\{\operatorname{depth}(\Phi_1), \operatorname{depth}(\Phi_2)\} + 1 + \operatorname{depth}(\Psi) = \operatorname{depth}(\Phi_1 \wedge \Phi_2) + \operatorname{depth}(\Psi)$. Аналогично для остальных булевых связок. $\operatorname{depth}(y)^x_{x\wedge y} = \operatorname{depth} y = 0 < 1 = 0 + 1 = \operatorname{depth}(y) + \operatorname{depth}(x \wedge y)$. **65.** (а) Использовать эквивалентность $\Phi \wedge 0 \equiv 0$. (б) Использовать эквивалентность $\Phi \vee 1 \equiv 1$. (в) Использовать эквивалентность $\Phi \wedge 1 \equiv \Phi$. (г) Использовать эквивалентность $\Phi \vee 0 \equiv \Phi$. **66.** (а) $\neg(x \vee \neg y) \wedge (x \to \neg y) \equiv (\neg x \wedge \neg\neg y) \wedge (\neg x \vee \neg y) \equiv (\neg x \wedge y) \wedge (\neg x \vee \neg y) \equiv$

$\neg x \wedge y \wedge \neg x \vee \neg x \wedge y \wedge \neg y \equiv \neg x \wedge y \vee \neg x \wedge 0 \equiv \neg x \wedge y \vee 0 \equiv \neg x \wedge y$; (б) $\neg[(x \wedge \neg y) \to (\neg x \vee z)] \equiv \neg[\neg(x \wedge \neg y) \vee (\neg x \vee z)] \equiv \neg\neg(x \wedge \neg y) \wedge \neg(\neg x \vee z) \equiv (x \wedge \neg y) \wedge (\neg\neg x \wedge \neg z) \equiv x \wedge \neg y \wedge x \wedge \neg z \equiv x \wedge \neg y \wedge \neg z$; (в) $(x \oplus y) \to (x \wedge \neg y) \equiv ((x \vee y) \wedge (\neg x \vee \neg y)) \to (x \wedge \neg y) \equiv \neg((x \vee y) \wedge (\neg x \vee \neg y)) \vee (x \wedge \neg y) \equiv (\neg(x \wedge y) \vee \neg(\neg x \wedge \neg y)) \vee (x \wedge \neg y) \equiv (\neg x \wedge \neg y) \vee (\neg\neg x \wedge \neg\neg y) \vee (x \wedge \neg y) \equiv (\neg x \wedge \neg y) \vee (x \wedge y) \vee (x \wedge \neg y) \equiv (\neg x \wedge \neg y) \vee x \wedge (y \vee \neg y) \equiv (\neg x \wedge \neg y) \vee x \wedge 1 \equiv (\neg x \wedge \neg y) \vee x$. **67.** Индукция по построению Φ. Базис: если $\Phi = x$, то $(\Phi)^x_\Theta = \Theta$ и $I(\Phi)^x_\Theta = I(\Theta) = J(x) = J(\Phi)$; если $\Phi = y$, то $(\Phi)^x_\Theta = y$ и $I(\Phi)^x_\Theta = I(y) = J(y) = J(\Phi)$. Шаг для конъюнкции: $I(\Phi_1 \wedge \Phi_2)^x_\Theta = I((\Phi_1)^x_\Theta \wedge (\Phi_2)^x_\Theta) = I((\Phi_1)^x_\Theta) \wedge I((\Phi_2)^x_\Theta) = J(\Phi_1) \wedge J(\Phi_2) = J(\Phi_1 \wedge \Phi_2) = J(\Phi)$. Аналогично для других связок. **68.** В условиях предыдущей задачи $I(\Phi)^x_\Theta = J(\Phi)$, $I(\Psi)^x_\Theta = J(\Psi)$. Тогда $I(\Phi)^x_\Theta = J(\Phi) = J(\Psi) = I(\Psi)^x_\Theta$. **69.** Если $f(\sigma_1, \ldots, \sigma_n) = 0$, то $\mathcal{C}_f$ содержит дизъюнкцию $\bigvee x_i^{\neg\sigma_i}$. Согласно (8) при $x_i = \sigma_i$ будет $x_i^{\neg\sigma_i} = 0$, следовательно, $\bigvee x_i^{\neg\sigma_i} = 0$, поэтому $\mathcal{C}_f(\sigma_1, \ldots, \sigma_n) = 0$. Если $\mathcal{C}_f(\sigma_1, \ldots, \sigma_n) = 0$, то существует дизъюнкция $\bigvee x_i^{\neg\tau_i}$ равная нулю. Следовательно, $x_i^{\neg\tau_i} = 0$ для всех x_i. Согласно (8) это означает, что $x_i \neq \neg\tau_i$, то есть $\sigma_i = x_i = \tau_i$. Итак, $\mathcal{C}_f$ содержит дизъюнкцию $\bigvee x_i^{\neg\sigma_i}$, по определению $\mathcal{C}_f$ это означает $f(\sigma_1, \ldots, \sigma_n) = 0$. **70.** (а) Если K — конъюнкция, а $x_1, \ldots, x_n$ — новые переменные, то $\mathcal{D} = \bigvee\limits_{(\sigma_1,\ldots,\sigma_n)\in\mathbb{B}^n} K \wedge \bigwedge\limits_i x_i^{\sigma_i}$. (б) Аналогично. **71.** Если $f(\sigma_1, \ldots, \sigma_k, \sigma_{k+1}, \ldots, \sigma_n) = 1$, то при $x_i = \sigma_i$ получаем $x_i^{\sigma_i} = 1$ и $x_1^{\sigma_1} \wedge \ldots x_k^{\sigma_k} \wedge f(\sigma_1, \ldots, \sigma_k, x_{k+1}, \ldots, x_n) = 1$, поэтому вся правая дизъюнкция истинна. Если правая дизъюнкция истинна при $x_i = \sigma_i$, то $x_1^{\tau_1} \wedge \ldots x_k^{\tau_k} \wedge f(\tau_1, \ldots, \tau_k, x_{k+1}, \ldots, x_n) = 1$ для каких-то $\tau_1, \ldots, \tau_k$. Из истинности $x_i^{\tau_i} = 1$ и (8) получаем $\tau_i = \sigma_i$. Следовательно, $f(\sigma_1, \ldots, \sigma_k, \sigma_{k+1}, \ldots, \sigma_n) = 1$. **72.** Заменить все применяемые законы на двойственные. **73.** $\mathcal{C}(x) = \mathcal{D}(x) = x$, $\mathcal{C}(\neg\Phi) \equiv \neg\mathcal{D}(\Phi)$, $\mathcal{C}(\Phi \wedge \Psi) = \mathcal{C}(\Phi) \wedge \mathcal{C}(\Psi)$, $\mathcal{C}(\Phi \vee \Psi) = \bigwedge\limits_{D_1\in\mathcal{C}(\Phi)} \bigwedge\limits_{D_2\in\mathcal{C}(\Psi)} (D_1 \vee D_2)$ и т. д. **74.** (а) $y \vee (\neg x \wedge \neg z)$; (б) $x_1 \wedge \neg x_2 \wedge x_3$; (в) $(x \wedge \neg y) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge \neg z \wedge u) \vee (\neg y \wedge \neg z \wedge u)$. **75.** Построить истинностные таблицы. **76.** При значениях переменной в i-й строке k-я элементарная конъюнкция в многочлене Жегалкина истинна только при $k \subseteq i$. Осюда следует (13). В (14) обе части содержат сумму по модулю два $N + M$ копий переменной x. **77.** $1 \oplus x \oplus xyz$. **78.** (а) $y \oplus yz \oplus x \oplus xyz$; (б)

$1 \oplus yz \oplus xy \oplus xyz$; (в) $1 \oplus y \oplus x$; (г) $z \oplus y \oplus x \oplus xy \oplus xyz$. **79.** $\neg x_1 \wedge \dots \wedge \neg x_n$. **80.** $\neg x = x \downarrow x$, $x \wedge y = (x \downarrow x) \downarrow (y \downarrow y)$, $x \vee y = (x \downarrow y) \downarrow (x \downarrow y)$. **81.** $f_3, f_4, f_5 \in \mathcal{S}_0$, $f_2, f_3, f_4, f_5 \in \mathcal{S}_1$, $f_3, f_4, f_5 \in \mathcal{S}$, $f_2, f_4, f_5 \in \mathcal{L}$, $f_3, f_5 \in \mathcal{M}$. **82.** $\{f_1, x, y\}$, $\{f_1, x\}$, $\{f_1\}$, так как f_1 единственная функция не сохраняющая единицу. **83.** Полна, $h \notin \mathcal{S}_1$. $0 = h(x, x, x)$, $1 = g(x, x, x)$, $\neg x = g(0, x, 0)$, $x \to y = \neg h(0, x, y)$. **84.** $\{\vee, \wedge, \to\} \subseteq \mathcal{S}_1$, можно добавить 0. **85.** $1 = x \to x$, $0 = \neg 1$, $x \vee y = \neg x \to y$, $x \wedge y = \neg(\neg x \vee \neg y)$, $x \leftrightarrow y = (x \wedge y) \vee (\neg x \wedge \neg y)$. **86.** $|\mathcal{S}_0| = |\mathcal{S}_1| = 2^{2^n - 1}$; $|\mathcal{S}| = 2^{2^{n-1}}$; $|\mathcal{L}| = 2^{n+1}$. **87.** Если $x_i = 0$, то первая конъюнкция равна нулю, результат равен второй. Если $x_i = 1$, то первая в силу монотонности имеет значение не меньшее, чем вторая, поэтому результат равен первой. **88.** Рассмотреть элементарные конъюнкции, содержащие ровно $[n/2]$ переменных. Тогда любая ДНФ, составленная из них (без повторений) даст уникальную монотонную функцию. **89.** Если $f = g_1 \oplus \alpha_1 x_1 = g_2 \oplus \alpha_2 x_2$, то $g_1 \oplus \alpha_2 x_2 = g_2 \oplus \alpha_1 x_1 = g_{12}$. Первая часть не зависит от x_1, вторая — от x_2, поэтому обе они не зависят от x_1 и x_2 и мы получаем $f = g_{12} \oplus \alpha_1 x_1 \oplus \alpha_2 x_2$. Далее по индукции. Обратно: в силу коммутативности и ассоциативности $\oplus$. **90.** В прямую сторону — по определению операции $\oplus$. Обратно: придать переменным значения как в лемме 34, рассмотреть восемь полученных вариантов. **91.** В прямую сторону — по определению, так как наборы удовлетворяют неравенствам для всех аргументов. В обратную: взять наборы $\sigma_i \leqslant \rho_i$, $i = 1, \dots, n$, и использовать индукцию по количеству i, для которых $\sigma_i < \rho_i$, то есть $\sigma_i = 0$, $\rho_i = 1$. На каждом шаге изменяется только один аргумент, поэтому значение функции не уменьшается. **92.** Если минимальный импликант для f содержит отрицание $D \wedge \neg a$, то D импликантом для f не является. Значит, существуе интерпретация I такая, что D равна единице, а f нулю. Тогда $I(\neg a) = 0$ и $I(a) = 1$. Если теперь изменить значение a на ноль, то получим интерпретацию J, в которой $J(D \wedge \neg a) = 1$ и f равно единице. Но тогда f немонотонна. В обратную сторону утверждение следует из монотонности конъюнкции и дизъюнкции, а также замкнутости множества монотонных функций. **93.** $\{0, \to\}$. **94.** Множество монотонных функций (задача 92). **95.** ДНФ сохраняет ноль, если нет элементарных конъюнкций, содержащих только переменные с отрицаниями, ДНФ сохраняет единицу, если есть элементарная конъюнкция, содержащая только переменные

без отрицаний. Для КНФ наоборот. **96.** Если $f = \alpha \oplus x_1 \oplus \dots \oplus x_n$, то $f^* = \alpha \oplus x_1 \oplus 1 \oplus \dots \oplus x_n \oplus 1 \oplus 1 = \beta \oplus x_1 \oplus \dots \oplus x_n$. Если $(\sigma_1, \dots, \neg\sigma_n) \leqslant (\neg\rho_1, \dots, \neg\rho_n)$, то $(\neg\sigma_1, \dots, \neg\sigma_n) \geqslant (\neg\rho_1, \dots, \neg\rho_n)$, поэтому $f(\neg\sigma_1, \dots, \neg\sigma_n) \geqslant f(\neg\rho_1, \dots, \neg\rho_n)$ и $\neg f(\neg\sigma_1, \dots, \neg\sigma_n) \leqslant \neg f(\neg\rho_1, \dots, \neg\rho_n)$. Свойства сохранения нуля и единицы меняются местами. **97.** 2^n. **98.** $n+2$: x_i и константы. **99.** Индукция по числу слагаемых в сумме: для x_i и $x_i \oplus 1 = \neg x_i$, количества нулей и единиц равны 2^{n+1}. При добавлении очередного слагаемого половина нулей превращаются в единцы, а половина единиц — в нули. Поэтому их количества остаются равными. Обратное неверно: функция $xy \oplus z$ нелинейна, хотя условию удовлетворяет. **100.** $(10xy\,zu00)$, $(11xy\,zu11)$, $(111x\,x000)$, $(110x\,x100)$, $(111x\,y100)$, $(110x\,y000)$, $(100x\,y010)$, $(101x\,y110)$, $(100x\,x110)$, $(101x\,x010)$, 56 функций. **101.** Пусть $I(x) = 1$ для всех переменных x. **102.** $\neg x_1 \vee \dots \vee \neg x_n \vee y \equiv x_1 \wedge \dots \wedge x_n \to y$. **103.** Если $\Phi = A \to x$ ложна в K, то $K(A) = 1$ и $K(x) = 0$. Из первого получаем $I(A) = J(A) = 1$, из второго — $I(x) = 0$ или $J(x) = 0$. Значит, в I или J формула Φ ложна. **104.** Если $D \vee a \vee b$ — минимальный антиимпликант формулы Φ, то $D \vee a$ и $D \vee b$ не следуют из Φ, следовательно, есть интерпретации I и J, в которых формула Φ истинна, а $D \vee a$ и $D \vee b$ соответственно ложны. Тогда в конъюнкции интерпретаций IJ будет ложно $D \vee a \vee b$. Следовательно, Φ не может быть конъюнкцией хорновских формул (задача 103). Если минимальный антиимпликант не содержит переменных без отрицаний, то он ложен в интерпретации, где истинны все переменные, тогда ложна и сама формула, что для хорновских формул невозможно (задача 101). В обратную сторону — задача 102. **105.** Процессы τ'_{i+1} запускаются, только при $L_t \subseteq X_i$. По индукции доказываем, что перед τ'_{i+1} доступны продукты из X_i. Базис: для $i = 0$ продукты X_0 имеются изначально, для $i = 1$ продукты X_0 есть изначально, а продукты $X_1 \setminus X_0$ получаются в результате τ_1. Если для i доказано, то после τ_{i+1} продукты X_i имеются по индукционному предположению, а продукты $X_{i+1} \setminus X_i$ получены в результате τ'_{i+1}. **106.** Для $k = 0$ это выполнено, так как $N_0 = X$, никаких процессов не нужно. Пусть N_k содержит все продукты, которые можно получить из X цепочками длины k и менее, а продукт z получается цепочкой длины не более $k+1$. Если $z \in N_k$, то $z \in N_{k+1}$ в силу $N_k \subseteq N_{k+1}$. Если $z \notin N_k$, то

пусть t — последний процесс в цепочке, порождающей z. Все исходные продукты y_i для этого процесса порождаются цепочками длины не более k, по индукционному предположению $y_i \in N_k$. Тогда условие $L_t \subseteq N$ выполнено на $(k+1)$-м шаге, поэтому $z \in N_{k+1}$. **107.** Создать в начале пустой список T, перед строкой 11 вставить **Если** $b_t \notin N$ **то**$T \leftarrow (T, t)$. **108.** $x_1 \wedge \cdots \wedge x_n \to y_1 \wedge \cdots \wedge y_m \equiv \bigwedge\limits_i (x_1 \wedge \cdots \wedge x_n \to y_i)$. **109.** Заменить в строке 11 присваивание на $N \leftarrow N \cup R_t$, где R_t — множество получаемых в процессе t продуктов. **110.** (6), (3), (4), (5), (2). **111.** $N_0 = \{c, d\}$, $N_1 = \{c, d, k\}$, $N_2 = \{c, d, k, h\}$, $N_3 = \{c, d, k, h, g\}$, $N_4 = \{c, d, k, h, g, b\}$, $N_5 = \{c, d, k, h, g, b, a\}$, $N_6 = \{c, d, k, h, g, b, a, e\}$, $N_7 = \{c, d, k, h, g, b, a, e, f\}$. $d, c \to k$, $c, d, k \to h$, $h, d, c \to g$, $d, g \to b$, $b, k \to a$. **112.** Докажем (а) индукцией по k. Для $k = 0$ это выполнено в силу изначальных присваиваний $C[t] \leftarrow |L_t|$ и $N = A = X$. Пусть утверждение выполнено для k. Заметим, что $N_{k+1} \setminus A_{k+1} = N_k = (N_k \setminus A_k) \cup A_k$. По индукционному предположению $C_k[t] = |L_t \setminus (N_k \setminus A_k)|$. На $k+1$ итерации цикла $C[t]$ будет уменьшено на единицу столько раз, сколько выполнится условие $t \in L[a]$ для $a \in A_k$, то есть $a \in L_t$. Следовательно, $C_{k+1}[t] = C_k[t] - |L_t \cap A_k| = |L_t \setminus (N_k \setminus A_k)| - |L_t \cap A_k| = |(L_t \setminus N_k) \cup (L_t \cap A_k)| - |L_t \cap A_k| = |L_t \setminus N_k| = |L_t \setminus (N_{k+1} \setminus A_{k+1})|$, что и требуется. Докажем (б) индукцией по k. Для $k = 0$ это выполнено, так как $N_0 = X$, никаких процессов не нужно. Пусть N_k содержит все продукты, которые можно получить из X цепочками длины k и менее, а продукт z получается цепочкой длины не более $k+1$. Если $z \in N_k$, то $z \in N_{k+1}$ в силу $N_k \subseteq N_{k+1}$. Если $z \notin N_k$, то пусть t — последний процесс в цепочке, порождающей z. Все исходные продукты y_i для этого процесса порождаются цепочками длины не более k, по индукционному предположению $y_i \in N_k$, то есть $L_t \subseteq N_k$. Согласно пункту (а), $C_{k+1}[t] = |L_t \setminus (N_{k+1} \setminus A_{k+1})| = |L_t \setminus N_k| = 0$. Но тогда должно быть выполнено условие в строке 22, поэтому $Z \in N_{k+1}$. Пункт (в) выполнен, так как новый элемент b_t в строках 23–24 добавляется в D и N одновременно. **113.** Создать в начале пустой список T, после строки 25 вставить $T \leftarrow (T, t)$. **114.** $C_0 = (3, 4, 2, 2, 2, 3)$, $A_0 = \{a, f\}$, $N_0 = \{a, f\}$, $C_1 = (2, 3, 1, 0, 2, 1)$, $A_1 = \{d\}$, $N_1 = \{a, f, d\}$, $C_2 = (2, 2, 1, 0, 1, 0)$, $A_2 = \{g\}$, $N_2 = \{a, f, d, g\}$, $C_3 = (2, 1, 1, 0, 0, 0)$, $A_3 = \{e\}$, $N_3 = \{a, f, d, g, e\}$, $C_4 = (2, 1, 0, 0, 0, 0)$, $A_4 = \{c\}$,

$N_4 = \{a, f, d, g, e, c\}$, $C_5 = (1, 0, 0, 0, 0, 0)$, $A_5 = \{h\}$, $N_5 = \{a, f, d, g, e, c, h\}$, $C_6 = (1, 0, 0, 0, 0, 0)$, $A_5 = \varnothing$, $N_5 = \{a, f, d, g, e, c, h\}$. $f, a \to d$, $d, f, a \to g$, $g, d \to e$, $e, f \to c$, $a, c, d, g \to h$. **115.** (а) $(\forall x)((\exists z)Q(x, z, y) \to (\forall x)(Q(z, x, y) \to (\exists y)(Q(z, y, x) \vee x \approx z)))$; (б) $(\exists y)((\exists x)(\neg x \approx y \wedge Q(x, x, y) \wedge (\forall x)(Q(x, z, y) \to (\exists z)(Q(x, y, z) \vee Q(z, z, y)))))$; (в) $(\forall x)((\exists z)Q(z, z, x) \wedge (\exists x)Q(y, y, x)) \vee (\exists x)((\exists z)Q(x, x, z) \to (\forall x)(Q(x, y, y)))$; (г) $(\exists y)Q(x, y, z) \vee (\exists x)(Q(z, x, y) \wedge (\forall y)Q(y, x, y)) \wedge (\forall z)(\exists x)Q(z, z, x)$. **116.** (а) Возможные первые две, результат не меняется. (б) Возможны первые четыре, результат не меняется. (в) Возможно две первые и две последние, результат не меняется, $(\Phi)_z^y = (\forall x)((\exists z)Q(z, z, x) \wedge (\exists x)Q(z, z, x)) \vee (\exists x)((\exists z)Q(x, x, z) \to (\forall x)(Q(x, z, z)))$. (г) $(\Phi)_z^x = (\exists y)Q(z, y, z) \vee (\exists x)(Q(z, x, y) \wedge (\forall y)Q(y, x, y)) \wedge (\forall z)(\exists x)Q(z, z, x)$, $(\Phi)_z^y = (\exists y)Q(x, y, z) \vee (\exists x)(Q(z, x, z) \wedge (\forall y)Q(y, x, y)) \wedge (\forall z)(\exists x)Q(z, z, x)$. **117.** (а) $M(x) \wedge (\exists z)(P(y, z) \wedge P(z, x))$; (б) $(\exists y)(\exists z)(P(x, y) \wedge P(x, z) \wedge \neg y \approx z)$; (в) $(\exists z)(P(x, z) \wedge P(y, z) \wedge \neg(\exists u)(P(x, u) \wedge P(y, u) \wedge \neg u \approx z))$; (г) $(\exists z)(\exists y)(P(z, x) \wedge P(z, y) \wedge \neg y \approx x \wedge \neg M(y) \wedge \neg(\exists u)(S(y, u) \wedge M(u))$; (д) $M(x) \wedge \neg M(y) \wedge (\exists u)(\exists v)(\exists w)(P(u, v) \wedge P(v, x) \wedge P(u, w) \wedge P(w, y) \wedge \neg v \approx w)$; (е) $(\exists u)(S(x, u) \wedge \neg M(u)) \wedge (\exists y)(M(y) \wedge P(x, y) \wedge (\exists u)(S(y, u) \wedge \neg M(u)) \wedge \neg(\exists z)(\neg z \approx y \wedge M(z) \wedge P(x, z) \wedge (\exists u)(S(z, u) \wedge \neg M(u))))$. **118.** (а) $\mathrm{pos}(x) = (\exists z)M(z, z, x)$; (б) $\mathrm{less}(x, y) = (\exists z)(\mathrm{pos}(z) \wedge A(x, z, y) \wedge \neg x \approx y)$; (в) $\mathrm{surj}(f) = (\forall y)(\neg F(y) \to (\exists x)V(f, x, y))$; (г) $\mathrm{bij}(f) = \mathrm{surj}(f) \wedge (\forall y)(\forall x_1)(\forall x_2)(V(f, x_1, y) \wedge V(f, x_2, y) \to x_1 \approx x_2)$; (д) вторая производная синуса противоположна ему: $\sin(f) = (\exists f')(\exists f'')(\exists n)(\exists e)(D(f', f) \wedge D(f'', f) \wedge \mathrm{zero}(n) \wedge \mathrm{one}(e) \wedge (\forall x)(\forall y)(\forall z)(V(f, x, y) \wedge V(f'', x, z) \to A(y, z, n)) \wedge V(f, n, n) \wedge V(f', n, e))$; (е) наименьшее положительное число, синус которого равен нулю: $\pi(x) = (\exists s)(\exists n)(\sin(s) \wedge \mathrm{zero}(n) \wedge V(s, x, n) \wedge \mathrm{less}(n, x) \wedge (\forall u)(V(s, u, n) \wedge \mathrm{less}(n, u) \to u \approx x \vee \mathrm{less}(x, u)))$; (ж) неотрицательная число, синус произведения его на π равен нулю: $\Omega(x) = \mathrm{pos}(x) \wedge (\exists n)(\exists p)(\exists u)(\exists s)(\mathrm{zero}(n) \wedge \pi(p) \wedge M(x, p, u) \wedge \sin(s) \wedge V(s, u, n))$. **119.** (а) $\mathrm{even}(x) = (\exists u)A(u, u, x)$; (б) $\mathrm{coprime}(x, y) = (\forall z)(\forall u)(\forall v)(M(z, u, x) \wedge M(z, v, y) \to z \approx 1)$; (в) $\mathrm{less}(x, y) = (\exists z)(A(x, z, y) \wedge \neg z \approx 0)$; $\mathrm{between}(x, y, z) = (\mathrm{less}(x, z) \wedge \mathrm{less}(z, y)) \vee (\mathrm{less}(y, z) \wedge \mathrm{less}(z, x))$; (г) $\text{НОД}(x, y, z) = (\forall u)(\forall v)(M(x, u, y) \wedge M(x, v, z) \wedge \mathrm{coprime}(u, v))$; (д) $\text{НОК}(x, y, z) = (\forall u)(\neg \mathrm{less}(u, y) \wedge \mathrm{less}(u, z) \to (\exists v)M(u, v, x)) \wedge$

$(\forall w)(\mathrm{less}(w,x) \to (\exists u)(\neg\,\mathrm{less}(u,y) \wedge \mathrm{less}(u,z) \wedge \neg(\exists v)M(u,v,w)))$. **120.** (а) $L'(x,y,z) = L(x,y,z) \vee L(y,z,x) \vee L(z,x,y)$; $\mathrm{triangle}(x,y,z) = \neg L'(x,y,z)$; (б) $\mathrm{middle}(x,y,z) = L(y,x,z) \wedge E(y,x,x,z)$; (в) $\mathrm{bissect}(x,y,z,u) = (\exists y')(\exists z')(\exists u')(L(x,y,y') \wedge L(x,z,z') \wedge L(x,u,u') \wedge E(x,y',x,z') \wedge E(x,y',x,u') \wedge E(u',y',y',z'))$; (г) $\mathrm{right}(x,y,z) = (\exists u)(\mathrm{middle}(x,u,z) \wedge E(u,x,u,y))$; (д) $\mathrm{circle}(x,y,z,u) = (\exists o)(E(o,x,o,y) \wedge E(o,y,o,z) \wedge E(o,z,o,u))$; (е) $\mathrm{acute}(x,y,z) = (\exists x')(\exists y')(\exists z')(L(x,y',z) \wedge L(y,z',x) \wedge L(z,x',y) \wedge \mathrm{right}(x,x',y) \wedge \mathrm{right}(y,y',z) \wedge \mathrm{right}(z,z',x))$; (ж) $\mathrm{less}(x,y,z,u) = (\exists v)(L(z,v,u) \wedge \neg v \approx u \wedge E(x,y,z,v))$; (з) $\mathrm{inside}(x,y,z) = \mathrm{less}(y,x,y,z)$. **121.** (а)–(г) Сигнатура: $S^{(1)}(x)$: «x — студент», $D^{(1)}(x)$: «x — дисциплина», $T^{(2)}(x,y)$: «x изучает y», $R^{(3)}(x,y,z)$: «x сдал y на z баллов», $x <^{(2)} y$: «x меньше y». (а) $\neg(\forall x)(S(x) \to T(x,\text{анализ}) \wedge T(x,\text{история}))$; (б) $(\exists x)(S(x) \wedge \neg(\exists b)R(x,\text{ДМ},b) \wedge (\forall u)(\neg u \approx x \to (\exists b)R(u,\text{ДМ},b)))$; (в) $(\exists x)(S(x) \wedge (\forall y)(R(x,y,100) \wedge (\forall u)(\neg u \approx x \to (\exists y)(\exists b)(R(u,y,b) \wedge \neg b \approx 100)))$; (г) $(\exists b)(\exists x)(R(x,\text{ДМ},b) \wedge (\forall x)(\forall v)(R(x,\text{Инф},v) \to v < b))$. (д) Сигнатура: $B^{(1)}(x)$: «x — брадобрей», $S^{(2)}(x,y)$: «x бреет y». $(\exists b)(B(b) \wedge (\forall x)(S(b,x) \to \neg B(x,x)))$. (е) Сигнатура: $P^{(1)}(x)$: «x — политик», $T^{(1)}(x)$: «x — момент времени», $L^{(3)}(x,y,z)$: «x лжёт y в момент z». $(\exists p)(P(p) \wedge (\exists z)(\forall y)(\neg T(y) \to L(p,y,z))) \wedge (\exists p)(P(p) \wedge (\exists y)(\forall z)(T(z) \to L(p,y,z))) \wedge \neg(\exists x)(\forall y)(\forall z)(\neg T(y) \wedge T(z) \to L(p,y,z))$. **122.** $(\exists x_1)\ldots(\exists x_k)(\bigwedge\limits_{i\neq j} \neg x_i \approx x_j \wedge (\forall y)\bigvee\limits_i y \approx x_i)$. **123.** $(\forall x_1)\ldots(\forall x_n)(\exists y)(F(x_1,\ldots,x_n,y) \wedge \neg(\exists z)(\neg z \approx y \wedge F(x_1,\ldots,x_n,y)))$. **124.** (а) Ложна: взять $x = 1$; (б) истинна: каждое число либо чётно, либо нечётно; (в) истинна: если $x \mid y$ и $y \mid z$, то $x \mid z$; (г) ложна: утверждается, что каждое число является квадратом или кубом; (д) истинна: если $x^2 \neq x$ (то есть x не равен нулю или единице), то разность $x^2 - x$ больше или равна двум. **125.** (а) Истинна: каждый отрезок xy можно продолжить на такое же расстояние; (б) ложна: не каждые три точки x, y, z лежат на одной окружности; (в) истинна: у каждого отрезка xy есть серединный перпендикуляр uv; (г) истинна: на некотором отрезке yz есть точка u, расстояние от которой до x не совпадает с расстоянием до x от других точек v этого отрезка; (д) истинна: если $|xz| < |xy|$, то u — точка пересечения отрезка xy и окружности с центром x и радиусом xz.

126. (а) Ложна: даже если производные совпадают, исходные функции могут отличаться; (б) истинна: для любых двух функций можно построить их композицию; (в) ложна: не каждая функция имеет обратную; (г) истинна: если производные функций совпадают, то функции отличаются на константу. Если эта константа ноль, то сами функции тоже совпадают; (д) истинна, в качестве x можно взять экспоненту exp; (е) истинна, в качестве x можно взять $1/exp$. **127.** Истинны (а), (в), (г), (д), (е). **128.** Базис: формулы Θ вида $x \approx y$ и $R(x_1, \ldots, x_k)$ при $R \neq Q$ не меняются, поэтому $(\Theta)^Q_\Phi = (\Theta)^Q_\Psi$. Для формулы Θ вида $Q(z_1, \ldots, z_n)$ по определению получаем $(\Theta)^Q_\Phi = (\Phi)^{x_1 \ldots x_n}_{z_1 \ldots z_n}$ и $(\Theta)^Q_\Psi = (\Psi)^{x_1 \ldots x_n}_{z_1 \ldots z_n}$, поэтому $(\Theta)^Q_\Phi \equiv (\Theta)^Q_\Psi$ в силу предложения 42. Индукционный шаг аналогичен теореме 18. **130.** 1) Если $I((\exists x)(\Phi \vee \Psi)) = 1$, то $(I)^x_a(\Phi \vee \Psi) = 1$ для некоторого a. Следовательно, $(I)^x_a(\Phi) = 1$ или $(I)^x_a(\Psi) = 1$, поэтому $I((\exists x)\Phi) = 1$ или $I((\exists x)\Psi) = 1$, значит, $I((\exists x)\Phi \vee (\exists x)\Psi) = 1$. Если $I((\exists x)\Phi \vee (\exists x)\Psi) = 1$, то $I((\exists x)\Phi) = 1$ или $I((\exists x)\Psi) = 1$. Значит, существует a, для которого $(I)^x_a(\Phi) = 1$, или существует b, для которого $(I)^x_b(\Psi) = 1$. В первом случае получаем $(I)^x_a(\Phi \vee \Psi) = 1$ и $I((\exists x)(\Phi \vee \Psi)) = 1$, во втором — $(I)^x_b(\Phi \vee \Psi) = 1$ и $I((\exists x)(\Phi \vee \Psi)) = 1$. 2) Если $I((\forall x)(\Phi \wedge \Psi)) = 1$, то $(I)^x_a(\Phi \wedge \Psi) = 1$ для любого a. Поэтому для любого a получаем $(I)^x_a(\Phi) = 1$ и $(I)^x_a(\Psi) = 1$, следовательно, $I((\forall x)\Phi) = 1$ и $I((\forall x)\Psi) = 1$, значит, $I((\forall x)\Phi \wedge (\forall x)\Psi) = 1$. Если $I((\forall x)\Phi \wedge (\forall x)\Psi) = 1$, то $I((\forall x)\Phi) = 1$ и $I((\forall x)\Psi) = 1$. Значит, для любого a выполнено $(I)^x_a(\Phi) = 1$ и $(I)^x_a(\Psi) = 1$, поэтому $(I)^x_a(\Phi \wedge \Psi) = 1$, и $I((\forall x)(\Phi \wedge \Psi)) = 1$. 3) Если $I((\exists x)(\exists y)\Phi) = 1$, то существуют a и b такие, что $((I)^x_a)^y_b(\Phi) = 1$. Если $x = y$, то $((I)^x_a)^y_b = (I)^y_b$, $(I)^x_b(\Phi) = 1$. $I((\exists x)\Phi) = 1$ и $I((\exists y)(\exists x)\Phi) = 1$. Если $x \neq y$, то $((I)^x_a)^y_b = ((I)^y_b)^x_a$, поэтому $((I)^y_b)^x_a(\Phi) = 1$, $(I)^y_b((\exists x)\Phi) = 1$, $I((\exists y)(\exists x)\Phi) = 1$. 4) Аналогично 3). **131.** (а) Если $I((\exists x)(x \approx y \wedge \Phi)) = 1$, то $(I)^x_a(x \approx y \wedge \Phi) = 1$ для некоторого a. Из последнего получаем, что значение y тоже равно a, а формула Φ истинна. Следовательно, $I((\Phi)^x_y) = (I)^x_a(\Phi) = 1$. Обратно: из $I((\Phi)^x_y) = 1$ получаем $(I)^x_a(\Phi) = 1$, где a — значение y. Тогда $(I)^x_a(x \approx y \wedge \Phi) = 1$ и $I((\exists x)(x \approx y \wedge \Phi)) = 1$. (б) Если $I((\forall x)(x \approx y \to \Phi)) = 1$, то $(I)^x_b(x \approx y \wedge \Phi) = 1$ для всех b. В частности $(I)^x_a(x \approx y \to \Phi) = 1$, где

a — значение y. В последнем случае равенство истинно, следовательно, $I((\Phi)^x_y) = (I)^x_a(\Phi) = 1$. Обратно: из $I((\Phi)^x_y) = 1$ получаем $(I)^x_a(\Phi) = 1$, где a — значение y. Тогда $(I)^x_a(x \approx y \to \Phi) = 1$. Для всех $b \neq a$ в силу ложности равенства тоже получаем $(I)^x_b(x \approx y \wedge \Phi) = 1$. следовательно, $I((\forall x)(x \approx y \to \Phi)) = 1$. (в) $\Phi \Rightarrow (\exists x)\Phi$, применяем задачу 62. (г) $(\forall x)\Phi \Rightarrow \Phi$, применяем задачу 62. **132.** 1) Если $I((\exists x)(\Phi \wedge \Psi)) = 1$, то $(I)^x_a(\Phi \wedge \Psi) = 1$ для некоторого a. Следовательно, $(I)^x_a(\Phi) = 1$ и $(I)^x_a(\Psi) = 1$, поэтому $I((\exists x)\Phi) = 1$ и $I((\exists x)\Psi) = 1$, значит $I((\exists x)\Phi \wedge (\exists x)\Psi) = 1$. 2) Если $I((\forall x)\Phi \vee (\forall x)\Psi) = 1$, то $I((\forall x)\Phi) = 1$ или $I((\forall x)\Psi) = 1$. Рассмотрим случай $I((\forall x)\Phi) = 1$, тогда $(I)^x_a(\Phi) = 1$ для любого a, значит, $(I)^x_a(\Phi \vee \Psi) = 1$ для любого a и $I((\forall x)(\Phi \vee \Psi)) = 1$. Аналогично в случае $I((\forall x)\Psi) = 1$. 3) Если $I((\exists x)(\forall y)\Phi) = 1$, то $(I)^x_a((\forall y)\Phi) = 1$ для некоторого a. Последнее означает $((I)^x_a)^y_b(\Phi) = 1$ для всех b. Далее возможны два случая: $x = y$ или $x \neq y$. Если $x = y$, то $((I)^x_a)^y_b = (I)^y_b$. Получаем $(I)^y_b(\Phi) = 1$ для всех b, в частности, при $b = a$: $(I)^x_a(\Phi) = 1$. Значит, $I((\exists x)\Phi) = 1$ и $I((\forall y)(\exists x)\Phi) = 1$. Если $x \neq y$, то $((I)^x_a)^y_b = ((I)^y_b)^x_a$. Тогда $(I)^y_b((\exists x)\Phi) = 1$ для всех b и $I((\forall y)(\exists x)\Phi) = 1$. **133.** 2) $(\forall x)(E(x) \vee O(x))$ и $(\forall x)E(x) \vee (\forall x)O(x)$ в той же интерпретации; 3) $(\forall x)(\exists y)x < y$ и $(\exists y)(\forall x)x < y$, где $<$ означает обычный порядок на натуральных числах. **134.** (а) $(\exists z)(\forall x)(\exists y)(\forall u)(Q(x, y) \wedge Q(z, u))$; (б) $(\forall x)(\exists y)(\exists u)(\forall v)(\neg Q(x, y) \vee Q(v, u))$; (в) $(\exists u)(\forall v)(\exists w)(\exists y)(Q(u, v) \wedge \neg Q(u, z) \wedge (\neg Q(w, x) \vee Q(y, x)))$. **135.** (а) $(\exists x)(\Phi \to \Psi) \equiv (\exists x)(\neg\Phi \vee \Psi) \equiv (\exists x)\neg\Phi \vee (\exists x)\Psi \equiv \neg(\forall x)\Phi \vee (\exists x)\Psi \equiv (\forall x)\Phi \to (\exists x)\Psi$; (б) $(\forall x)(\Theta \to \Phi) \equiv (\forall x)(\neg\Theta \vee \Phi) \equiv \neg\Theta \vee (\forall x)\Phi \equiv \Theta \to (\forall x)\Phi$; (в) $(\forall x)(\Phi \to \Theta) \equiv (\forall x)(\neg\Phi \vee \Theta) \equiv (\forall x)\neg\Phi \vee \Theta \equiv \neg(\exists x)\Phi \vee \Theta \equiv (\exists x)\Phi \to \Theta$. **136.** $A =$ Комнаты[НомерКомнаты]\Оборудование[НомерКомнаты], (Сотрудники $\bowtie$ Комнаты $\bowtie$ A)[ФИО], $(\exists n)(\exists o)(\exists d)(\exists z)$(Сотрудники$(n, f, o, d, z) \wedge \neg(\exists e)(\exists nk)(\exists ob)$(Комнаты$(n, e, nk) \wedge$ Оборудование$(e, nk, ob)))$. **137.** $C = R[A_1, \ldots, A_n]$, $D = (C \times S) \setminus R$, $Q = C \setminus (D[A_1, \ldots, A_n])$. Если формулы Φ_R и Φ_S имеют свободные переменные $x_1, \ldots, x_n, y_1, \ldots, y_m$ и $y_1, \ldots, y_m$ соответственно, то $\Phi_Q = (\exists y_1)\ldots(\exists y_m)\Phi_R \wedge (\forall y_1)\ldots(\forall y_m)(\Phi_S \to \Phi_R)$. **138.** (а) `SELECT DISTINCT ФИО FROM Сотрудники WHERE Оклад > 5500`; (б) `SELECT DISTINCT Отдел FROM Сотрудники WHERE Оклад > 8000`; (в) `SELECT DISTINCT Должность, Оклад FROM Сотрудники`; (г) `SELECT`

`DISTINCT ФИО FROM Сотрудники, Комнаты WHERE Отдел = 'Торговый' AND Оклад >= 6000 AND Оклад <= 6500 AND Номер = НомерСотрудника AND Этаж != 3`; (д) `SELECT DISTINCT НомерКомнаты FROM Комнаты EXCEPT SELECT DISTINCT НомерКомнаты FROM Сотрудники, Комнаты WHERE Оклад >= 7500 AND Номер = НомерСотрудника`. **139.** Второе не выполнено. **140.** (а) $(\forall ns_1)(\forall nk_1)(\forall ns_2)(\forall nk_2)((\exists e)\text{Комнаты}(ns_1, e, nk_1) \wedge (\exists e)\text{Комнаты}(ns_2, e, nk_2) \to ns_1 \approx ns_2 \wedge nk_1 \approx nk_2)$; (б) $(\forall n)((\exists f)(\exists o)(\exists d)(\exists z)\text{Сотрудники}(n, f, o, d, z) \to (\exists e)(\exists nk)\text{Комнаты}(n, e, nk))$; (в) $(\forall nk)((\exists n)\text{Комнаты}(n, 2, nk) \to 10 < nk \wedge nk < 20) \wedge (\forall nk)((\exists n)\text{Комнаты}(n, 3, nk) \to 20 < nk)$. **141.** 2^{n^2}. **142.** Пути не могут иметь сколь угодно большую длину (иначе вершины начунт повторяться и будет цикл). Тогда вершина, пути из которой имеют наименьшую длину будет стоком, а наибольшую — истоком. **143.** Если v — сток (задача 142 на стр. 196), то он получает номер n. По индукционному предположению остальные вершины можно пронумеровать $v_1, \ldots, v_{n-1}$. **144.** Aw — множество вершин, из которых ведут рёбра в W, $w^T A$ — множество вершин, в которые ведут рёбра из W. **145.** Допустим, что $u \neq v$ и нет ребра (v, u). Пусть количество вершин равно n, а полустепень исхода v равна m, X — множество вершин, куда из v ведут рёбра, Y — откуда в u ведут рёбра. Тогда $|X| = m$, $|V \setminus (Y \cup \{u\})| \leqslant m$, $|Y| \geqslant n - m - 1$. Но $|X \cup Y| \leqslant n - 2$, так как ни X, ни Y не содержат ни v, ни u. Следовательно, X и Y пересекаются. **146.** Для двух вершин очевидно. Пусть для n вершин есть путь $(v_1, \ldots, v_n)$. Если есть только рёбра (v_i, v), то новый путь: $(v_1, \ldots, v_n, v)$. Если есть ребро (v, v_1), то новый путь: $(v, v_1, \ldots, v_n)$. Иначе есть ребро (v_1, v), пусть i — наименьший номер такой, что нет ребра (v_{i+1}, v). Тогда есть рёбра (v_i, v) и (v, v_{i+1}) и новый путь: $(v_1, \ldots, v_i, v, v_{i+1}, \ldots, v_n)$. **147.** Построить граф, где вершины — команды, а стрелки направлены от выигравшей к проигравшей. Тогда этот граф — турнир, далее применяем задачу 145. **148.** (а) Граф с петлями у каждой вершины, (б) полный граф, то есть со всеми возможными рёбрами. **149.** Кроме петель: взаимно достижимые: a и b, c и f. Из a и b достижимы c, d, f, из g: c, d, e, f, из e: c, d, f, из d: c, f. Базы: $\{a, g\}$, $\{b, g\}$. **150.** Кроме петель: (а) взаимно достижимые: d, e, f, Из c достижимы все, из d, e, f: a, b; (б) взаимно достижимые: c, d, e, Из a и b достижимы c, d, e, f, из c, d, e: f.

151. Антирефлексивность следует из определения строгой достижимости. Антисимметриночть: если K и K' достижимы друг из друга, то это — одна компонента. Транзитивность: по определению достижимости вершин. **152.** Кроме петель: из v_5 достижимы все, из v_1: v_2, v_3, v_4; из v_3: v_2; шесть новых рёбер, включая две петли. **153.** Если компонента сильной связности K содержит более одного элемента: u и v, $u \neq v$, то пути из u в v и из v в u образуют цикл. Если в графе есть цикл, то его вершины попадут в одну компоненту. **154.** Пусть $V = \{1, \ldots, n\}$, $E = \{(i, j) : i < j\}$. Если рёбер больше, и нет петель, то есть два ребра (u, v) и (v, u), образующие цикл. **155.** Все истоки должны попасть в любую базу, так как они ниоткуда больше не достижимы. Если для любой вершины v рассмотреть самый длинный путь через неё проходящий, то он должен начинаться с истока (задача 142). Следовательно, множество истоков является порождающим. **156.** $\mathfrak{G}_1$: $K_1 = \{a, d, g\}$, $K_2 = \{b, f\}$, $K_3 = \{c, e, h\}$, из K_1 строго достижимы K_2 и K_3, из K_2 — K_3. $\mathfrak{G}_2$: $K_1 = \{a\}$, $K_2 = \{c, d, f\}$, $K_3 = \{b, g, h\}$, $K_4 = \{e\}$, из K_1 строго достижимы K_2, K_3, K_4, из K_2 — K_3, K_4, из K_3 — K_4. **157.** Каждое ребро имеет два конца, сумма совпадает с общим количеством этих концов. **158.** С одной вершиной: один. С двумя вершинами: два. С тремя вершинами: четыре. С четырьмя вершинами: одиннадцать (по одному с 0, 1, 5 и 6 рёбрами, по два с 2 и 4 рёбрами, три с тремя рёбрами). **159.** Если после удаления ребра (a, b) граф остался связным, значит, есть простой пусть $(a, \ldots, b)$, не содержащий этого ребра. Тогда путь $(a, \ldots, b, a)$ будет простым циклом. Если (a, b) принадлежит простому циклу $(a, \ldots, b, a)$, то, удалив это ребро, мы не нарушим связности графа, так как все вершины этого цикла по-прежнему достижимы друг из друга при помощи рёбер простого пути $(a, \ldots, b)$. **160.** Индукция по n. Для $n = 1$ оба утверждения очевидны, в (б) будет петля. Пусть для n доказано, докажем для $n+1$. Если убрать першину v и k инцидентных рёбер, то образуется $\ell \leqslant k$ компонент связности, для каждой выполнено $|E_i| \geqslant |V_i| - 1$. Просуммировав и добавив удалённые, получим $|E| = \sum\limits_i |E_i| + k \geqslant (\sum\limits_i |V_i| - \ell) + k \geqslant \sum\limits_i |V_i| = |V| - 1$. Если $|E| \geqslant |V|$, то либо $|E_i| \geqslant |V_i|$ для какого-то i и цикл есть в V_i по индукционному предположению, либо $\ell < k$. В последнем случае две вершины соединённые с v соединены между ещё как-то, следовательно, есть цикл. **161.** Пусть V_a — люди знакомые с a, не включая a. Если в V_a

есть двое знакомых, то они вместе с a образуют искомую тройку. Если в V_a знакомых нет, но $|V_a| \geqslant 3$, то само V_a (или его подмножество) образует искомую тройку. Если $|V_a| \leqslant 2$, то рассмотрим $V \setminus V_a$ — множество не знакомых с a: $|V \setminus V_a| \geqslant 3$. Если в $V \setminus V_a$ есть двое незнакомых, то они вместе с a образуют искомую тройку. Иначе все в V_a знакомы и оно само или его подмножество является нужной тройкой. **162.** Если при разбиении такого ребра нет, то очевидно, граф не связен. Если граф не связен, то взять в качестве V_1 одну компоненту связности, а в качестве V_2 — остальные. **163.** Для каждой компоненты связности сумма степеней всех вершин чётна, задача 157. Следовательно, если нечётных вершин две, то они принадлежат одной компоненте, то есть связаны путём. **164.** Пусть пути $(u_1, \ldots, u_n)$ и $(v_1, \ldots, v_n)$ не имеют общих вершин. Так как граф связен, то вершины этих двух путей достижимы друг из друга. Пусть самый короткий путь соединяет u_i и v_j: $(u_i, w_1, \ldots, w_k, v_j)$. Если, например, $i \leqslant n/2$ и $j \leqslant n/2$, то путь $u_n, \ldots, u_i, w_1, \ldots, w_k, v_j, \ldots, v_n)$ будет иметь длину большую n. Аналогично строится более длинный путь для других случаев i и j. **165.** Индукция по $n = |V|$. При $n = k$ рёбер быть не может. Пусть для n доказано. Самая большая из компонент содержит не более $n - k + 1$ вершин. Следовательно, добавив новую вершину мы можем добавить не более $n-k+1$ рёбер. Получаем $(n-k)(n-k+1)/2+n-k+1 = (n+1-k)((n-k)/2+1) = (n+1-k)(n+1-k+1)/2$. **166.** Чётность следует из задачи 157. Если количество нечётных вершин равно $2n$, то добавив $n-1$ ребро можно оставить две нечётные вершины (граф станет полуэйлеровым), а добавив n рёбер убрать все нечётные вершины (граф станет эйлеровым). **167.** Триангуляция — плоский граф, у которого все грани, кроме внешней, треугольные: $e + 2 = f + v$, где количество вышек совпадёт с количеством вершин, а количество граней на единицу больше числа треугольников: $f = N + 1$. Далее $2e = 3N + K$, следовательно, $(3N+K)/2+2 = N+1+v$ и $v = (N+K)/2+1$. **168.** $2e = 3f$, $e+2 = f+v$, следовательно, $v = f/2 + 2$, $e = 3f/2$. Только для чётных f. Построение индукцией по f: на каждом шаге вместо одной из треугольных граней «приделываем» тетраэдр. **169.** При исключении изображение на плоскости можно оставить тем же, только убрать вершину. Чётность оставшихся вершин не меняется. При исключении вершины из цикла длины четыре

(два цвета) получится цикл длины три (три цвета). При исключении e в графе $\{(a,b),(b,c),(c,d),(d,a),(a,e),(e,c)\}$ появится гамильтонов цикл. **170.** Эйлеров только октаэдр, гамильтоновы все, хроматическое число у куба 2, у октаэдра и додекаэдра 3, у тетраэда и икосаэдра 4. **171.** Граф с рёбрами $\{(a,b),(b,c),(c,a)\}$ имеет оба цикла, $\{(a,b)\}$ ни одного, $\{(a,b),(b,c),(c,d),(d,a),(a,c)\}$ имеет только гамильтонов цикл, $\{(a,b),(a,c),(a,d),(e,b),(e,c),(e,d),(a,e)\}$ только эйлеров. **172.** Имеется две нечётных вершины: g и m, граф полуэйлеров, но не эйлеров. При удалении ребра (g,m) граф станет эйлеровым: эйлеров цикл $(h,g,e,f,n,k,f,b,k,c,b,m,c,a,k,h,a,m,h)$. **173.** Есть цикл длины три (b,f,m,b) — граф не двудольный. При удалении ребра (b,m) граф станет двудольным: доли $\{a,b,c,e,m\}$ и $\{f,g,h,k,n\}$. **174.** Всегда плоский. Эйлеров, только если нет рёбер. Полуэйлеров, только если граф является неветвящейся «линией». Гамильтонов, только в двух указанных выше случаях. Хроматическое число равно единице для дерева без рёбер, два — с рёбрами. **175.** 1) v — единственный исток: если $u \neq v$ и $(u_0 = v, u_1, \ldots, u_{n-1}, u_n = u)$ — кратчайший путь из v в u, то $n(u) > n(u_{n-1})$, поэтому ребро (u_{n-1}, u_n) ведёт из u_{n-1} в u, следовательно, u — не исток. 2) Если в $u \neq v$ ведёт два ребра (x,u) и (y,u), $x \neq y$, то $n(x) < n(u)$ и $n(y) < n(u)$, следовательно, существуют пути из v в x и y, не проходящие через u. Тогда в графе $\mathfrak{G}$ есть цикл $(\ldots, x, u, y, \ldots)$, что для дерева невозможно. 3) Если $(v = u_0, u_1, \ldots, u_{n-1}, u_n = u)$ — кратчайший путь из v в u в графе $\mathfrak{G}$, то ориентация рёбер будет именно такой: (u_i, u_{i+1}), поэтому в новом графе u достижима из v. **176.** Индукция по количеству вершин. Базис: $|V| = 2$ (иначе ребёр не может существовать). Тогда единственно ребро соединяет эти вершины, обе они являются висячими, следовательно, $|V'| = \varnothing$ и формула верна. Пусть для k вершин утверждение доказано. Рассмотрим дерево с $k+1$ вершиной. Среди них есть хотя бы одна висячая вершина u, иначе существовал бы цикл. Уберём u и инцидентное ребро (u,w), получим дерево $\mathfrak{T}'$ с k вершинами. По индукционному предположению количество висячих вершин в $\mathfrak{T}'$ равно $2 + \sum\limits_{v \in V_2'} (\deg v - 2)$. Если в дереве $\mathfrak{T}'$ вершина w висячая, то количество висячих вершин в $\mathfrak{T}$ и $\mathfrak{T}'$ совпадает. С другой стороны $\deg w = 2$, поэтому добавление слагаемого $\deg w - 2$ не меняет

суммы и формула остаётся верной. Если в дереве $\mathfrak{T}'$ вершина w невисячая, то количество висячих вершин в $\mathfrak{T}$ увеличивается на единицу. Поскольку $w \in V_2'$, то слагаемое $\deg w - 2$ при переходе от $\mathfrak{T}'$ к $\mathfrak{T}$ тоже возрастает на единицу и формула снова остаётся справедливой. **177.** Согласно задаче 160 в графе есть цикл, тогда можно выбросить любое ребро, которое в него входит, граф останется связным, и будет выполнено $|V| = |E| + 1$, следовательно, граф станет деревом. **178.** Если выбрать на каждом максимальном пути p вершину v_p, то из них можно построить множество X. Пусть X — минимальное по мощности из всех таких множеств. Если $|X| = 1$, то единственная его вершина принадлежит всем максимальным путям, что и требуется. Предположим, $|X| \geqslant 2$, $w, u \in X$, $w \neq u$. Если бы всякий максимальный путь, проходящий через w, проходил и через u, то w можно было бы выбросить и получить множество меньшей мощности, следовательно, есть максимальный путь p, проходящий через w, но не через u. Аналогично есть максимальный путь q, проходящий через u, но не через w. Согласно задаче 164 максимальные пути p и q имеют общую вершину v. Аналогично предыдущему показывается, что есть максимальный путь r, проходящий через w или через u, но не через v. Рассмотрим случай, когда r проходит через w, второй случай аналогичен. Максимальные пути q и r имеют общую вершину x. Но тогда в графе есть цикл $x \xrightarrow{q} v \xrightarrow{p} w \xrightarrow{r} x$, что невозможно. **179.** Индукция по построению графа $\mathfrak{T}$. Базис: если $\mathfrak{T}$ состоит из одной вершины и не имеет рёбер, то пункты 1)–3) определения 93 очевидно выполнены. Если для $\mathfrak{T}_i$, $i = 1, \ldots, k$, условия 1)–3) выполнены и дерево $\mathfrak{T}$ построено из $\mathfrak{T}_i$, то 1) r_0 — единственный исток, так как r_i, $i = 1, \ldots, k$, истоками быть перестали; 2) в r_i, $i = 1, \ldots, k$, ведёт по одному ребру, так как раньше в них никаких рёбер не вело, в остальные вершины кроме r_0 ведёт по одному ребру по индукционному предположению; 3) r_0 достижима сама из себя, если v принадлежит дереву $\mathfrak{T}_i$, то из r_0 можно попасть в r_i, а оттуда — в v по индукционному предположению. **180.** (а) 1) Единственный исток — v, так как достижимость остальных вершин означает, что в них должно входить хотя бы одно ребро; 2) в остальные вершины ведёт по одному ребру, так как это было выполнено в $\mathfrak{T}$; 3) по определению $\mathfrak{T}_v$. (б) Если бы $\mathfrak{T}_v$ и $\mathfrak{T}_u$ пересекались и имели общую вершину w, то w достижима из v и из u: есть пути $(u = u_0, \ldots, u_n = w)$ и

$(v = v_0, \ldots, v_m = w)$, Выберем среди всех таких вершин w ту, у которой сумма $n + m$ будет наименьшей, тогда $u_{n-1} \neq v_{m-1}$ и получается, что в w ведут два ребра, что для дерева невозможно. **181.** 1) r — единственный исток, так как у любого другого u найдётся меньший его $v < u$, поэтому среди них есть максимальный w и $(w, u) \in E$. 2) Для $u \neq r$ наличие ребра вида (v, u) показано в предыдущем пункте. Если $(v, u), (w, u) \in E$, то v и w — максимальные элементы меньшие u. Они несравнимы, так как иначе один из них не был бы максимальным. Но тогда согласно (б) u несравним с u, что невозможно. 3) Индукция по количеству k элементов строго меньших u. Если $k = 0$, то $u = r$, так как иначе $r < u$ и $k > 0$, тогда r достижим из r. Если для всех $\ell < k$ утверждение доказано и u имеет k меньших его элементов, то выберем v — максимальный из них. Тогда $(v, u) \in E$, а у v количество меньших элементов равно $\ell < k$. По индукционному предположению из r в v есть путь, значит, есть путь и в u. **182.** Рефлексивность: каждая вершина достижима из себя. Антисимметричность: если u достижима из v, а v — из u, то при $u \neq v$ был бы цикл, что невозможно. Транзитивность: если из u достижима v, а из v — w, то из u достижима w. Корень является наименьшим элементом, так как из него достижимы все вершины. Если вершины u и v недостижимы друг из друга, то большие их вершины образую деревья $\mathfrak{T}_u$ и $\mathfrak{T}_v$, которые не могут иметь общих вершин (задача 180), поэтому они тоже недостижимы друг из друга. **183.** В прямую сторону. Если бы был цикл $(u_0, u_1, \ldots, u_n = u_0)$, то он не мог бы содержать корня r. Так как из r есть путь в вершины этого цикла, то хотя бы в одну из этих вершин должно входить второе ребро, что невозможно. Второе и третье условие повторяют определение дерева. В обратную сторону. Пункты 1) и 2) определения повторяют условия задачи. 3) Так как в графе нет циклов, то пути не могут иметь сколь угодно большую длину. Рассмотрим самый длинный путь, ведущий в u: $(u_0, u_1, \ldots, u_n = u)$. Тогда u_0 является истоком (иначе путь можно сделать длиннее), следовательно, $u_0 = r$ и из r достижима u. **184.** Из задачи 176 получаем $26 = 2 + (5 - 2) + x(5 - 2) + 2x(4 - 2)$, где x — количество вершин с 4 сыновьями. Следовательно, $7x = 21$, количество вершин с четырьмя сыновьями равно трём, с тремя — шести. Общее количество вершин 37, рёбер — 36. **185.** Индукция по k — длине

$P = (v_1^{(s_1)}, \ldots, v_k^{(s_k)})$. Базис: $k = 0$,единственный вариант — пустой лес $\mathfrak{F} = ()$. Пусть для k утверждение доказано, а последовательность имеет длину $k+1$: $P = (v_0^{(s_0)}, v_1^{(s_1)}, \ldots, v_k^{(s_k)})$. По индукционному предположению по последовательности $P' = (v_1^{(s_1)}, \ldots, v_k^{(s_k)})$ длины k можно однозначно восстановить лес $\mathfrak{F}' = (\mathfrak{T}_1, \ldots, \mathfrak{T}_m)$. Если $s_0 = 0$, то в $\mathfrak{F}$ первое дерево $\mathfrak{T}_0$ состоит из единственной вершины, поэтому $\mathfrak{F} = (\mathfrak{T}_0, \mathfrak{T}_1, \ldots, \mathfrak{T}_m)$. Если $1 \leqslant s_1 \leqslant m$, то первые s_1 деревьев $\mathfrak{F}'$ войдут в новое дерево $\mathfrak{T}_0$ с корнем v_0, поэтому $\mathfrak{F} = (\mathfrak{T}_0, \mathfrak{T}_{s_1+1}, \ldots, \mathfrak{T}_m)$. Наконец, при $s_1 > m$ указанная последовательность не может быть префиксным обходом никакого леса, так как у v_0 должно быть s_1 сыновей, которых в лесу $\mathfrak{F}'$ нет. Для суффиксных обходов нужно аналогичным образом рассматривать последнюю вершину. **186.** Базис: дерево содержит одну вершину, которая является листом, вершин с двумя сыновьями нет, поэтому $0 = 1 - 1$. Шаг: пусть для деревьев $\mathfrak{T}_1$ и $\mathfrak{T}_2$ утверждение доказано: $n_1 = \ell_1 - 1$ и $n_2 = \ell_2 - 1$. Если $\mathfrak{T}$ включает $\mathfrak{T}_1$ и $\mathfrak{T}_2$, то $n = n_1 + n_2 + 1 = \ell_1 - 1 + \ell_2 - 1 + 1 = \ell_1$. Если $\mathfrak{T}$ включает в себя только $\mathfrak{T}_1$, то n и ℓ остаются без изменений. **187.** Количество листьев 2^h, количество вершин $2^{h+1} - 1$. **188.** ПРФ $= (\vee, \wedge, \vee, x, \neg, y, \neg, \to, z, \wedge, x, y, \oplus, \neg, z, y)$; СФФ $= (x, y, \neg, \vee, z, x, y, \wedge, \to, \neg, \wedge, z, \neg, y, \oplus, \vee)$; ИНФ $= (x, \vee, \neg, y, \wedge, \neg, z, \to, x, \wedge, y, \vee, \neg, z, \oplus, y)$. **189.** Сокращается 8 вершин. **190.** $\{(v_1, v_9), (v_1, v_3), (v_8, v_9), (v_3, v_2), (v_6, v_8), (v_6, v_7), (v_7, v_5), (v_5, v_4)\}$. **191.** Допустим, $c_i > d_i$. Выберем наименьшее из таких i. Так как на i-м шаге ребро d_i не было выбрано, то возможны две ситуации: либо ребро d_i выбрано раньше, либо рёбра $c_1, \ldots, c_{i-1}, d_i$ образуют цикл. В первом случае одно из рёбер $d_1, \ldots, d_{i-1}$ не выбрано до i-го шага. Следовательно, рёбра $c_1, \ldots, c_{i-1}, d_j$ образуют цикл. **192.** Пусть $\mathfrak{T}$ — минимальное остовное дерево $\mathfrak{G}$. Если оно не содержит e, то является и оствным деревом $\mathfrak{G}'$, очевидно, минимальным. Если $\mathfrak{T}$ содержит e, то при его удалении оно распадётся на две компоненты связности $\mathfrak{T}_1$ и $\mathfrak{T}_2$, причём цикл будет содержать вершины из обеих этих компонент. Выберем в цикле ребро e', отличное от e, соединяющее эти компоненты. Тогда связность восстановится, то есть получится дерево $\mathfrak{T}'$. Так как $w(e') \leqslant w(e)$, то вес нового дерева не может увеличиться. Теперь $\mathfrak{T}'$ — минимальное оствное дерево для $\mathfrak{G}$ и $\mathfrak{G}'$. **193.** Если $\mathfrak{T}_i = (V_i, T_i)$ — дерево полученное после i-го шага, то докажем, что $\mathfrak{T}_i$ можно расширить до минимального

остовного дерева. Базис: $i = 0$ — очевидно. Шаг: пусть для i утверждение доказано и $\mathfrak{T}_i$ расширяется до минимального остовного дерева $\mathfrak{T} = (V, T)$. Допустим на $(i+1)$-м шаге выбрано ребро (u, v), где $u \in V_i$, $v \notin V_i$. Если $(u, v) \in T$, то условие вновь выполняется. В противном случае $(u, v) \notin T$, но тогда есть путь $(u = u_0, \ldots, u_n = v)$ в $\mathfrak{T}$. Выберем наименьшее i такое, что $(u_i, u_{i+1}) \notin T$. Согласно пункту (б) алгоритма $w(u_i, u_{i+1}) \geqslant w(u, v)$. Рассмотрим $\mathfrak{T}' = (V, E \setminus \{(u_i, u_{i+1})\} \cup \{(u, v)\})$. Это снова остовное дерево, вес не мог увеличиться, поэтому оно минимальное и $\mathfrak{T}_{i+1}$ — поддерево $\mathfrak{T}'$. **194.** Пусть $Num[u] < Num[v]$, то есть вызов Поиск (u) происходит раньше вызова Поиск (v). Докажем, что u — предок v. Рассмотрим вызов Поиск (u). Так как $v \in L_u$ и ребро (u, v) не является прямым, то вызов Поиск (v) произошёл до того, как в цикле 20–25 была рассмотрена вершина v. Следовательно, вершина v была обработана в ходе выполнения этого цикла, это и означает, что v принадлежит поддереву с корнем u, то есть является потомком u. **195.** Кроме уже описанных строк для Up добавить после строки 23: **Если** $Up[w] = Num[w]$ **то Печать** (v, w). **196.** $Num = (1, 2, 8, 6, 4, 9, 10, 7, 3, 5)$, первое обратное ребро (v_{10}, v_9), цикл (v_9, v_5, v_{10}, v_9), $Up = (1, 2, 7, 6, 3, 7, 7, 7, 3, 3)$, мосты: (v_1, v_2), (v_1, v_4), (v_1, v_8), (v_2, v_9). **197.** Вставить: после строки 11 **Печать** («{»), после строки 12 **Печать** («}»), после строки 16 **Печать** (v). **198.** Применимы алгоритмы Ярника-Прима-Дейкстры и алгоритм поиска в глубину, если начинать с вершины r. Алгоритм Крускала непосредственно неприменим, так как не всякий ориентированный граф без циклов является деревом. **199.** $\{(a, c, 17), (c, e, 50), (c, b, 123), (b, d, 25), (d, f, 5)\}$. **200.** Длина пути P_2 может быть отрицательной, и путь из a в w_k через u может оказаться короче и пункт (а) после очередного шага не будет выполнен. Например, в графе $\{(a, c, 2), (a, b, 3), (b, c, -2\}$. **201.** Если она не является предпоследней: $(a, \ldots, x, w_k, z, \ldots, u)$, где $x, z \in R_k$, то есть не проходящий через w_k путь из a в z не большей длины. **202.** От одного до пяти (с учётом инициализации). Пример: $\{(a_i, a_j, (j-i)^2)\}$ для всех $1 \leqslant i < j \leqslant 6$. **203.** Пункты определения: 1) в a рёбер не входит, а во все остальные вершины — входят; 2) если в v входит больше одного ребра, то есть два различных кратчайших пути, проходящих через v, тогда и до v есть не менее двух кратчайших путей; 3) по условию задачи. **204.** При изменении значения любой переменной значение

всей функции меняется. Если бы какая-то элементарная конъюнкция не содержали переменную x_i и её отрицание, то она осталась бы истинной при изменении значения этой переменной. Следовательно, все импликанты должны иметь длину n. Поскольку функция принимает значение 1 в точности на 2^{n-1} наборах, то в ДНФ этих импликантов именно столько и должно быть. **205.** Индукция по длине программы Π. Если присваиваний нет, то вычисляются только тождественные функции равные x_i, которые и реализуются формулами вида $\Phi = x_i$. Пусть $\Pi = (\Pi', y \leftarrow f(y_1, \ldots, y_k))$. Тогда программа Π' в переменных $y_1, \ldots, y_k$ вычисляет функции $f_1, \ldots, f_k$ соответственно, а по индукционному предположению можно построить формулы $\Phi_1, \ldots, \Phi_k$, их реализующие. Но тогда функция f, вычисляемая программой Π в переменной y, реализуется формулой $f(\Phi_1, \ldots, \Phi_k)$. **206.** Так как базис $\mathcal{B}$ позволяет получить константу 0, то существует формула Φ_0 с переменной x_1 в этом базисе, значение которой всегда равно нулю. Согласно предложению 84 существует линейная программа Π_0, вычисляющая константу 0 в переменной y, причём в Π_0 каждая переменная не используется до того, как ей присвоено значение. Будем считать, что Π и Π_0 не имеют общих переменных. Преобразуем Π в Π', для каждой невходной переменной x все её вхождения до первого присваивания ей заменим на y: если $\Pi = (\Pi_1, y \leftarrow e, \Pi_2)$, где Π_1 не содержит присваиваний переменной x, то $\Pi' = ((\Pi_1)^x_y, y \leftarrow (e)^x_y, \Pi_2)$. Пусть $\Pi'' = \Pi_0\Pi'$. Очевидно, что Π'' вычисляет в переменных те же функции, что и Π, но теперь каждое использование невходной переменной происходит только после явного присваивания ей значения. Далее применяем индукцию по длине Π''. Если Π'' пусто, то вычисляются только значения входных переменных и схема состоит из одних входов. Если $\Pi'' = (\Pi''', z \leftarrow f(z_1, \ldots, z_k))$, то по индукционному предположению для Π''' уже построена схема $\mathfrak{S}_{\Pi'''}$, в вершинах v_{z_i} которой вычисляются те же функции, что и в переменных z_i программы Π'''. Тогда, чтобы построить $\mathfrak{S}_{\Pi''}$, нужно к $\mathfrak{S}_{\Pi'''}$ добавить новую вершину v_z, помеченную f, и рёбра из всех v_{z_i} в v_z. **208.** Построить для f ДНФ, использовать одну переменную для отрицаний входов, вторую — для вычисления элементарных конъюнкций, третью — для вычисления ДНФ. Если воспользоваться базисом $\{1, \wedge, \oplus\}$ и многочленами Жегалкина, то нужны только две переменных: одна для вычисления

отдельных слагаемых, вторая — суммы. **209.** Для n-разрядных двоичных чисел выполнено $d = d + 2^n$ (так как в числе 2^n младшие n разрядов нулевые). Следовательно, $d = a - b = a + (2^n - b) = a + 1 + (2^n - 1 - b)$. Число $2^n - 1 - b$ получается просто инвертированием всех двоичных разрядов b: $2^n - 1 - b = \bar{b}$, следовательно, $d = a + \bar{b} + 1$. Для нахождения первой суммы достаточно использовать n инверторов $\neg$ и сумматор SUM_n. Для второй нужен инвертор $\neg$ (для младшего разряда) и $n - 1$ сумматор по модулю два $\mathfrak{S}_\oplus$ для остальных (суммируем текущий разряд с переносом из предыдущего). **210.** 3 для $\mathfrak{S}_\oplus$, $3n - 3$ для $\mathfrak{S}_{\mathrm{odd}}$, $6n$ для SUM_1, $3n$ для SUM_n при $n \geqslant 2$. **211.** Использовать схемы $\mathfrak{S}_\oplus$ для отдельного сравнения младших и старших разрядов. Далее построить отрицания результатов и их конъюнкцию. **212.** Строим $\mathfrak{S}_n$ индукцией по n. Если $n = 1$, то выход «$=_1$» — это отрицание $\mathfrak{S}_\oplus$, а выходы «$>_1$» и «$<_1$» — конъюнкция одного числа с отрицанием другого. Для $n + 1$: используем схему $\mathfrak{S}_n$ для сравнения старших n разрядов и схему $\mathfrak{S}_1$ для сравнения младших. Выход «$=_{n+1}$» является конъюнкцией «$=_n$» и «$=_1$». Выход «$>_{n+1}$» — дизъюнкция «$>_n$» и конъюнкции «$=_n$» и «$>_1$». Выход «$<_{n+1}$» — дизъюнкция «$<_n$» и конъюнкции «$=_n$» и «$<_1$». **213.** Реализовать Следующую схему: $c_0 = a_0 \wedge b_0$, $e = a_1 \wedge b_1$, $c_3 = e \wedge c_0$, $c_2 = e \wedge \neg c_0$, $d = a_1 \vee b_1$, $f = a_0 \vee b_0$, $c_1 = d \wedge f \wedge \neg c_3$. **214.** Пусть входы x_1, x_2, x_3, y — голоса «за» трёх обычных членов и председателя соответственно. Реализовать схему для формулы $(y \wedge ((x_1 \vee x_2) \vee x_3)) \vee ((x_1 \wedge x_2) \wedge x_3)$. **215.** (а) Реализовать схему для вычисления $\neg x_1 \vee x_2 \vee \neg x_3$. (б) Построить $\mathfrak{S}_\oplus$ с отрицанием для x_2 и x_3. (в) Построить схемы для первых четырёх значений ($\neg x_1 \wedge x_2$), для вторых четырёх значений ($\mathfrak{S}_\oplus$ с отрицанием для x_2 и x_3 и конъюнкция с x_1), соединить результаты дизъюнкцией. **216.** Базис: перед первым шагом ($i = n + 1$), тогда подфункция $f_{\sigma_1,\ldots,\sigma_n}$ не зависит от переменных, то есть является константой. Так как переменных нет, то УБДР для констант не может содержать ничего кроме стоков. Констант всего две и УБДР имеет ровно два стока — по одному для каждой из констант. Допустим, для $i + 1$ утверждение доказано и докажем его для i. Рассмотрим функцию $f_{\sigma_1,\ldots,\sigma_j}(x_{j+1}, \ldots, x_n)$ для произвольного $j \geqslant i - 1$. Если $j \geqslant i$, то для него утверждение выполнено по индукционному предположению, так как вершины для таких x_{j+1} на i-м шаге не меняются.

Рассмотрим случай $j = i - 1$, то есть функции $f_{\sigma_1,\ldots,\sigma_{i-1}}(x_i,\ldots,x_n)$. Если зафиксировать первый аргумент такой функции то мы получим функции $f_{\sigma_1,\ldots,\sigma_{i-1},0}(x_{i+1},\ldots,x_n)$ и $f_{\sigma_1,\ldots,\sigma_{i-1},1}(x_{i+1},\ldots,x_n)$. Для них можно применить индукционное предположение: существует по одной вершине (например, w_0 и w_1 соответственно), реализующей каждую из этих функций. Тогда для каждой вершины v, помеченной x_i, реализующей функцию $f_{\sigma_1,\ldots,\sigma_{i-1}}(x_i,\ldots,x_n)$, её 0-сын — это w_0, а 1-сын — это w_1. Если $f_{\sigma_1,\ldots,\sigma_{i-1}}(x_i,\ldots,x_n)$ не зависит от x_i, то функции $f_{\sigma_1,\ldots,\sigma_{i-1},0}(x_{i+1},\ldots,x_n)$ и $f_{\sigma_1,\ldots,\sigma_{i-1},1}(x_{i+1},\ldots,x_n)$ совпадают, а по индукционному предположению, w_0 и w_1 тоже совпадают. Тогда в строках 11–14 алгоритма вершина w будет удалена, поэтому вершина $w_0 = w_1$ останется единственной, реализующей функцию $f_{\sigma_1,\ldots,\sigma_{i-1}}(x_i,\ldots,x_n)$. Пусть теперь функции $f_{\sigma_1,\ldots,\sigma_{i-1},0}(x_{i+1},\ldots,x_n)$ и $f_{\sigma_1,\ldots,\sigma_{i-1},1}(x_{i+1},\ldots,x_n)$ различны. Если в исходной диаграмме существовало несколько вершин w, реализующих функцию $f_{\sigma_1,\ldots,\sigma_{i-1}}(x_i,\ldots,x_n)$, то все они имели бы одних и тех же 0-сына и 1-сына. Но тогда в строках 22–25 алгоритма из всех этих вершин останется только одна. **217.** Любая УБДР, реализующая функцию f, должна в своих поддиаграммах реализовывать и всевозможные подфункции $f_{\sigma_1,\ldots,\sigma_j,0}(x_j,\ldots,x_n)$. Различные подфункции не могут реализовываться в одной и той же вершине. Поэтому количество вершин не меньше чем количество попарно различных подфункций. По лемме 92, в построенной УБДР для каждой подфункции есть одна вершина, следовательно такая УБДР содержит наименьшее возможное количество вершин. По той же причине она является единственно возможной. **218.** Полное БДР содержит 2^{i-1} вершин для переменной x_i. При слиянии останется 2 вершины для x_n, 4 для x_{n-1} и т. д., 2^{i+1} для x_{n-i}. Всего $1 + 2 \times 2 + 2 \times 2^2 + \cdots \leqslant 2^{n/2+2} - 1$. **219.** Сложение по модулю два, как на рис. 68 (z заменить на x). Остальные функции — проверяем значение переменной x, если есть необходимость (0 для дизъюнкции, 1 для остальных), проверяем значение y. **220.** (а) 10 вершин: на каждом шаге вычисляем конъюнкцию x_{2i-1} и x_{2i}, а затем — сложение по модулю два с предыдущим результатом. Всего по одной вершине для x_1 и x_2, по две для остальных. (б) Всего 17 вершин: для x_6 — две вершины (функции x_6 и $x_6 \oplus 1$), для x_4 — четыре (функции x_4, $x_4 \oplus 1$, $x_4 \oplus x_6$, $x_4 \oplus x_6 \oplus 1$), для x_2 — четыре (функции x_2, $x_2 \oplus x_4$, $x_2 \oplus x_6$,

$x_2 \oplus x_4 \oplus x_6$). Вершины x_1, x_3, x_5 образуют полное дерево из 7 вершин. **221.** (а) УДБР можно наглядно изобразить в виде параллелограмма, где ребро 0 идёт вертикально вниз, а ребро 1 — вправо вниз по диагонали. Тогда i-я вертикальная линия (начиная с нулевой) соответствует полученным уже i единицам. Как только в $(k-1)$-й вертикальной линии получаем единицу — ответ 1. Если в i-й вертикальной линии дошли до $x_{n-k+i+1}$ и получили ноль — ответ 0. (б) Нет. (в) Когда мы доходим до вершины с переменной x_i нам нужно знать, какое количество предыдущих переменных имеет значение 1. Таким образом, для x_1 нужна одна вершина, для x_2 — две, для x_3 — три, ..., для x_k — k вершин. Наоборот, значение переменной x_n требуется анализировать только если среди предыдущих переменных в точности $k-1$ единица (одна вершина), значение x_{n-1} — если $k-1$ или $k-2$ единицы (две вершины), ..., значение x_{n-k+1} — от 0 до $k-1$ единиц. Таким образом, «параллелограмм» из (а) является минимальной УБДР с количеством внутренних вершин $(n-k+1)(k+1)$. **222.** (а) Сначала проверяем совпадение старших разрядов чисел (3 вершины), затем — младших (ещё 3 вершины). (б) Сначала учитываем голос председателя y, Если он равен единице по очереди проверяем остальные и если хотя бы один равен 1, то ответ 1. Если голос председателя равен 0, по очереди проверяем остальные и если хотя бы один равен 0, то ответ 0. Последние вершины для x_3 в обеих ветках можно совместить, всего окажется 6 вершин. **223.** Обратная индукция по глубине вершин. Для стоков 0 и 1 берём $\Pi_0 = y \leftarrow 0$ и $\Pi_1 = y \leftarrow 1$ соответственно. Если вершина w с переменной x имеет 0-сына w_0 и 1-сына w_1, для которых уже построены программы Π_{w_0} и Π_{w_1}, то $\Pi_w =$ **Если** x **то** Π_{w_1} **Иначе** Π_{w_0}. **224.** Входной алфавит: $1, 2, 5$ — приём монет, b — нажатие на кнопку. Выходной алфавит: r и g — красная и зелёная лампочки, c — кофе, $1, 2, 5$ — выдача монет. $q_0, 1 \to q_1, r$; $q_0, 2 \to q_2, r$; $q_0, 5 \to q_5, r$; $q_0, b \to q_0, r$; $q_1, 1 \to q_2, r$; $q_1, 2 \to q_3, r$; $q_1, 5 \to q_6, r$; $q_1, b \to q_1, r$; $q_2, 1 \to q_3, r$; $q_2, 2 \to q_4, r$; $q_2, 5 \to q_7, r$; $q_2, b \to q_2, r$; $q_3, 1 \to q_4, r$; $q_3, 2 \to q_5, r$; $q_3, 5 \to q_8, g$; $q_3, b \to q_3, r$; $q_4, 1 \to q_5, r$; $q_4, 2 \to q_6, r$; $q_4, 5 \to q_9, g$; $q_4, b \to q_4, r$; $q_5, 1 \to q_6, r$; $q_5, 2 \to q_7, r$; $q_5, 5 \to q_{10}, g$; $q_5, b \to q_5, r$; $q_6, 1 \to q_7, r$; $q_6, 2 \to q_8, g$; $q_6, 5 \to q_{11}, g$; $q_6, b \to q_6, r$; $q_7, 1 \to q_8, g$; $q_7, 2 \to q_9, g$; $q_7, 5 \to q_{12}, g$; $q_7, b \to q_7, r$; $q_8, 1 \to q_8, g1$; $q_8, 2 \to q_8, g2$; $q_8, 5 \to q_8, g5$; $q_8, b \to q_0, cr$; $q_9, 1 \to q_9, g1$;

$q_9, 2 \to q_9, g2$; $q_9, 5 \to q_9, g5$; $q_9, b \to q_0, c1r$; $q_{10}, 1 \to q_{10}, g1$; $q_{10}, 2 \to q_{10}, g2$; $q_{10}, 5 \to q_{10}, g5$; $q_{10}, b \to q_0, c2r$; $q_{11}, 1 \to q_{11}, g1$; $q_{11}, 2 \to q_{11}, g2$; $q_{11}, 5 \to q_{11}, g5$; $q_{11}, b \to q_0, c12r$; $q_{12}, 1 \to q_{12}, g1$; $q_{12}, 2 \to q_{12}, g2$; $q_{12}, 5 \to q_{12}, g5$; $q_{12}, b \to q_0, c22r$. **225.** Входной алфавит: $1, 2$ — нажатие на кнопки. Выходной алфавит: n, h, m, s — режим, H, M, S — увеличение соответствующего числа. $q_0, 1 \to q_1, h$; $q_0, 2 \to q_0, n$; $q_1, 1 \to q_2, m$; $q_1, 2 \to q_1, hH$; $q_2, 1 \to q_3, s$; $q_2, 2 \to q_2, mM$; $q_3, 1 \to q_0, n$; $q_3, 2 \to q_3, sS$. **226.** $q_0, 0 \to q_0, 0$; $q_0, 1 \to q_1, 1$; $q_0, \Lambda \to q_0, \varepsilon$; $q_1, 0 \to q_0, 1$; $q_1, 1 \to q_2, 0$; $q_1, \Lambda \to q_0, 1$; $q_2, 0 \to q_1, 0$; $q_2, 1 \to q_2, 1$; $q_2, \Lambda \to q_0, 01$. **227.** $q_0, 0 \to q_0, 0$; $q_0, 1 \to q_1, 0$; $q_0, 2 \to q_2, 0$; $q_0, 3 \to q_0, 1$; $q_0, 4 \to q_1, 1$; $q_0, 5 \to q_2, 1$; $q_0, 6 \to q_0, 2$; $q_0, 7 \to q_1, 2$; $q_0, 8 \to q_2, 2$; $q_0, 9 \to q_0, 3$; $q_1, 0 \to q_1, 3$; $q_1, 1 \to q_2, 3$; $q_1, 2 \to q_0, 4$; $q_1, 3 \to q_1, 4$; $q_1, 4 \to q_2, 4$; $q_1, 5 \to q_0, 5$; $q_1, 6 \to q_1, 5$; $q_1, 7 \to q_2, 5$; $q_1, 8 \to q_0, 6$; $q_1, 9 \to q_1, 6$; $q_2, 0 \to q_2, 6$; $q_2, 1 \to q_0, 7$; $q_2, 2 \to q_1, 7$; $q_2, 3 \to q_2, 7$; $q_2, 4 \to q_0, 8$; $q_2, 5 \to q_1, 8$; $q_2, 6 \to q_2, 8$; $q_2, 7 \to q_0, 9$; $q_2, 8 \to q_1, 9$; $q_2, 9 \to q_2, 9$. **228.** Индукция по k. При $k = 0$ единственный возможный случай $\tau = \sigma$. Пусть для k доказано. Если $\sigma \vdash_{\mathfrak{M}}^{k+1} \tau$, то по определению существует ρ такое, что $\sigma \vdash_{\mathfrak{M}}^{k} \rho$ и $\rho \vdash_{\mathfrak{M}} \tau$. По индукционному предположению такое ρ единственно, пусть $\rho = (q, u, v)$. Так как $\rho \vdash_{\mathfrak{M}} \tau$, то $u = au'$, в программе $\mathfrak{M}$ есть команда $q, a \to p, y$, и $\tau = (p, u', vy)$. Так как команда такого вида единственна, получаем единственность τ. **229.** Базис: $w = \varepsilon$, тогда $(q, wx) = (q, x)$. Поскольку входное слово не изменилось, то количество шагов равно нулю, следовательно, $p = q$, путь (q) несёт пустое слово. Наоборот: если путь несёт пустое слово, он не содержит ни одного ребра, следовательно, $p = q$. Предположим, что всех слов, длина которых меньше w, утверждение доказано и $|w| > 0$. Тогда $w = w'a$. По индукционному предположению $(q, w'ax) \vdash_{\mathfrak{M}}^{*} (r, ax)$ тогда и только тогда, когда когда в $\mathfrak{D}_{\mathfrak{M}}$ есть путь из q в w, несущий w'. Если $(q, wx) \vdash_{\mathfrak{M}}^{*} (p, x)$, то $(r, ax) \vdash_{\mathfrak{M}} (p, x)$, что означает наличие команды $r, a \to p$ в программе $\mathfrak{M}$. Следовательно, в $\mathfrak{D}_{\mathfrak{M}}$ есть ребро (r, p), помеченное a. Объединяя путь из q в r, несущий w', с этим ребром, получим путь из q в p, несущий w. Наоборот, если в $\mathfrak{D}_{\mathfrak{M}}$ есть путь из q в p, несущий w, то есть путь из q в r, несущий w', и ребро (r, p), помеченное a. Последнее означает команду $q, a \to p$, поэтому $(r, ax) \vdash_{\mathfrak{M}} (p, x)$, и $(q, wx) \vdash_{\mathfrak{M}}^{*} (p, x)$. Пусть автомат принимает слово w. Тогда $(q_0, w) \vdash_{\mathfrak{M}}^{*} (p, \varepsilon)$,

где $p \in F$. Из утверждения получаем, что есть путь из q_0 в p, несущий w. Наоборот, пусть есть путь из q_0 в какое-то $p \in F$. Из утверждения получаем $(q_0, w) \vdash^*_{\mathfrak{M}} (p, \varepsilon)$, то есть $\mathfrak{M}$ принимает w. **230.** (а) $q_0, a \to q_1$; $q_0, b \to q_1$; $q_1, a \to q_2$; $q_1, b \to q_2$; $q_2, a \to q_3$; $q_2, b \to q_3$; $q_3, a \to q_4$; $q_3, b \to q_5$; $q_4, a \to q_0$; $q_4, b \to q_0$; $F = \{q_0\}$. (б) $q_0, a \to q_a$; $q_0, b \to q_b$; $q_a, a \to q_{aa}$; $q_a, b \to q_b$; $q_{aa}, a \to q_{aa}$; $q_{aa}, b \to q_e$; $q_b, a \to q_a$; $q_b, b \to q_{bb}$; $q_{bb}, a \to q_e$; $q_{bb}, b \to q_{bb}$; $q_e, a \to q_e$; $q_e, b \to q_e$; $F = \{q_0, q_a, q_b, q_{aa}, q_{bb}\}$. (в) $q_0, a \to q_1$; $q_0, b \to q_2$; $q_1, a \to q_0$; $q_1, b \to q_3$; $q_2, a \to q_3$; $q_2, b \to q_0$; $q_3, a \to q_2$; $q_3, b \to q_1$; $F = \{q_2\}$. (г) $q_0, a \to q_1$; $q_0, b \to q_3$; $q_1, a \to q_2$; $q_1, b \to q_4$; $q_2, a \to q_0$; $q_2, b \to q_5$; $q_3, a \to q_4$; $q_3, b \to q_0$; $q_4, a \to q_5$; $q_4, b \to q_1$; $q_5, a \to q_6$; $q_5, b \to q_2$; $F = \{q_0\}$. **231.** $q_0, \begin{bmatrix}\begin{bmatrix}0\\0\\0\end{bmatrix}\end{bmatrix} \to q_0$; $q_0, \begin{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}\end{bmatrix} \to q_2$; $q_0, \begin{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}\end{bmatrix} \to q_2$; $q_0, \begin{bmatrix}\begin{bmatrix}0\\1\\1\end{bmatrix}\end{bmatrix} \to q_0$; $q_0, \begin{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\end{bmatrix} \to q_2$; $q_0, \begin{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix}\end{bmatrix} \to q_0$; $q_0, \begin{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}\end{bmatrix} \to q_1$; $q_0, \begin{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}\end{bmatrix} \to q_2$; $q_1, \begin{bmatrix}\begin{bmatrix}0\\0\\0\end{bmatrix}\end{bmatrix} \to q_2$; $q_1, \begin{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}\end{bmatrix} \to q_0$; $q_1, \begin{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}\end{bmatrix} \to q_1$; $q_1, \begin{bmatrix}\begin{bmatrix}0\\1\\1\end{bmatrix}\end{bmatrix} \to q_2$; $q_1, \begin{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\end{bmatrix} \to q_1$; $q_1, \begin{bmatrix}\begin{bmatrix}1\\0\\1\end{bmatrix}\end{bmatrix} \to q_2$; $q_1, \begin{bmatrix}\begin{bmatrix}1\\1\\0\end{bmatrix}\end{bmatrix} \to q_2$; $q_1, \begin{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}\end{bmatrix} \to q_1$; $q_2, \begin{bmatrix}[*]\end{bmatrix} \to q_2$; $*$ — любые варианты; $F = \{q_0\}$. **232.** Базис: $w = \varepsilon$. Тогда из $(q_1, \varepsilon) \vdash^*_{\mathfrak{M}_1} (p_1, \varepsilon)$, $(q_2, \varepsilon) \vdash^*_{\mathfrak{M}_2} (p_2, \varepsilon)$, $((q_1, q_2), \varepsilon) \vdash^*_{\mathfrak{N}} ((r_1, r_2), \varepsilon)$, получаем $p_1 = q_1 = r_1$, $p_2 = q_2 = r_2$, что и требуется. Пусть для слов длины меньше w утверждение доказано и $w = w'a$. Пусть $(q_1, w'a) \vdash^*_{\mathfrak{M}_1} \vdash^*_{\mathfrak{M}_1} (p_1, \varepsilon)$, $(q_2, w'a) \vdash^*_{\mathfrak{M}_2} (p_2, \varepsilon)$, $((q_1, q_2), w'a) \vdash^*_{\mathfrak{N}} ((r_1, r_2), \varepsilon)$. По определению это означает, что существуют состояния p'_1, p'_2, (r'_1, r'_2), для которых $(q_1, w') \vdash^*_{\mathfrak{M}_1} (p'_1, \varepsilon)$, $(q_2, w') \vdash^*_{\mathfrak{M}_2} (p'_2, \varepsilon)$, $((q_1, q_2), w') \vdash^*_{\mathfrak{N}} ((r'_1, r'_2), \varepsilon)$, $(p'_1, a) \vdash^*_{\mathfrak{M}_1} \vdash^*_{\mathfrak{M}_1} (p_1, \varepsilon)$, $(p'_2, a) \vdash^*_{\mathfrak{M}_2} (p_2, \varepsilon)$, $((r'_1, r'_2), a) \vdash^*_{\mathfrak{N}} ((r_1, r_2), \varepsilon)$. По индукционному предположению $p'_1 = r'_1$ и $p'_2 = r'_2$. Кроме того, в $\mathfrak{N}$ должна быть команда $(r'_1, r'_2), a \to (r_1, r_2)$. Следовательно, в $\mathfrak{M}_1$ и $\mathfrak{M}_2$ есть команды $r'_1, a \to r_1$ и $r'_2, a \to r_2$ соответственно. Кроме того в $\mathfrak{M}_1$ и $\mathfrak{M}_2$ есть команды $p'_1, a \to p_1$ и $p'_2, a \to p_2$ соответственно. В силу детерминированности и $p'_1 = r'_1$ и $p'_2 = r'_2$ делаем вывод, что $r_1 = p_1$ и $p_2 = r_2$. **233.** Из любой вершины слово aba ведёт в заключительное состояние $\{q_0, q_1, q_3\}$, следовательно, слово оканчивающееся на aba, будет принято независимо от начала. **234.** (а) После первого этапа: $P_1 = \{q_0, a \to q_1; q_0, b \to q_2; q_1, b \to q_2; q_2, b \to q_2\}$; $F_1 = \{q_0, q_1, q_2\}$.

Фактически автомат уже детерминированный, добавляем состояние $\varnothing$ для отсутствующих команд: $P_2 = \{q_0, a \to q_1; q_0, b \to q_2; q_1, a \to \varnothing; q_1, b \to q_2; q_2, a \to \varnothing; q_2, b \to q_2; \varnothing, a \to \varnothing; \varnothing, b \to \varnothing\}$; $F_2 = \{q_0, q_1, q_2\}$. (б) После первого этапа: $P_1 = \{q_0, a \to q_1; q_0, a \to q_2; q_1, b \to q_0; q_1, b \to q_2; q_2, b \to q_0\}$; $F = \{q_1, q_2\}$. После детерминизации: $P_2 = \{q_0, a \to q_{12}; q_0, b \to \varnothing; q_{12}, a \to \varnothing; q_{12}, b \to q_{02}; q_{02}, a \to q_{12}; q_{02}, b \to q_0; \varnothing, a \to \varnothing; \varnothing, b \to \varnothing\}$; $F = \{q_{12}, q_{02}\}$. **235.** (а) $\{a, ab, abb, aa, aba, abba, abbb, aab, abab, abbab, aba, abba, abbba\}$; (б) $\{a, aa, aba, abba, b, abbb, abb, aabb, ababb, abbabb, ba, aba, abbba\}$; (в) $\{\varepsilon, a, b, ab, aba, aa, ba, abaa, bb, abb, abab, aab, bab, abaab, bba, abba, ababa\}$. **236.** (в). **237.** 1) следует из коммутативности объединения множеств; 2) следует из ассоциативности объединения множеств; 3) следует из ассоциативности конкатенации слов; 4) если $w \in L((r^*)^*)$, то $w = v_1 \dots v_n$, где $v_i \in L(r^*)$. В свою очередь $v_i = u_{i1} \dots u_{im_i}$, где $u_{ij} \in L(r)$. Тогда $w = u_{11} \dots u_{nm_n}$, что и означает $w \in L(r^*)$. Обратное включение непосредственно следует из $L \subseteq L^*$ для любого языка L; 5) если $w \in L((r + p)q)$, то $w = uv$, где $u \in L(r + p)$, $v \in L(q)$. В свою очередь $u \in L(r + p)$ означает, что $u \in L(r)$ или $u \in L(p)$. В первом случае $w = uv \in L(r)L(q) = L(rq) \subseteq L(rq) \cup L(pq) = L(rq + pq)$, во втором — $w = uv \in L(p)L(q) = L(pq) \subseteq L(rq) \cup L(pq) = L(rq + pq)$. Обратно, если $w \in L(rq + pq)$, то $w \in L(rq)$ или $w \in L(pq)$. В первом случае получаем $w = uv$, где $u \in L(r)$, $v \in L(q)$. Тогда $u \in L(r) \cup L(p) = L(r + p)$. Следовательно, $w = uv \in L(r + p)L(q) = L((r + p)q)$. Аналогично во втором случае. **238.** Если слово удовлетворяет регулярному выражению, то оно не может содержать трёх нулей подряд: после двух будет либо единица, либо конец слова. Если в слове нет трёх нулей подряд, то можно разбить его на фрагменты, каждый из которых состоит из нулей и заканчивается (кроме, может быть, последнего) единицей. Тогда каждый такой фрагмент имеет вид 1, 01 или 001, а последний — 0, 00 или пустой. Следовательно, такое слово описывается регулярным выражением $(1 + 01 + 001)^*(0 + 00 + \varepsilon)$. **239.** Будем для удобства с помощью p, q и r обозначать и произвольные слова, удовлетворяющие выражениям p, q и $p + q$ соответственно. Скобки в словах расставлены для удосбства, сами они в слова не входят. (а) Если $w \in L(p^*(p + q)^*)$, то $w = (p \dots p)(r \dots r) = (p \dots p)(p \dots pqp \dots pq \dots p \dots qp \dots p) =$

$p\dots pp\dots p(qp\dots p)(qp\dots p)\dots(qp\dots p))$, что удовлетворяет выражению $p^*(qp^*)^*$. Если $w \in L(p^*(qp^*)^*)$, то $w = (p\dots p)(qp\dots p)(qp\dots p)\dots(qp\dots p) = p\dots p(qp\dots p)(qp\dots p)\dots(qp\dots p)$, следовательно, $w \in L((p+qp^*)^*)$. Если $w \in L((p+qp^*)^*)$, то $w \in L((p+q)^*)$, так как последний язык содержит все возможные комбинации p и q. Если $w \in L((p+q)^*)$, то $w \in L(p^*(p+q)^*)$, так как в качестве p^* можно взять пустое слово. (б) Если $w \in L(p(qp)^*)$, то $w = p(qp)(qp)\dots(qp) = (pq)(pq)(p\dots q)p$, что соответствует $(pq)^*p$. Аналогично в обратную сторону. (в) Так как итерация содержит пустое слово, то используя (б) получаем $L((p^*q^*)^*) \subseteq L((p^*q^*)^*p^*) \subseteq L(p^*(q^*p^*)^*) \subseteq L(q^*p^*(q^*p^*)^*) \subseteq L((q^*p^*)^*)$. Аналогично в обратную сторону. (г) $(pq)^+(q^*p^*+q^*) \equiv (pq)^*(pq)(q^*p^*+q^*) \equiv (pq)^*p(qq^*p^*+qq^*) \equiv (pq)^*p(q^+p^*+q^+) \equiv (pq)^*p(q^+p^*+q^+) \equiv (pq)^*pq^+(p^*+\varepsilon) \equiv (pq)^*pq^+p^*$. **240.** (а) разобъём слово на двухсимвольные фрагменты и последний символ, тогда количество 00 и 11 может быть произвольным, а количество 01 и 10 чётно, если последний символ 0, и нечётно если последний символ 1. В первом случае получаем $(00+11+(01+10)(00+11)^*(01+10))^*0$, во втором — $(00+11+(01+10)(00+11)^*(01+10))^*(01+10)(00+11)^*1$. Общее выражение: $(00+11+(01+10)(00+11)^*(01+10))^*(0+(01+10)(00+11)^*1)$; (б) $(0+1)^*(001+110)(0+1)^*$; (в) $(0+1)^*00(0+1)^*$; (г) 1 может стоять после 0 только в самом конце слова: $1^*0^*(\varepsilon+1)$. **241.** (а) Слова, которые оканчиваются нулём, до него стоит сколько-то (может быть, ноль) единиц, а до них — сколько-то (может быть, ноль) нулей; (б) слова, которые начинаются и оканчиваются одним нулём, а между ними стоит сколько-то (может быть, ноль) единиц; (в) все слова, начинающиеся и оканчивающиеся нулями, в том числе и просто 0; (г) слова с чётным числом и нулей, и единиц. **242.** (а) $\varepsilon + 000^*$; (б) $(0+1)(00)^*$; (в) 0^*1. **243.** Индукция по n: для языка $\{w^{m_1}\}$ длина регулярного выражения r_1, построенного простой конкатенацией всех букв, равна $4km_1 - 3 = 4km_1 + 4 \times 1 - 7$ (km_1 букв и по $km_1 - 1$ амперсандов, открывающих и закрывающих скобок). Пусть для языка $\{w^{m_2-m_1}, \dots, w^{m_n-m_1}\}$ построено регулярное выражение r' длины $4k(m_n - m_1) + 4(n-1) - 7$. Тогда для языка $\{w^{m_1}, \dots, w^{m_n}\}$ таким выражением будет $r = (r_1 \,\&\, (\varepsilon + r'))$. Длина r равна $4km_1 - 3 + 4k(m_n - m_1) + 4(n-1) - 7 + 7 = 4km_n + 4n - 7$.

244. $\left(\begin{bmatrix}0\\0\\0\end{bmatrix} + \begin{bmatrix}0\\1\\1\end{bmatrix} + \begin{bmatrix}1\\0\\1\end{bmatrix} + \begin{bmatrix}1\\1\\0\end{bmatrix}\left(\begin{bmatrix}1\\1\\1\end{bmatrix}\begin{bmatrix}0\\1\\0\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}\right)^*\begin{bmatrix}0\\0\\1\end{bmatrix}\right)^*$. **245.** Индукция по сложности r. Для базисных регулярных выражений все утверждения очевидны (рис. 89). Пусть для выражений r_1 и r_2 утверждение доказано. (а) При использовании утверждения 105 условие (а) сохраняется, так как новые рёбра исходят только из старых заключительных состояний, а из нового заключительного состояния рёбер не исходит вовсе. При построении автомата для объединения (рис. 86) из новой вершины (не заключительной!) исходят два ребра, из остальных — столько же, сколько и раньше. При построении автомата для конкатенации (рис. 87) новое ребро выходит из старого заключительного состояния, которое перестаёт быть таковым. При построении автомата для итерации (рис. 88) по одному новому ребру выходит из старого заключительного состояния, которое перестаёт быть таковым, и из нового начального состояния, которое и становится новым заключительным. Следовательно, на любом шаге новый автомат удовлетворяет пункту (а). (б) При использовании утверждения 105 в каждом из автоматов появляется одно новое состояние. При построении автоматов для объединения, конкатенации и итерации, появляется ещё не более одного состояния. Таким образом, $|Q| \leqslant |Q_1| + 1 + |Q_2| + 1 + 1 = |Q_1| + |Q_2| + 3$. По индукционному предположению $|Q_i| \leqslant 3|r_i|$, следовательно, $|Q| \leqslant |Q_1| + |Q_2| + 3 \leqslant 3|r_1| + 3|r_2| + 3 = 3(|r_1| + |r_2| + 1) \leqslant 3|r|$, что и требуется. (в) Если для обоих исходных автоматов было использовано утверждение 105, то из их заключительных состояний не будет переходов, следовательно, такие заключительные состояния можно слить. Поэтому мы получим только два новых состояния. При построении автоматов для конкатенации и итерации общее количество новых состояний и так не превосходит двух. Поэтому получаем $|Q| \leqslant |Q_1| + |Q_2| + 2 \leqslant 2|r_1| + 2|r_2| + 2 = 2(|r_1| + |r_2| + 1) \leqslant 2|r|$. **246.** Базис: $m = 0$, тогда $w = \varepsilon$, следовательно, путь (q_0) несёт w и автомат $\mathfrak{N}$ принимает w. Шаг: пусть для m утверждение доказано. Рассмотрим слово $w = w_1 \dots w_m w_{m+1} \in (L(\mathfrak{M}_1))^*$. Так как $w_1 \dots w_m \in (L(\mathfrak{M}_1))^*$, то по индукционному предположению в $\mathfrak{D}_{\mathfrak{M}_1}$ есть путь P из q_0 в q_0, несущий это слово. Тогда P можно продолжить так: сначала по ε переходим в

q_0^1, далее, так как $w_{m+1} \in L(\mathfrak{M}_1)$, то есть путь R из q_0^1 в q_f^1, несущий w_{m+1}, и, наконец, с помощью ε возвращаемся в q_0. Тогда описанное нами продолжение пути P несёт слово w_{m+1}, а весь путь (P, R, q_0) будет нести $w_1 \dots, w_m w_{m+1} = w$. Значит, $\mathfrak{N}$ принимает слово w. **247.** После первого этапа состояния p_6 и q_6 исчезнут, заключительными будут q_0, p_3, q_7, q_8. После детерминизации $\{q_0\}, 0 \to \{p_2, s_2, q_7, q_8\}$; $\{q_0\}, 1 \to \{p_3\}$; $\{p_3\}, 0 \to \{p_2, s_2, q_7, q_8\}$; $\{p_3\}, 1 \to \{p_3\}$; $\{p_2, s_2, q_7, q_8\}, 0 \to \{s_3, q_8\}$; $\{p_2, s_2, q_7, q_8\}, 1 \to \{p_3\}$; $\{s_3, q_8\}, 0 \to \varnothing$; $\{s_3, q_8\}, 1 \to \{p_3\}$; $\varnothing, 0 \to \varnothing$; $\varnothing, 1 \to \varnothing$; заключительные — все, кроме $\varnothing$. **248.** $(aa)^* + a(aa)^*(ba(aa)^*)^*b(aa)^* + a(aa)^* + a(aa)^*(ba(aa)^*)^*ba(aa)^* \equiv (aa)^*(aa)^* + (aa)^*a(ba(aa)^*)^*b(aa)^* + a(aa)^*(aa)^* + (aa)^*a(ba(aa)^*)^*ba(aa)^* \equiv (aa)^*(aa)^* + (aa)^*a(ba(aa)^*)^*b(aa)^* + (aa)^*a(aa)^* + (aa)^*a(ba(aa)^*)^*ba(aa)^* \equiv (aa)^*(\varepsilon + a(ba(aa)^*)^*b + a + a(ba(aa)^*)^*ba)(aa)^* \equiv (aa)^*(\varepsilon + a + a(ba(aa)^*)^*b(\varepsilon + a))(aa)^* \equiv (aa)^*((\varepsilon + a(ba(aa)^*)^*b)(\varepsilon + a))(aa)^* \equiv (aa)^*((\varepsilon + a(b(aa)^*a)^*b)(\varepsilon + a))(aa)^* \equiv (aa)^*(\varepsilon + (ab(aa)^*)^*ab)(\varepsilon + a)(aa)^* \equiv (aa)^*(\varepsilon + (ab(aa)^*)^*ab)((aa)^* + a(aa)^*) \equiv (aa)^*(\varepsilon + (ab(aa)^*)^*ab)((aa)^* + (aa)^*a) \equiv (aa)^*(\varepsilon + (ab(aa)^*)^*ab)(aa)^*(\varepsilon + a) \equiv (aa)^*(\varepsilon + (ab(aa)^*)^*ab)(aa)^*(\varepsilon + a) \equiv (aa)^*((aa)^* + (ab(aa)^*)^*ab(aa)^*)(\varepsilon + a) \equiv (aa)^*((aa)^* + (ab(aa)^*)^+)(\varepsilon + a) \equiv ((aa)^*(aa)^* + (aa)^*(ab(aa)^*)^+)(\varepsilon + a) \equiv ((aa)^* + (aa)^*(ab(aa)^*)^+)(\varepsilon + a) \equiv (aa)^*(\varepsilon + (ab(aa)^*)^+)(\varepsilon + a) \equiv (aa)^*(ab(aa)^*)^*(\varepsilon + a) \equiv (aa + ab)^*(\varepsilon + a)$. **249.** Новый автомат получается добавлением команд вида $q, a \to q$ для всех состояний и всех $a \in \Sigma \setminus \Omega$. **250.** Новый автомат получается из старого $\mathfrak{M} = (Q, \Sigma, P, q_0, F)$ обращением всех рёбер в противоположную сторону, добавлением нового начального состояния q_0', команд $q_0', \varepsilon \to p$ для всех $p \in F$. Единственным заключительным состоянием будет q_0. **251.** (а) Объявить в новом автомате заключительными все состояния, из которых были достижимы заключительные состояния старого автомата. (б) Создать новое начальное состояние q_0' и добавить команды $q_0', \varepsilon \to p$ для всех p, достижимых из старого начального состояния. (в) Сначала применить (а), потом (б). (г) Оставить заключительными только те состояния, из которых заключительные не являются строго достижимыми (то есть из них за 1 или более шагов нельзя попасть в заключительное). **252.** С автоматом $\mathfrak{M}$, распознающем язык L сделать такое преобразование. Вместо каждой команды $c = q, a_i \to p$ создать уникальную копию $\mathfrak{M}_i^c = (Q_i^c, \Delta, P_i^c, q_{0i}^c, F_i^c)$

автомата $\mathfrak{M}_i$ и добавить команды $q, \varepsilon \to q_{0i}^c$ и $q_f, \varepsilon \to p$ для всех $q_f \in F_i^c$. **253.** Точка обозначена нулём, тире — единицей. Рёбра: $(\varepsilon, 1)$, $(10, 00)$, $(1, 00)$, $(1, \varepsilon)$, $(1, 000)$, $(\varepsilon, \varepsilon)$, $(1, 10)$, $(10, 10)$, $(101, \varepsilon)$, $(10, \varepsilon)$, $(\varepsilon, 10)$, $(00, 10)$, две петли $(\varepsilon, \varepsilon)$ нетривиальные: их метки $100, 0$ и $0, 01, 0$. **254.** $(0, 1)$, $(1, 00)$, $(10, 0)$, $(\varepsilon, 10)$, $(0, 0)$, $(0, 110)$, $(011, 0)$, $(1, 1)$, $(\varepsilon, 00)$, $(0, \varepsilon)$, $(010, 0)$, $(1, \varepsilon)$, $(10, 1)$. Цикл: $\varepsilon \xrightarrow{01} 10 \xrightarrow{\varepsilon} 1 \xrightarrow{01} \varepsilon$. Слово 0110101 можно декодировать как afa и как cf. **255.** Чтобы из вершины ε исходило ребро, не являющееся тривиальной петлёй, нужно, чтобы одно кодовое слово было префиксом другого. **256.** (а) $\varphi(\text{К}) = 01$; $\varphi(\text{А}) = 00$; $\varphi(\text{Р}) = 110$; $\varphi(\text{Т}) = 111$; $\varphi(\text{И}) = 100$; $\varphi(\text{Ц}) = 101$; длина кода 24, при равномерном кодировании для каждого символа нужно три знака: $2^2 < 6 \leqslant 2^3$, длина будет равна 30, что на 6 символов длиннее. (б) $\varphi(\text{П}) = 01$; $\varphi(\text{А}) = 000$; $\varphi(\text{Р}) = 001$; $\varphi(\text{Л}) = 11$; $\varphi(\text{Е}) = 101$; $\varphi(\text{И}) = 1001$; $\varphi(\text{Д}) = 1000$; длина кода 38, при равномерном кодировании для каждого символа нужно три знака $2^2 < 7 \leqslant 2^3$, длина будет равна 42, что на 4 символа длиннее. (в) $\varphi(\text{Т}) = 01$; $\varphi(\text{Е}) = 100$; $\varphi(\text{Л}) = 1110$; $\varphi(\text{А}) = 00$; $\varphi(\text{П}) = 101$; $\varphi(\text{Р}) = 110$; $\varphi(\text{У}) = 1111$; длина кода 38, при равномерном кодировании для каждого символа нужно три знака $2^2 < 7 \leqslant 2^3$, длина будет равна 42, что на 4 символа длиннее. (г) $\varphi(\text{И}) = 11$; $\varphi(\text{Н}) = 101$; $\varphi(\text{Д}) = 100$; $\varphi(\text{В}) = 0010$; $\varphi(\text{У}) = 0011$; $\varphi(\text{А}) = 0000$; $\varphi(\text{Л}) = 0001$; $\varphi(\text{Ь}) = 011$; $\varphi(\text{О}) = 0100$; $\varphi(\text{С}) = 01010$; $\varphi(\text{Т}) = 01011$; длина кода 54, при равномерном кодировании для каждого символа нужно четыре знака $2^3 < 11 \leqslant 2^4$, длина будет равна 64, что на 10 символов длиннее. (д) $\varphi(\text{П}) = 001$; $\varphi(\text{Е}) = 01$; $\varphi(\text{Р}) = 11$; $\varphi(\text{А}) = 0000$; $\varphi(\text{С}) = 0001$; $\varphi(\text{Д}) = 1000$; $\varphi(\text{Л}) = 1001$; $\varphi(\text{Н}) = 1011$; $\varphi(\text{И}) = 1010$; длина кода 48, при равномерном кодировании для каждого символа нужно четыре знака $2^3 < 9 \leqslant 2^4$, длина будет равна 68, что на 20 символов длиннее. Оптимальный код не единственен, но длина минимального кода постоянна. **257.** $q_0, 0 \to q_0, \text{О}$; $q_0, 1 \to q_1, ;$ $q_0, \Lambda \to q_0, ;$ $q_1, 0 \to q_{10}, ;$ $q_1, 1 \to q_0, \text{В}$; $q_1, \Lambda \to q_0, \text{Е}$; $q_{10}, 0 \to q_{100}, ;$ $q_{10}, 1 \to q_0, \text{Д}$; $q_{10}, \Lambda \to q, \text{Е}$; $q_{100}, 0 \to q_0, \text{Т}$; $q_{100}, 1 \to q_0, \text{Р}$; $q_{100}, \Lambda \to q, \text{Е}$; с помощью Е обозначен сигнал ошибки. **258.** Единственным новым кодовым словом в φ' является u. Остальные кодовые слова были и в ψ'. Если бы ψ'_c было префиксом u, то оно было бы и префиксом $\psi'_a = u0$, что невозможно. Если u является префиксом ψ'_c, то $\psi'_c = uxw$, где $x \in \{0, 1\}$. Но тогда ψ'_a или ψ'_b были бы префиксом

или совпадали с ψ'_c, что также невозможно. **259.** Предположим, что язык L автоматный, тогда возьмём для него константу k из леммы о разрастании. (а) Возьмём слово $w = a^{k+3}b^k \in L$, разобьём на три части: $\alpha = a^{k+3}$, $u = b^k$, $\beta = \varepsilon$. Тогда существует разбиение $b^k = xyz$, для которого $1 \leqslant |y| \leqslant k$. Поскольку x, y и z состоят только из букв b, то $x = b^p$, $y = b^q$, $z = b^{k-p-q}$ для каких-то p и q, причём $1 \leqslant q \leqslant k$. Согласно лемме о разрастании $w_0 = \alpha xy^0z\beta = a^{k+3}b^pb^{k-p-q}\varepsilon = a^{k+3}b^{k-q} \in L$. Тогда должно выполняться $k + 3 = k - q + 3$, откуда $q = 0$, что противоречит $q \geqslant 1$. Противоречие доказывает, что язык неавтоматный. (б) Возьмём слово $w = a^kcb^{3k+1} \in L$, разобьём на три части: $\alpha = \varepsilon$, $u = a^k$, $\beta = cb^{3k+1}$. Тогда существует разбиение $a^k = xyz$, для которого $1 \leqslant |y| \leqslant k$. Поскольку x, y и z состоят только из букв a, то $x = a^p$, $y = a^q$, $z = a^{k-p-q}$ для каких-то p и q, причём $1 \leqslant q \leqslant k$. Согласно лемме о разрастании $w_2 = \alpha xy^2z\beta = \varepsilon a^pa^{2q}a^{k-p-q}cb^{3k+1} = a^{k+q}cb^{3k+1} \in L$. Тогда должно выполняться $3k + 1 > 3(k + q) = 3k + 3q$, откуда $3q < 1$, что противоречит $q \geqslant 1$. Противоречие доказывает, что язык неавтоматный. (в) Возьмём слово $w = a^2b^kacab^ka^2 \in L$, разобьём на три части: $\alpha = a^2$, $u = b^k$, $\beta = acab^ka^2$. Тогда существует разбиение $b^k = xyz$, для которого $1 \leqslant |y| \leqslant k$. Поскольку x, y и z состоят только из букв b, то $x = b^p$, $y = b^q$, $z = b^{k-p-q}$ для каких-то p и q, причём $1 \leqslant q \leqslant k$. Согласно лемме о разрастании $w_0 = \alpha xy^0z\beta = a^2b^pb^{k-p-q}acab^ka^2 = a^2b^{k-q}acab^ka^2 \in L$. Тогда должно выполняться $(a^2b^{k-q}a)^{-1} = ab^ka^2$, что неверно, так как количество букв b в словах различается. Противоречие доказывает, что язык неавтоматный. (г) Возьмём слово $w = a^{2^k} \in L$, разобьём на три части: $\alpha = \varepsilon$, $u = a^{2^k}$, $\beta = \varepsilon$. Тогда существует разбиение $b^k = xyz$, для которого $1 \leqslant |y| \leqslant k$. Поскольку x, y и z состоят только из букв b, то $x = a^p$, $y = a^q$, $z = a^{2^k-p-q}$ для каких-то p и q, причём $1 \leqslant q \leqslant k$. Согласно лемме о разрастании $w_2 = \alpha xy^2z\beta = \varepsilon a^pa^{2q}a^{2^k-p-q}\varepsilon = a^{2^k+q} \in L$. Так как $q \geqslant 1$, то $2^k + q > 2^k$. С другой стороны, так как $a^{2^k+q} \in L$, то $2^k + q$ должно быть степенью двойки. Наименьшая степень двойки, превосходящая 2^k, — это 2^{k+1}. Следовательно, $2^k + q \geqslant 2^{k+1}$, то есть $q \geqslant 2^k > k$, что противоречит $q \leqslant k$. Противоречие доказывает, что язык неавтоматный. (д) Возьмём слово $a^kc^k \in L$, далее как в предложении 131. **260.** Из определения легко по индукции получается, что в λ-выражении

колиства открывающих и закрывающих скобок совпадают, так как они всегда добавляются парами. Допустим, что множество λ-выражений является автоматным языком, пусть тогда k — константа из леммы о разрастании для него. Возьмём такое λ-выражение: $(x(x(\dots(xx)\dots)))$, в котором k открывающих и k закрывающих скобок. Разобьём его на три части: $\alpha = (x(x(\dots(xx$, $u =)^k$, $\beta = \varepsilon$. Далее как в предложении 131. **261.** Допустим, что такой язык автоматный, а k — константа из леммы о разрастании для него. Рассмотрим такое слово: $\begin{bmatrix}0\\0\\0\end{bmatrix}^{k+1}\begin{bmatrix}1\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\end{bmatrix}^{k}\begin{bmatrix}0\\0\\1\end{bmatrix} \in L$, здесь в двоичном виде записано произведение $2^{k+1} \times 2^{k+1} = 2^{2k+2}$. Разобьём его на следующие части: $\alpha = \begin{bmatrix}0\\0\\0\end{bmatrix}^{k+1}\begin{bmatrix}1\\1\\0\end{bmatrix}$, $u = \begin{bmatrix}0\\0\\0\end{bmatrix}^{k}$, $\beta = \begin{bmatrix}0\\0\\1\end{bmatrix}$. Согласно лемме о разрастании u может быть представлено в виде xyz, где $x = \begin{bmatrix}0\\0\\0\end{bmatrix}^{p}$, $y = \begin{bmatrix}0\\0\\0\end{bmatrix}^{q}$, $z = \begin{bmatrix}0\\0\\0\end{bmatrix}^{k-p-q}$, причём $1 \leqslant q \leqslant k$. По лемме получаем $w_0 = \alpha xz\beta = \begin{bmatrix}0\\0\\0\end{bmatrix}^{k+1}\begin{bmatrix}1\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\end{bmatrix}^{p}\begin{bmatrix}0\\0\\0\end{bmatrix}^{k-p-q}\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix}^{k+1}\begin{bmatrix}1\\1\\0\end{bmatrix}\begin{bmatrix}0\\0\\0\end{bmatrix}^{k-q}\begin{bmatrix}0\\0\\1\end{bmatrix} \in L$. Но теперь записано произведение $2^{k+1} \times 2^{k+1} = 2^{2k+2-q} \neq 2^{2k+2}$, то есть ошибочный результат. Тогда слово не должно принадлежать L. Противоречие показывает, что предположение неверно, язык неавтоматный. **262.** Если бы язык L был автоматным, то и его дополнение $\bar{L}$ тоже было бы автоматным языком. Неавтоматность $\bar{L}$ доказывается так же как в предложении 131, достаточно взять слово $a^k b^k \in \bar{L}$, где k — константа из леммы о разрастании. **264.** (а) mC_m^n, $m2^m$. (б) Частный случай (в). (в) $x_1 \leftarrow 2^m + \sum\limits_{i=1}^{m} x_i \times 2^{i-1}$, $x_2 \leftarrow x_1 + 2^m$, $\dots$, $x_m \leftarrow x_{m-1} + 2^m$. Если входные переменные $i_1, \dots, i_n$, а выходная x_k, то $f(1, \dots, 1) = a$, где число a имеет такую двоичную запись: младшие m разрядов содержат единицы на тех разрядах, которые соответствуют номерам входных переменных, а старшие — номер выходной переменной. Следовательно, все такие функции f различны. **265.** (а) mA_m^n, $mA_m = me_m\, m! \approx me\, m!$ (см. задачу 34). (в) $x_1 \leftarrow 2^{m^2} + \sum\limits_{i=1}^{m} 2^{(m+1)(i-1)+x_i}$, $x_2 \leftarrow x_1 + 2^{m^2}$, $\dots$, $x_m \leftarrow x_{m-1} + 2^{m^2}$. Если входные переменные $i_1, \dots, i_n$, а выходная x_k, то $f(1, 2, \dots, n) = a$,

где число a имеет такую двоичную запись: единица в младших $m+1$ разрядах определяет позицию x_1 среди входных переменных (0 — x_1 не входная, i — x_1 находится на i-м месте среди входных), следующие $m+1$ — позицию x_2 и т. д. Разряды, начиная с m^2 — номер выходной переменной. Следовательно, все такие функции f различны. **266.** При наличии одной переменной $x \leftarrow (x < x)$ эквивалентно $x \leftarrow 0$, а $x \leftarrow x$ ничего не меняет, поэтому такие операторы можно исключить. Ветвление имеет смысл только как самый первый оператор, поскольку после $x \leftarrow s(x)$ или $x \leftarrow 0$ его результат заранее известен. Таким образом, имеются следующие три варианта программ: (а) увеличение x на $m \leqslant n$: $\Phi(x) = x + m$; (б) присваивание $x \leftarrow 0$ и увеличение на $m < n$: $\Phi(x) = m$; (в) зацикливание. Всего $2n + 2$ варианта. Кроме того, возможна проверка в начале, равен ли аргумент нулю и дальше — комбинация (а), (б) и (в) в зависимости от результата сравнения. Для всюду останавливающихся программ если $x = 0$, то (а) и (б) эквивалентны, следовательно, результат такой программы определяется исключительно случаем $x \neq 0$ и эти комбинации можно не рассматривать. Следовательно, добавляется ещё три группы: ноль (а) и не ноль (в) — n вариантов, ноль (в) и не ноль (а) — n вариантов, ноль (в) и не ноль (б) — $n - 1$ вариант. Таким образом, всего получается $5n + 1$ вариант. **267.** После оператора $\alpha\ x \leftarrow s(x)\ \beta$ ветвление β **Если** x **то** γ **Иначе** δ предопределено, поэтому можно сразу написать $\alpha\ x \leftarrow s(x)\ \gamma$. Если после оператора $\alpha\ x \leftarrow s(x)\ \beta$ идёт $\beta\ x \leftarrow d(x)\ \gamma$, то суммарно ничего не меняется, поэтому вместо перехода к α можно сразу переходить к γ. Исходя из вышесказанного делаем вывод, что операторы $x \leftarrow s(x)$ могут встречаться только в самом конце, а начало состоит из декрементов и ветвлений. После ветки **Иначе** декременты и ветвления смысла уже не имеют, поэтому можно считать, что после **Иначе** всегда начинаются инкременты. Таким образом, программа устроена так: сначала идут несколько ветвлений и декрементов, затем (возможно) начинается цикл, состоящий из декрементов и ветвлений, который продолжается, пока x не станет равным нулю. Следовательно, вычисляемая функция имеет такой вид: на некотором начальном отрезке она задана таблично: $f(0) = a_0, \ldots, f(k) = a_k$, а для $x > k$ либо $f(x) = x + c$ для некоторой константы c, либо $f(x)$ задана таблично в зависимости от $x \bmod c$ для некоторой константы

с. **268.** Если программа содержит ℓ меток, то она имеет $\ell 2^n$ всевозможных конфигураций, так как переменные могут принимать только два значения. Следовательно, если вычисление имеет длину больше $\ell 2^n$, то оно уже содержит две одинаковые конфигурации, поэтому программа зацикливается. Значит, чтобы найти Φ, нужно перебрать все возможные наборы аргументов (их не более 2^n), для каждого из них построить вычисление длины не больше $\ell 2^n + 1$, если до этого момента программа остановится, то результат будет значением функции Φ, а иначе результат не определён. **270.** Последовательно перебираем все натуральные числа $i = 0, 1, 2, \ldots$, для каждого вычисляем $f(i)$, как только будет $f(i) = x$, то $i = f^{-1}(x)$. **272.** (а) $! = \Pr[1, s(\mathrm{id}_1^2) \times \mathrm{id}_2^2]$; (б) индукция по n. Для $n = 1$: $\min^{(1)} = \mathrm{id}_1^1$, для $n + 1$: $\min^{(} n + 1) = \mathrm{if}(\mathrm{less}(\mathrm{id}_1^{n+1}, \mathrm{id}_{n+1}^{n+1}), \min^{(n)}(\mathrm{id}_1^{n+1}, \ldots, \mathrm{id}_n^{n+1}), \min^{(n)}(\mathrm{id}_2^{n+1}, \ldots, \mathrm{id}_{n+1}^{n+1}))$; (в) аналогично (б); (г) $\lfloor \mathrm{id}_1^2 - \mathrm{id}_2^2 \rfloor + \lfloor \mathrm{id}_2^2 - \mathrm{id}_1^2 \rfloor$. **273.** Суперпозиция: $g = f(\mathrm{id}_{i_1}^n, \ldots, \mathrm{id}_{i_n}^n)$. **274.** $f = \mathrm{if}(\mathrm{less}(\mathrm{id}_n^n, b), a, g(\mathrm{id}_1^n, \ldots, \mathrm{id}_{n-1}^n, \lfloor \mathrm{id}_n^n - b \rfloor))$. **275.** $A(1, y) = y + 2$, $A(2, y) = 2y + 3$. **276.** (а) $\mathrm{rt} = \mu\, \mathrm{leq}(\mathrm{pow}(s(\mathrm{id}_3^3), \mathrm{id}_1^3), \mathrm{id}_2^3)$; (б) $\log = \mu\, \mathrm{leq}(\mathrm{pow}(\mathrm{id}_1^3, s(\mathrm{id}_3^3)), \mathrm{id}_2^3)$; (в) $p = \mathrm{less}(1, p) \times \mathrm{eq}(2, \sum\limits_{x < \mathrm{id}_1^1} (s(x) \mid \mathrm{id}_1^1))$; (г) $\mathrm{pn} = \Pr[2, \mu(\mathrm{leq}(\mathrm{id}_3^3, \mathrm{id}_2^3) + \mathrm{not}(p(\mathrm{id}_3^3)))]$; (д) $\tau = \sum\limits_{x < \mathrm{id}_1^1} (s(x) \mid \mathrm{id}_1^1)$; (е) $\mathrm{digit} = [\mathrm{id}_1^3 \,/\, \mathrm{pow}(\mathrm{id}_2^3, \mathrm{id}_3^3)] \bmod \mathrm{id}_2^3$; (ж) $\mathrm{НОД} = \mathrm{id}_1^2 - \mu((\mathrm{id}_1^3 \bmod (\mathrm{id}_1^3 - \mathrm{id}_3^3)) + (\mathrm{id}_2^3 \bmod (\mathrm{id}_1^3 - \mathrm{id}_3^3)))$. **277.** $g'(\bar{x}, y) = \prod\limits_{z < y} g(\bar{x}, z)$; $g''(\bar{x}, y) = \mathrm{test}(g'(\bar{x}, y)) \times \mathrm{not}(g(\bar{x}, y))$. Теперь $g(\bar{x}, y)$ равно единице для наименьшего y, для которого $g(\bar{x}, y) = 0$, и равно нулю для всех остальных y. Следовательно, $h(\bar{x}) = \sum\limits_{y < f(\bar{x})} y \times g''(\bar{x}, y)$. Все использованные конструкции не требуют применения минимизации. **278.** $g = \Pr[\langle 0, 1 \rangle, \langle \langle \mathrm{id}_2^2 \rangle_2, \langle \mathrm{id}_2^2 \rangle_1 + \langle \mathrm{id}_2^2 \rangle_2 \rangle]$, $F(x) = \langle g(x) \rangle_1$. **279.** Если f' получена из f заменой значений на b_i в точках a_i: $f'(a_i) = b_i$, для $i = 1, \ldots, k$, то $f'(x) = \mathrm{if}(\mathrm{eq}(x, a_1), b_1, \mathrm{if}(\mathrm{eq}(x, a_2), b_2, \mathrm{if}(\ldots \mathrm{if}(\mathrm{eq}(x, a_k), b_k, f(x)) \ldots)))$. **280.** Функции, построенные из 0 и id_m^n, на единичных аргументах не могут дать значение 2. **281.** $f^-1 = \mu(\mathrm{not}(\mathrm{eq}(f(\mathrm{id}_2^2), \mathrm{id}_1^2)))$. **282.** $f = \mathrm{if}(\mathrm{less}(\mathrm{id}_{n+2}^{n+2}, \mathrm{id}_{n+1}^{n+2}), 0, \sum\limits_{x < s(\mathrm{id}_{n+2}^{n+2})} g(\mathrm{id}_1^{n+2}, \ldots, \mathrm{id}_n^{n+2}, \mathrm{id}_{n+1}^{n+2} + x))$. **283.** $F = \mu(f(\mathrm{id}_1^{n+1}, \ldots, \mathrm{id}_n^{n+1}, \mathrm{id}_{n+1}^{n+1}) \times \mathrm{leq}(g(\mathrm{id}_1^{n+1}, \ldots, \mathrm{id}_n^{n+1}, \mathrm{id}_{n+1}^{n+1}), h(\mathrm{id}_1^{n+1}, \ldots, \mathrm{id}_n^{n+1}, \mathrm{id}_{n+1}^{n+1})))$. **284.** (а) Имеется $n + 1$ пара (x, y), у которых $x + y = n$.

Поэтому количество пар (x, y), у которых $x + y < x_0 + y_0$ равно сумме арифметической прогрессии $1+2+3+\cdots+x_0+y_0 = (x_0+y_0)(x_0+y_0+1)/2$. Среди пар (x, y), у которых $x + y = x_0 + y_0$, пара (x_0, y_0) будет x_0-я по счёту, следовательно, $k(x_0, y_0) = (x_0 + y_0)(x_0 + y_0 + 1)/2 + x_0$. (б) $f(z) = [(\sqrt{1+8z} - 1)/2]$, $k_1(z) = z - f(z)(f(z) + 1)/2$, $k_2(z) = f(z) - k_1(z)$. (в) k — суперпозиция сложения, умножения, целочисленного деления и констант. k_1 и k_2 — суперпозиция сложения, вычитания, умножения, целочисленного деления, констант и извлечения квадратного корня. **285.** (а) $q_0, x \to q^x, \Lambda, +1$, $q^x, y \to q^x, y, +1$, $q^x, \Lambda \to q_1, x, 0$ для всех непустых символов $x, y \in \Sigma$. (б) Отметить первый символ и двигать головку вперёд, пока не найдутся два пробела подряд: $q_0, x \to q_1, x', +1$, $q_1, x \to q_1, x, +1$, $q_1, \Lambda \to q_2, \Lambda, +1$, $q_2, x \to q_1, x, +1$ для всех непустых $x \in \Sigma$. Поставить $*$: $q_2, \Lambda \to q_3, \Lambda, -1$, $q_3, \Lambda \to q_4, *, -1$. После этого переносить символы по одному из начала (помеченного штрихом) за звёздочку: $q_4, x \to q_4, x, -1$, $q_4, d' \to q^d, \Lambda, +1$, $q_4, \Lambda' \to q_5, \Lambda, +1$, $q^x, y \to q_1^x, y', +1$, $q_1^x, z \to q_1^x, z, +1$, $q_1^x, * \to q_2^x, *, +1$, $q_2^x, u \to q_2^x, u, +1$, $q_2^x, \Lambda \to q_4, x, -1$, $q_5, z \to q_5, z, +1$, $q_5, * \to q_6, \Lambda, 0$ для всех $x, y, z, u \in \Sigma$, $z \neq *$, $u \neq \Lambda$. (в) Отметим все символы штрихами, переносим по одному символу из конца на место после первого пустого, сдвигая слово вправо, пустой символ перемещаем на место первого штрихованого: $q_0, x \to q_0, x', +1$, $q_0, \Lambda \to q_1, \Lambda', +1$, $q_1, x \to q_0, x', +1$, $q_1, \Lambda \to q_2, \Lambda, -1$, $q_2, \Lambda' \to q_4, \Lambda, -1$, $q_3, z' \to q_3, z', -1$, $q_3, z \to q_4, z, +1$, $q_4, x' \to q_4^x, \Lambda, +1$, $q_4^z, u' \to q_4^u, z', +1$, $q_4^x, \Lambda \to q_5^x, \Lambda, -1$, $q_4^\Lambda, \Lambda \to q_3, \Lambda, -1$, $q_5^x, z' \to q_5^x, z', -1$, $q_5^x, y \to q_5^x, y, -1$, $q_5^x, \Lambda \to q_6^x, \Lambda, +1$, $q_6^x, y \to q_6^y, x, +1$, $q_6^x, y' \to q_4^y, x, +1$, $q_6^x, \Lambda \to q_7, x, 0$, для всех $x, y, z, u \in \Sigma$, $x, y \neq \Lambda$. (г) Ставим в конце слова звёздочку и по одной переносим буквы за звёздочку, начиная с первой, записывая буквы каждый раз по две штуки: $q_0, x \to q_0, x, +1$, $q_0, \Lambda \to q_1, *, -1$, $q_1, x \to q_1, x, -1$, $q_1, \Lambda \to q_2, \Lambda, +1$, $q_2, z \to q^z, \Lambda, +1$, $q_2, * \to q_f, \Lambda, 0$, $q^x, y \to q^x, y, +1$, $q^x, \Lambda \to q_1^x, x, +1$, $q_1^x, \Lambda \to q_1, x, -1$ для всех $x, y, z \in \Sigma$, $x, y \neq \Lambda$, $z \neq *$. (д) Аналогично предыдущему, только вместо двух a пишем одну, вместо двух b пишем bc, а c не переносим: $q^a, \Lambda \to q_1, a, -1$, $q^b, \Lambda \to q_1^b, b, +1$, $q_1^b, \Lambda \to q_1, c, -1$, $q_2, c \to q_2, \Lambda, +1$. (е) Заменяем a на чётных позициях звёздочками, затем удаляем звёздочки, пока они есть. $q_0, x \to q_1, x, +1$, $q_0, a \to q_1, *, +1$, $q_0, \Lambda \to q_2, \Lambda, -1$, $q_1, y \to q_0, y, +1$, $q_1, \Lambda \to q_2, \Lambda, -1$, $q_2, * \to q_2, \Lambda, -1$,

$q_2, \Lambda \to q_f, \Lambda, 0$, $q_2, y \to q^y, \Lambda, -1$, $q^y, z \to q^z, y, -1$, $q^y, * \to q_3, y, +1$, $q^y, \Lambda \to q_f, y, 0$, $q_3, y \to q_3, y, +1$, $q_3, \Lambda \to q_2, \Lambda, -1$, для всех $x \in \{b, c\}$, $y, z \in \Sigma \setminus \{\Lambda\}$. (ж) Дойти до пробела между словами, поставить звёздочку, по одной удалять буквы a из левого и правого слова, пока в одном из них букв a не останется. Проверить, что и в другом не осталось. $q_0, x \to q_0, x, +1$, $q_0, \Lambda \to q_1, *, +1$, $q_1, z \to q_1, z, +1$, $q_1, a \to q_2, b, -1$, $q_1, \Lambda \to q_3, \Lambda, -1$, $q_2, z \to q_2, z, -1$, $q_2, a \to q_1, z, +1$, $q_2, \Lambda \to q_5, z, +1$, $q_3, z \to q_3, \Lambda, -1$, $q_3, \Lambda \to q_f, 1, 0$, $q_3, a \to q_4, \Lambda, -1$, $q_4, y \to q_4, \Lambda, -1$, $q_4, \Lambda \to q_f, 0, 0$, $q_5, y \to q_5, \Lambda, +1$, $q_5, \Lambda \to q_f, 0, 0$ для всех $x \in \Sigma$, $y \in \{a, b, c, *\}$, $z \in \{b, c, *\}$. (з) Удалять буквы с концов, одновременно проверяя, совпадают ли они. $q^0, \Lambda \to q_f, 1, 0$, $q^0, x \to q^x, \Lambda, +1$, $q^x, y \to q^x, y, +1$, $q^x, \Lambda \to q_1^x, \Lambda, -1$, $q_1^x, x \to q_1, \Lambda, -1$, $q_1^x, \Lambda \to q_f, 1, 0$, $q_1^x, y \to q_2, \Lambda, -1$ при $y \neq x$, $q_1, x \to q_1, x, -1$, $q_1, \Lambda \to q_0, \Lambda, +1$, $q_2, x \to q_2, \Lambda, -1$, $q_2, \Lambda \to q_f, 0, -1$ для всех $x, y \in \Sigma$. (и) Выполнять ряд следующих шагов: последовательно проходить по слову и заменять на b буквы a, стоящие после b или c, и считать, сколько последовательностей a закончилось во время этого прохода. Если их больше одной, то ответ 1. Если букв a больше нет, то ответ 0. (к) Сначала убирать (заменять на звёздочки, например) из слова по одной букве a, одной b и одной c на каждом шаге. Если b или c закончились раньше a, то ответ 0. Иначе нужно проверить, что букв c осталось в два раза больше, чем b: для этого на каждом шаге убираем одну букву b и две буквы cc. Если обе буквы закончились одновременно, то ответ 1, иначе 0. **286.** Вычитаем единицы из двоичной записи и добавляем к унарной. $q_0, \Lambda \to q_f, |, 0$, $q_0, x \to q_1, x, -1$, $q_1, \Lambda \to q_2, |, +1$, $q_2, y \to q_2, y, +1$, $q_2, \Lambda \to q_3, \Lambda, -1$, $q_3, 1 \to q_4, 0, -1$, $q_3, 0 \to q_3, 1, -1$, $q_3, | \to q_5 55, |, +1$, $q_4, y \to q_4, y, -1$, $q_4, \Lambda \to q_2, |, +1$, $q_5, x \to q_5, \Lambda, +1$ для всех $x \in \{0, 1\}$, $y \in \{0, 1, |\}$. **287.** Копировать символы последнего слова по одному, разделяя исходной слово и копию пробелом и штрихуя уже скопированные символы. В конце штрихи убрать. **288.** В примере 109 вместо штриховки смещать рассмотренные символы на одну ячейку влево, разделяя пробелом рассмотренную и нерассмотренную части слова. В примере 110 аналогично использовать пробел, чтобы отделить уже просуммированные разряды от ещё не просумированных. **289.** Дойти вправо до места, где закончатся штрихи, двигаться влево, стирая штрихи у Λ', до первого непустого

символа. Если дошли до Λ^*, то стереть звёздочку и остановиться — лента пуста. В противном случае сдинуть всё на одну ячейкеу влево до звёздочки. Если ячейка со звёздочкой по прежнему пуста: Λ^*, то повторить действия, начиная с q_6, иначе перейти в q_5. **290.** Пусть $\sigma = (q, i, \alpha)$ — конфигурация машины $\mathfrak{M}$. С помощью σ' будем обозначать следующую конфигурация машины $\mathfrak{N}$: $\sigma' = (q', i', \alpha')$, где $q' = q^+$, если $i \geqslant 0$, $q' = q^-$ иначе, $i' = i + 1$, если $i \geqslant 0$, $i' = -i$ иначе, $\alpha'(0) = \#$, $\alpha'(j+1) = \left[\!\!\left[\begin{smallmatrix} \alpha(j) \\ \alpha(-j-1) \end{smallmatrix} \right]\!\!\right]$. Индукцией по k доказывается такое утверждение: если $\sigma \vdash^k_{\mathfrak{M}} \tau$, то $\sigma' \vdash^\ell_{\mathfrak{N}} \tau'$ для некоторого $\ell \geqslant k$. Для $k = 0$ получаем $\sigma = \tau$, поэтому $\ell = 0$. Если для k утверждение доказано и $\sigma \vdash^{k+1}_{\mathfrak{M}} \tau$, то $\sigma \vdash^k_{\mathfrak{M}} \rho$ и $\rho \vdash_{\mathfrak{M}} \tau$. По индукционному предположению $\sigma' \vdash^\ell_{\mathfrak{N}} \rho'$ для $\ell \geqslant k$. В зависимости от команды, исполняемой в конфигурации ρ рассматриваем соответствующую часть программы машины $\mathfrak{N}$ и показываем, что $\rho' \vdash^1_{\mathfrak{N}} \tau'$ или $\rho' \vdash^2_{\mathfrak{N}} \tau'$. Следовательно, $\sigma' \vdash^{\ell'}_{\mathfrak{N}} \tau'$ для $\ell' \geqslant \ell + 1 \geqslant k + 1$. Пусть σ_0 — начальная конфигурация машины $\mathfrak{M}$, тогда σ'_0 — вторая конфигурация машины $\mathfrak{N}$ на том же входе. Если $\mathfrak{M}$ останавливается, то $\sigma_0 \vdash^k_{\mathfrak{M}} \tau$ для некоторой заключительной конфигурации τ. Согласно утверждению $\sigma'_0 \vdash^\ell_{\mathfrak{N}} \tau'$, где τ' также будет заключительной конфигураций. Поскольку τ была стандартной, то τ' содержит те же символы, что и τ, поэтому результаты машин $\mathfrak{M}$ и $\mathfrak{N}$ совпадают. Пусть теперь $\mathfrak{M}$ не останавливается, тогда для любого k существует τ_k такое, что $\sigma_0 \vdash^k_{\mathfrak{M}} \tau_k$. Согласно утверждению $\sigma'_0 \vdash^\ell_{\mathfrak{N}} \tau'_k$ для $\ell \geqslant k$, а так как k может быть сколь угодно большим, то $\mathfrak{N}$ также не останавливается. **291.** В начале работы раздвинуть ячейки, вставив на чётных позициях пустой символ. Далее вернуться в первую ячейку и перейти в состояние q_0^+. Для всех состояний добавляем команду $q^-, \# \to q^+, \#, +1$ (нулевая ячейка имеет чётный номер, поэтому там оказываемся в «отрицательных» состояниях). Вместо команды $q, a \to p, b, 0$ вставляем $q^+, a \to p^+, b, 0$, $q^-, a \to p^-, b, 0$. Вместо команды $q, a \to p, b, +1$ вставляем $q^+, a \to p_1^+, b, +1$, $p_1^+, c \to p^+, c, +1$, $q^-, a \to p_1^-, b, -1$, $p_1^-, c \to p^-, c, -1$ для всех символов $c \neq \#$. Вместо команды $q, a \to p, b, -1$ вставляем $q^+, a \to p_2^+, b, -1$, $p_2^+, c \to p^+, c, -1$, $p_2^+, \# \to p_2^-, \#, +1$, $q^-, a \to p_2^-, b, +1$, $p_2^-, c \to p^-, c, +1$, для всех символов $c \neq \#$. В конце нужно «сжать» ленту, убрав пустые символы в чётных ячейках. **292.** При помощи функции digit (задача 276) ищем

позиции $z_1, \ldots, z_m$, на которых в k-ичной записи r стоят нули, после чего разбиваем эту запись на соответствующие части: $r_1 \leftarrow r \bmod \mathrm{pow}(k, z_1)$, $r_i \leftarrow [(r \bmod \mathrm{pow}(k, z_i))/\mathrm{pow}(k, z_{i-1}+1)]$ при $i > 1$. **293.** Сначала можно копировать вход, разделяя исходный код и копию двумя пробелами, при этом после каждого символа вставлять a. Далее ещё правее строим унарную запись числа: отнимая на каждом шаге по единице от копии и добавляя a к унарной записи. Перемежающиеся буквы a при этом не трогаем, так как они задают границу копии. В результате получается унарное представление кода ленты со входом. Остаётся только стереть всё левее этого кода и сдвинуть его к началу. Восстановление выхода — в обратной последовательности: убирая по одной букве a и добавляя единицу к выходу, в котором основные символы перемежаются с a, после этого убираем промежуточные a, отделяя результат двумя пробелами, и наконец передвигаем результат к началу. **294.** На первом этапе вместо палочек вставляем нули. На втором этапе в начале пропускаем единицы и нули. Для команды инкремента: доходим до конца i-го числа, начинаем прибавлять единицу, учитывая переносы, если перенос произошёл из последнего разряда на пробел-разделитель, то правую часть ленты сдвигаем. Для команды декремента: доходим до конца i-го числа, проверяя, есть ли в числе единицы. Если их нет, то возвращаемся назад и переходим к следующей команде, если есть, то начинаем вычитать единицу, двигаясь справа налево, учитывая заемы. Ветвление: аналогично декременту проверяем, есть ли в i-м числе хоть одна единица и переходим либо к одной, либо к другой команде. **295.** (а) Записать | на место пустого символа между словами, стереть две палочки в конце. (б) Стереть по одной палочке из второго слова, затем из первого. После того, как во втором слове стёрта последняя палочка, первое слово является разностью. (в) Стереть самую первую палочку первого слова и записать новое слово | (ноль) правее второго. Далее выполнять такую последовательность, пока первое слово не опустеет: убрать одну палочку из первого слова, скопировать второе слово в конец, убрать одну палочку, соединить его с третьим. После этого третье слово является произведением. (г) Стирать по одной палочке из каждого слова, если первое опустело раньше, то ответ 1, иначе 0. (д) Стереть самую первую палочку первого слова и

записать новое слово || (один) правее второго. Далее выполнять такую последовательность, пока первое слово не опустеет: убрать одну палочку из первого слова, скопировать второе слово в конец, поставить перед третьим разделитель $\#'$, запустить машину (в), чтобы найти произведение, стереть разделитель. После этого третье слово является степенью. (е) Записать новое слово | (ноль) правее исходного. Далее выполнять такую последовательность. Скопировать в конец первое слово и два раза второе, поставить перед четвёртым разделитель $\#'$, запустить машину (в), чтобы найти квадрат, перенести разделитель перед третьим словом, запустить машину (г), чтобы сравнить исходное слово с квадратом. Если ответ 0, то добавить палочку ко второму слову и начать всё заново. Иначе — стереть одну палочку из второго слово, это и будет результат. (ж) Стереть самую первую палочку первого слова и записать новое слово | (ноль) правее исходного. Далее выполнять такую последовательность, пока в первом слове не останется только одна палочка: стереть вторую половину первого слова, добавить палочку ко второму. После этого второе слово является логарифмом. (з) Вставить новое слово | (ноль) между исходными. Далее выполнять такую последовательность, пока в третьем слове не останется только одна палочка: добавить палочку ко второму слову (сдвинув ленту вправо), скопировать первое слово в конец, вставить разделитель $\#'$ после второго, запустить машину (б), стереть разделитель. После этого из второго слова убрать одну палочку, оно и будет частным. (и) Создать копии исходных слов, отделив их разделителем $\#'$, запустить машину (з), передвинуть разделитель после первого слова, запустить машину (в), убрать разделитель, запустить машину (б). (к) Стереть всё, кроме m-го слова и передвинуть его к началу. **296.** Использовать методы комбинации машин описанные в конце параграфа 20.4 и предыдущей задаче. **297.** Сравнивать символы в словах по одному слева направо, штрихуя уже рассмотренные. Как только обнаружатся неравные символы или одно из слов закончится, стереть всё на ленте и записать ответ. **298.** (а) Создать две копии каждого состояния q^+ и q^-, в которых запоминать направление последнего сдвига: заменить команды $q, a \to p, b, +1$ на $q^x, a \to p^+, b, +1$, команды $q, a \to p, b, -1$ на $q^x, a \to p^-, b, -1$, команды $q, a \to p, b, 0$ на $q^x, a \to p^x, b, 0$. Тогда возврат происходит командой $q^s, a \to p, b, s1$. (б)

Отметить в начале работы нулевую ячейку, например, звёздочкой: $q'_0, a \to q_0, a^*, 0$, сохранять эту звёздочку до конца работы. Для возврата в нулевую ячейку двигать головку влево и вправо со всё увеличивающейся амплитудой, пока звёздочка не будет найдена. (в) Реализовать пункт (б), для «отражения» отметить текущую ячайку, например, штрихом, затем использовать метод из пункта (б) для поиска звёздочки, когда будет понятно, с какой стороны от звёздочки находится головка, отсчитать от звёзлочки в противоположную сторону столько же ячеек, сколько находится от звёзлдочки до штриха. (г) Аналогично (в), только в конце отсчитывать ячейки от штриха. (д) Реализовать пометку штрихами как в теореме 155. Для реализации поиска a следует отметить текущую ячейку, далее искать справа букву a', пока не закончатся штрихи. Если буква найдена, то стереть метку со старой ячейки и перейти в новую. Если не найдена — вернуться в старую ячейку. (е) Реализовать пометку штрихами как в теореме 155. Для вставки сдвинуть все заштрихованые ячейки на одну позицию вправо, записать в освободившуюся ячейку b'. (ж) Расширить алфавит символами вида a^b, где $a, b \in \Sigma$, b будет обозначать начальный символ. Команды $q, a \to p, b, s$ заменяем на $q, a \to p, b^a, s$ и $q, a^c \to p, b^c, s$. Для восстановления выполняется команда $q, a^b \to p, b, s$. **300.** $(|Q| \times |\Sigma| \times 3 + 1)^{|Q|\times|\Sigma|}$. **303.** $\mathrm{bb}(1) = 1$: $q_0, \Lambda \to q_0, |, 0$. $\mathrm{bb}(2) = 4$: $q_0, \Lambda \to q_1, |, +1$; $q_1, \Lambda \to q_0, |, -1$; $q_0, | \to q_1, |, -1$. **304.** $A \leqslant_m A$ посредством тождественной функции id^1_1, если $A \leqslant_m B$ и $B \leqslant_M C$ посредством f и g соответственно, то $A \leqslant_M c$ посредством $f \circ g$. **305.** Использовать тот же метод, что и в теореме 170. **306.** Свести TOTAL к ZERO с помощью такой модификации программы: построить машину со стандартной заключительной конфигурацией, а после остановки стереть весь вход. **307.** Свести ZERO к EQU: модифицировать машину как в предыдущей задаче, а в качестве второй — взять машину, просто стирающую весь вход. **308.** $I_{A\cap B} = I_A \times I_B$, $I_{A\cup B} = \mathrm{test}(I_A + I_B)$. **309.** Для выяснения $x \in A + B$ перебрать все $y = 0, \ldots, x$ и проверить $y \in A$, $x - y \in B$, если хотя бы в одном случае выпоолнено, ответ «да». Аналогично для умножения. **310.** Для конкатенации — аналогично предыдущей задаче. Для проверки $x \in L^*$ перебирать все y — собственные префиксы x, и проверять $y \in L$ и $z \in L^*$ рекурсивно ($x = yz$). **311.** $F(x) = \mathrm{if}(I_A(x), g(x), h(x))$.

312. Воспользоваться универсальной машиной Тьюринга, отсчитывать количество проделанных шагов. **313.** Спроектировать ограниченную проблему остановки в HALT, убрав t. **314.** В доказательстве теоремы 172 заменить в алгоритме $\mathfrak{M}$ условие $\mathrm{arc}(w) > n$ на $f(w) > n$.

Указатель обозначений

Указатель терминов

Литература

[1] Агафонов В. Н. Математические основы обработки информации. — Новосибирск : НГУ, 1982. — 92 с. [Главы 9–13]

[2] Верещагин Н. К. Шень А. Начала теории множеств. — М. : МЦНМО, 2000. — 128 с. [Глава 1]

[3] Виленкин Н. Я. Популярная комбинаторика. — М. : Наука, 1975. — 208 с. [Глава 2]

[4] Гаврилов Г. П., Сапоженко А. А. Задачи и упражнения по дискретной математике. — 3-е изд., перераб. — М. : Физматлит, 2009. — 416 с. [Главы 3–5]

[5] Гиндикин С. Г. Алгебра и логика в задачах. — М. : Наука, 1972. — 288 с. [Главы 3–5]

[6] Дейт К. Дж. Введение в системы баз данных. — 7-е изд. — М. и др. : Вильямс, 2002. — 1071 с. [Глава 8]

[7] Дехтярь М. И. Лекции по дискретной математике. — М. : Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2012. — 259 с.

[8] Дудаков С. М., Карлов Б. Н. Математическое введение в информатику — Изд. 2-е, испр. и доп. — Тверь : ТвГУ, 2017. — 320 с. [Глава 18]

[9] Ерусалимский Я. М. Дискретная математика : Теория, задачи, приложения. — 6-е изд. — М. : Вузовская книга, 2004. — 265 с.

[10] Карпов Ю. Г. Теория автоматов. — М. и др. : Питер, 2003. — 206 с. [Главы 15–17]

[11] Кристофидес Н. Теория графов : алгоритмический подход. — М. : Мир, 1978. — 432 с. [Главы 9–12]

[12] Липский В. Комбинаторика для программистов. — М. : Мир, 1988. — 213 с. [Главы 2 и 12]

[13] Мейер Д. Теория реляционных баз данных. — М. : Мир, 1987. — 608 с. [Главы 6 и 8]

[14] Новиков Ф. А. Дискретная математика : для бакалавров и магистров. — 2-е изд. — М. [и др.] : Питер, 2013. — 399 с.

[15] Оре О. Теория графов. — М. : Наука, 1968. — 352 с. [Главы 9–12]

[16] Соминский И. С. О математической индукции / И. С. Соминский, Л. И. Головина, И. М. Яглом. — М. : Наука, 1967. — 145 с. [Глава 2]

[17] Тайцлин М. А., Столбоушкин А. П. Математические основания информатики. — Тверь : ТвГУ, 2013. — 377 с. [Главы 18–21]

[18] Трахтенброт Б. А. Алгоритмы и вычислительные автоматы. — М. : Советское радио, 1974. — 200 с. [Главы 19–21]

[19] Трахтенброт Б. А., Барздинь Я. М. Конечные автоматы : (поведение и синтез). — М. : Наука, 1970. — 400 с. [Главы 15–17]

[20] Яблонский С. В. Введение в дискретную математику. — Изд. 5-е, стер. — М. : Высшая школа, 2008. — 384 с. [Главы 3–5 и 13]

Учебное издание

Дехтярь Михаил Иосифович
Дудаков Сергей Михайлович
Карлов Борис Николаевич

Лекции по дискретной математике

Учебник

В авторской редакции

Подписано в печать 12.08.2019.
Усл. п. л. 32. Уч.-изд. л. 20,6.
Тираж 10 экз. [электр.].
Заказ № 289 от 12.08.2019.

Тверской государственный университет
Факультет прикладной математики и кибернетики
Адрес: 170100, г. Тверь, ул. Желябова, 33