Донской В.И.

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

Учебное пособие

Симферополь
«КУБ»
2016

ББК 22.18+32.811
УДК 519.7
Д676

ДОНСКОЙ В. И.

Д676 Теоретические основы информатики: учебное пособие / В.И.Донской.– Симферополь: КУБ, 2016. – 232 с.

ISBN 978-5-9908044-1-8

Книга предназначена для студентов математических специальностей, но будет полезной не только математикам, но и программистам и инженерам-разработчикам информационных систем. Её появление связано с необходимостью обеспечения литературой обязательного курса "Теоретические основы информатики", входящего в учебный план специальности "Прикладная математика и информатика". Также предполагается использование этой книги студентами специальности "Математика" при изучении предмета "Математические основы информатики".

ББК 22.18+32.811
УДК 519.7

Табл. 7. Ил. 53. Библиогр. 12 назв.

Рецензент – заведующий кафедрой информатики и математики Международного университета в Москве (МУМ) профессор Л. М. Местецкий.

УЧЕБНОЕ ИЗДАНИЕ

ДОНСКОЙ Владимир Иосифович

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ
Учебное пособие

Подписано в печать 17.02.2016 г. Формат 60x84/16.
Усл. печ. л. 13,49. Тираж 50 экз. Заказ № 1702/02.

ИП Лавриненко Е. В.
295022, г. Симферополь, ул. Кечкеметская, 97.
Телефон: (978) 800-30-22.
E-mail: cube_simf@mail.ru http://cube-press.com

Отпечатано с готового оригинал-макета в полиграфцентре «КУБ» (ИП Лавриненко Е.В.)
295000, г. Симферополь, ул. Одесская, 4. Тел. (978) 772 40 21

ISBN 978-5-9908044-1-8

Содержание

1 Введение

Информатика – наука, изучающая общие свойства информации, закономерности и принципы ее создания, накопления, поиска, передачи, преобразования.

Объектом исследования информатики как науки является широчайшее поле информационных процессов.

Предметом исследований в информатике является совокупность всевозможных характеристик и особенностей информационных процессов, например, таких как кодирование информации, алгоритмическое преобразование информации, в том числе с целью решения математических задач, построение моделей поиска необходимой информации, анализ и распознавание свойств информации и многое другое.

Становление информатики происходило на основе результатов исследований в области математической логики, теории алгоритмов, создания вычислительных машин и развития теоретического и прикладного программирования.

Термин "информатика" (*Informatics, Information Science*) впервые ввел французский учёный Филипп Дрейфус в 1962 г. Слово информация происходит от латинского *informatio* – разъяснение, изложение, осведомленность. Неформально, информация – это сведения (сообщения, данные), рассматриваемые независимо от формы их представления.

Единого определения информации как научного термина не существует. Это основное, первичное понятие, такое как точка или прямая в планиметрии. Можно сказать, что *информация – это общенаучный базовый термин.*

Тем не менее, можно говорить о способах представления информации, формализовать эти способы математически. Например, строковое представление: информация представляется в форме множества строк над заданным конечным алфавитом; числовое представление – массивом чисел; представление в виде сигнала – вещественной функции времени на заданном отрезке.

Информация как центральное понятие информатики является абстрактным, не имеет обязательной физической природы, что свой-

ственно математическим наукам. Тем не менее, при рассмотрении некоторых информационных процессов приходится учитывать физические свойства объектов и окружающей среды.

Место информатики в последовательности эпох развития человечества показано в следующей таблице.

Этапы развития человечества	**I Натуральный**	**II Энергетический**	**III Информационный**
Исследуемые процессы и объекты	Отдельные предметы и их совокупности	Энергетические процессы, движение, механическое управление	Процессы обработки информации, компьютеры
Основные математические объекты	Натуральные числа, простые фигуры	Непрерывные функции	Информация
Развивающиеся разделы математики	Арифметика и геометрия	Дифференциальное и интегральное исчисление	Теория алгоритмов, теория информации [6] и вычислительная математика
Применение математики	Методы и устройства счёта, измерения фигур	Непрерывные математические модели	Информационные и алгоритмические модели

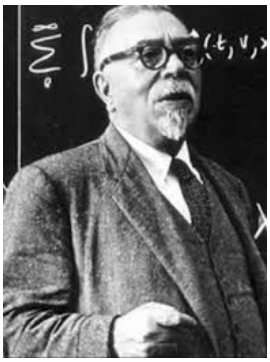


Рис. 1.1: Норберт Винер (1894 – 1964), выдающийся американский математик, пионер кибернетики

До повсеместного применения термина "информатика", к началу 50-х годов XX века уже вошла в обиход и использовалась для обозначения широкой области знаний *кибернетика* (от греческого $\kappa\upsilon\beta\varepsilon\rho\nu\eta\tau\iota\kappa\eta$ — "искусство управления") — наука об общих закономерностях управления сложными системами — машинами, живыми организмами, социальными системами, в том числе на основе использования сбора и преобразования информации. Так определил эту науку и сформулировал основные её положения в своем фундаментальном труде "Кибернетика, или управление и связь в животном и машине"в 1948 г. выдающийся математик Норберт Винер [3]. Ранее термин "кибернетика" уже использовался А.-М. Ампером, который в своем фундаментальном труде "Наброски по философии науки"(1834—1843) определил ки-

бернетику как науку об управлении государством. Но именно начиная с пионерских работ Н. Винера кибернетика получила направленность на автоматическое управление, самообучение и обрела черты математической науки.

Объектом исследования кибернетики является широкий круг процессов управления, а предмет её исследования включает теорию управляющих систем, и прежде всего — автоматов. В этом направлении работал один из величайших математиков XX века Джон фон Нейман, создатель общей математической теории автоматов. Универсальным управляющим цифровым автоматом, как определил фон Нейман, является цифровая вычислительная машина (компьютер).

В 1946 году трое учёных — Артур Бёркс, Герман Голдстайн и Джон фон Нейман — опубликовали статью "Предварительное рассмотрение логического конструирования электронного вычислительного устройства". В статье была предложена структура компьютера, описывались принципы программного управления, выдвигалась идея использования общей памяти для программы и данных. Эти идеи, актуальные и в настоящее время, получили название "принципы фон Неймана". С этой поры и по настоящее время в кибернетике и информатике главными объектами становятся компьютеры, программное обеспечение, информационные технологии.

С течением времени и тотальным применением компьютеров практически во всех разделах науки, в области экономики, техники, в быту — закономерно возникает термин "компьютерные науки", объединяющий все теоретические и прикладные аспекты создания и применения компьютеров. Однако этот термин не вполне обоснованно получает смысловую нагрузку, соответствующую информатике. Во многих странах слово "информатика" используется редко, уступая термину "*Computer Science*". Теоретическая информатика часто обозначается "*Computer Science Theory*". Но термин "информатика" представляется существенно более широким, чем *Computer Science*, поскольку *с каждым годом все большее значение приобретают задачи обработки речи, изображений, других описываемых изначально непрерывными функциями времени видов информации, а также развитие новых способов переработки информации, основанных на*

квантовомеханических, оптоэлектронных и других принципах.

Возвращаясь к идеям Норберта Винера, следует обратить внимание на то, что кибернетика не ограничивается рассмотрением только процессов компьютерного управления. Речь идет даже об управлении и связи в живых организмах. А самой сложной из известных в настоящее время биологических систем обработки, хранения, выдачи информации и принятии управляющих решений и действий является человек.

Рис. 1.2: Джон фон Нейман (1894 – 1964), один из самых выдающихся и знаменитых математиков XX века

Изучение сложнейших методов обработки информации, свойственных человеку, таких как узнавание, понимание, запоминание, привело к становлению нового важного раздела информатики, который в настоящее время получил название "искусственный интеллект".

Обработка информации может происходить с различными целями. В том числе – с целью управления. Поэтому можно полагать, что информатика включает в себя кибернетику. А также включает в себя *Computer Science*. С другой стороны, практически всегда информация извлекается с целью управления. Хотя можно представить некоторые информационные задачи лингвистики, криптографии, специфические задачи наблюдения, которые непосредственно для управления не используются.

На пути к созданию компьютеров математики провели широкие исследования в области теории алгоритмов – рекурсивных функций, машин Тьюринга, теории вычислимости, алгоритмической сложности и разрешимости. Такие исследования иногда носят чисто математический характер, и соответствующие разделы в настоящее время определены в математике. Их называют: математическая кибернетика, компьютерная математика, теоретическая информатика.

Наконец, нужно подчеркнуть, что на современном этапе сложился целый ряд направлений в информатике, основными из которых являются:

теоретическая (*математическая*) *информатика*, которой, в частности, посвящено это учебное пособие;

техническая информатика, изучающая вопросы создания информационных систем и машин как физических объектов;

информационные технологии;

биологическая информатика;

экономическая информатика.

В соответствии с перечисленными направлениями в настоящее время сформировались в некоторой степени различающиеся представления об информатике как науке и подходы к её изучению и преподаванию.

В предлагаемом учебном пособии информатика излагается на основе математического подхода.

Книга рассчитана на студентов математических специальностей, но будет полезна инженерам и программистам. Её появление связано с необходимостью обеспечения литературой обязательного курса "Теоретические основы информатики", входящего в учебный план специальности "Прикладная математика и информатика". Кроме этого, предполагается использование этой книги студентами специальности "Математика" при изучении предмета "Математические основы информатики".

Теоретические основы информатики в широком смысле включают в себя многие разделы: теорию автоматов, теорию кодирования, теорию программирования и другие. Они преподаются студентам специальности "Прикладная математика и информатика" на младших курсах и поэтому не включены в настоящее учебное пособие. Но знание этих разделов (также как дискретной математики, математического анализа, алгебры, теории вероятностей) облегчит понимание курса "Теоретических основ информатики".

Большинство необходимых понятий, использованных при изложении материала, поясняется по мере их использования, в некоторых случаях – размещаются в приложениях к учебному пособию. Предполагалось, что оно будет небольшим, компактным, удобным в работе и понятным студентам.

2 Измерение информации. Энтропия и её свойства

2.1 Энтропия источника дискретной информации

Общая схема передачи информации представлена на рис. 2.3. Будем рассматривать передачу источником дискретной информа-

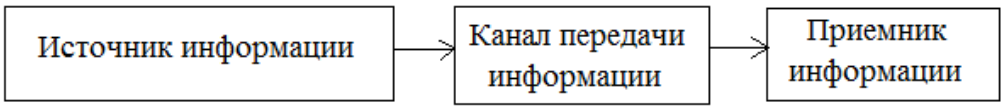


Рис. 2.3: Схема передачи информации

ции, представляемой словами, составленными из букв конечного алфавита $A = \{a_1, ..., a_n\}$. Тогда различных слов длины l будет ровно $M = n^l$. Это число характеризует *разнообразие источника информации*, производящего слова равной длины над указанным алфавитом.

Ральф Хартли (1888 − 1970) предложил в 1928 г. измерять количество информации рассматриваемого типа как

$$I = log_a M,$$

где M − количество различимых состояний источника информации.

Выбор основания логарифма $a = e, 10, 2$ определил название соответствующей единицы измерения информации − *нат, дит и бит* (*binary digit*).

Введенную таким образом меру информации называют мерой Хартли или логарифмической мерой.

Если алфавит двоичный ($n = 2$), и передается один двоичный символ, то

$$I = log_2 M = log_2 2^1 = 1 \ бит,$$

а при передаче l двоичных символов

$$I = log_2 2^l = l \ бит.$$

Далее при измерении информации будем пользоваться только логарифмами по основанию 2 и писать log, подразумевая именно это основание.

В соответствии с таким выбором единицы измерения, 16−ти разрядный двоичный регистр компьютера может быть источником информации в 16 бит, 32−х разрядный – источником информации в 32 бит и т.д.

Введенное определение информации не учитывает, с какой исходной вероятностью в данных может появиться каждый символ алфавита или определённое слово.

Пусть имеется M равновероятных исходов до извлечения информации (например, перед броском правильной игральной кости $M = 6$). Измеряемая при этом неопределённость исхода должна возрастать с ростом M. Эту возрастающую с ростом числа исходов неопределённость называют *энтропией* и обозначают

$$H = f(M),$$

где f − функция от числа исходов, *вид которой нужно определить*. Это было сделано Клодом Шенноном с учётом следующих исходных требований.

1. Функция f должна быть неотрицательной возрастающей функцией.

2. $f(1) = 0$. Действительно, если, к примеру, произведен бросок игральной кости, то уже реализован один конкретный исход, получен результат – достоверная информация об этом исходе, и тогда неопределённости уже нет, $M = 1$, $f(M) = 0$.

3. Функция f должна быть аддитивной.

Требование аддитивности поясняется следующим образом. Если одноразовая передача информации от источника может давать m

исходов, то передача независимо n раз такой информации может давать $M = m^n$ исходов. Тогда, согласно условию аддитивности, должно выполняться соотношение

$$f(m^n) = nf(m).$$

Обозначим $m^n = x,\ m \neq 0$. Тогда $n = \frac{\ln x}{\ln m}$;

$$f(x) = \frac{\ln x}{\ln m} f(m) = \frac{f(m)}{\ln m} \ln x;\ \ f(x) = K \ln x,$$

где $K = \frac{f(m)}{\ln m}$ – положительная константа, не зависящая от x, являющегося аргументом искомой функции $f(x)$. Поэтому вид функции f определен с точностью до константы, которая может служить для задания единицы измерения энтропии.

Если $K = 1$, то $H = \ln M$ *нат*;

Если $K = \frac{1}{ln2}$, то $H = \log M$ *бит*, поскольку $\ln x = \ln 2 \log x$.

Пусть источник информации содержит M равновероятных возможных передаваемых сообщений. Тогда каждое из них может быть передано с вероятностью $p_i = \frac{1}{M},\ i = 1, ..., M$, в предположении что $\sum_i p_i = 1$. Поскольку

$$-\log p_i = -\log \frac{1}{M} = \log M,$$

то для M равновероятных исходов (каждый – с вероятностью p) формула энтропии имеет вид

$$H = -\log p$$

Неопределённость, оставшуюся после получения информации, называют *апостериорной* (*aposteriori – после опыта*) и обозначают H_{aps}. А исходную, имевшуюся до получения информации неопределённость, называют *априорной* (*apriori – до опыта*) и обозначают H_{apr}. Количество полученной информации естественно измерять величиной исчезнувшей неопределённости

$$I = H_{apr} - H_{aps}.$$

Если $H_{aps} = 0$, то количество полученной информацией может совпадать с величиной исходной неопределённости, и тогда это количество информации называют количеством информации источника.

При передаче символьной информации символы – буквы алфавита источника $a_1, ..., a_n$ могут передаваться с различными вероятностями. Так, при передаче информации на русском языке, вероятность появления в сообщении буквы *ы* гораздо меньше вероятности появления буквы *к*. Поэтому в общем случае

дискретный источник информации характеризуется так называемым ансамблем

$$\begin{pmatrix} a_1 & a_2 & ... & a_n \\ p_1 & p_2 & ... & p_n \end{pmatrix},$$

где символам a_i приписаны вероятности их появления p_i, и выполняется условие нормировки $p_1 + ... + p_n = 1$.

Таким образом, ансамбль – это набор данных вместе с их вероятностными характеристиками или распределение дискретной случайной величины, которая может принимать n значений.

Клод Шеннон в 1948 г. предложил считать информационной мерой ансамбля величину

$$H(p_1, ..., p_n) = -\sum_{i=1}^{n} p_i \log p_i,$$

названную *энтропией* источника информации.

Шенноновская энтропия является математическим ожиданием случайной величины или усредненной энтропией передаваемого сообщения.

2.2 Свойства энтропии источника дискретной информации

Сначала рассмотрим случай, когда источник информации может находится только в двух состояниях, обозначаемых, например, 0 и 1, с вероятностями, соответственно, p, $0 \le p \le 1$, и $1-p$. Тогда

энтропия

$$H(p) = -p\log p - (1-p)\log(1-p)$$

определена на отрезке $0 < p \leq 1$.

Для построения графика функции $H(p)$ определим предел

$$\lim_{p\to 0} p\log p,$$

который, как легко убедиться, равен нулю; доопределим $H(0) = 0$; $H(1) = 0$: $H(\frac{1}{2}) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} = 1$. Вторая производная

$$H''(p) = -\frac{\log e}{p} - \frac{\log e}{1-p}$$

меньше нуля в интервале $(0,1)$, поэтому $H(p)$ – выпуклая вверх функция.

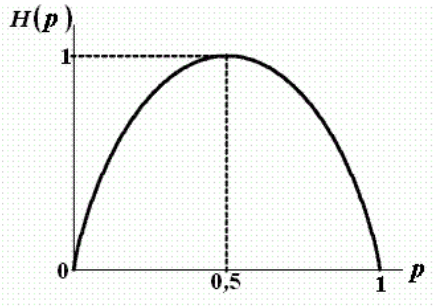


Рис. 2.4: График функции $H(p)$

График функции $H(p)$ (рис.2.4) симметричен относительно прямой $p = \frac{1}{2}$. При $p = \frac{1}{2}$ мера Шеннона совпадает с мерой Хартли: $H(\frac{1}{2}) = 1$ *бит*.

Пример. Пусть студент может априорно сдать экзамен с вероятностью $p = \frac{3}{4}$ и не сдать экзамен – с вероятностью $1 - p = \frac{1}{4}$. Энтропия источника информации о возможной сдаче экзамена этим студентом есть

$$H\Big(\frac{3}{4}\Big) = -\frac{3}{4}\ log\frac{3}{4} - \frac{1}{4}\log\frac{1}{4}$$

$$= \frac{3}{4}\ log\frac{4}{3} + \frac{1}{4}\log 4 \approx 0,75\cdot 0,416 + 0,5 \approx 0,81.$$

Свойство 1. Энтропия

$$H(p_1, ..., p_n) = -\sum_{i=1}^{n} p_i \log p_i$$

равна нулю только в том случае, когда одно из значений p_i равно единице, а остальные вероятности равны нулю.

Доказательство. Пусть для некоторого $j \quad p_j = 1 \quad$ и $\forall i \neq j$ $p_i = 0$. Легко проверить, что в этом случае $H(p_1, ..., p_n) = 0$.

Если же не существует значения $p_i = 1$, то для всех или некоторых i значения $p_i > 0$, но $0 \leq p_i < 1$, и тогда $-\log p_i = \log \frac{1}{p_i} > 0$ и $H(p_1, ..., p_n) = -\sum_{i=1}^{n} p_i \log p_i > 0$.

Свойство 2. Энтропия $H(p_1, ..., p_n) = -\sum_{i=1}^{n} p_i \log p_i$ является выпуклой функцией как сумма выпуклых функций.

Свойство 3. Энтропия $H(p_1, ..., p_n) = -\sum_{i=1}^{n} p_i \log p_i$ достигает максимального значения $\log n$ при $p_1 = ... = p_i = ... = p_n = \frac{1}{n}$.

(На рис. 2.5 представлен график функции $H(p_1, p_2, p_3)$, где вероятность p_3 задана формулой $p_3 = 1 - (p_1 + p_2)$ для удобства изображения этого графика. Максимум достигается при $p_1 = p_2 = p_3 = \frac{1}{3}$.)

Доказательство. Функция H является выпуклой вверх, непрерывной, дифференцируемой и заданной на симплексе

$$\{p_1, ..., p_i, ..., p_n : p_i \geq 0, \ \sum_{i=1}^{n} p_i = 1\}.$$

Поэтому для нахождения её единственного максимума при условии $\sum_{i=1}^{n} p_i - 1 = 0$ можно применить метод множителей Лагранжа. Функция Лагранжа

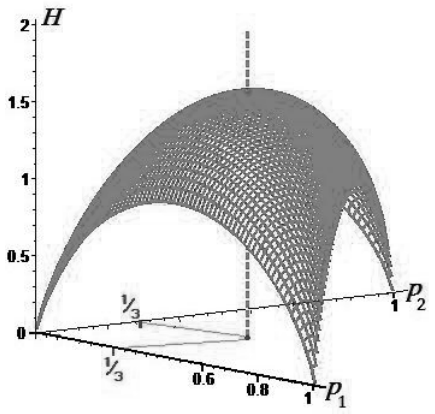


Рис. 2.5: График функции $H(p_1, p_2, p_3)$, где $p_3 = 1 - (p_1 + p_2)$

$$F(\tilde{p}, \lambda) = -\sum_{i=1}^{n} p_i \log p_i + \lambda(\sum_{i=1}^{n} p_i - 1);$$

$$\frac{\partial F(\tilde{p}, \lambda)}{\partial p_i} = -\log p_i - \frac{1}{\ln 2} \cdot p_i \cdot \frac{1}{p_i} + \lambda.$$

Приравнивая частные производные к нулю, получаем

$$\log p_i = \lambda - \frac{1}{\ln 2},$$

откуда следует, что все вероятности p_i равны одной и той же величине. Тогда $p_i = \frac{1}{n}, \quad i = 1, ..., n$, и, следовательно,

$$\max_{\tilde{p}} H(\tilde{p}) = -\sum_{i=1}^{n} \frac{1}{n} \log \frac{1}{n} = -\log \frac{1}{n} = \log n.$$

Пример. Определить максимально возможную энтропию системы S, состоящей из трёх элементов, каждый из которых может находиться в одном из четырёх возможных состояний.

Поскольку общее число состояний n системы равно $4^3 = 64$, то максимальная энтропия $\max H(S) = \log n = \log 64 = 6(\textit{бит})$, и максимум достигается в случае, когда все состояния системы равновероятны.

Свойство 4. Всякое изменение вероятностей $p_1, ..., p_i, ..., p_n$ в сторону их выравнивания влечёт увеличение энтропии $H(\tilde{p})$.

Доказательство. Не теряя общности, возьмём пару вероятностей p_1 и p_2, и пусть $p_2 > p_1$. Изменим эти вероятности на достаточно малую величину $\Delta p: \ 0 < \Delta p < \frac{p_2 - p_1}{2}$ так, что $p_1^* = p_1 + \Delta p$, а $p_2^* = p_2 - \Delta p$, и покажем, что

$$H(p_1^*, p_2^*, p_3, \ldots, p_n) > H(p_1, p_2, p_3, \ldots, p_n).$$

Действительно, приращение энтропии как функции многих переменных определяется выражением

$$\begin{aligned} \Delta H &= H(p_1^*, p_2^*, p_3, \ldots, p_n) - H(p_1, p_2, p_3, \ldots, p_n) \\ &= \frac{\partial H}{\partial p_1} \Delta p + \frac{\partial H}{\partial p_2}(-\Delta p) + o(\rho) \\ &= \Big(\frac{\partial H}{\partial p_1} - \frac{\partial H}{\partial p_2}\Big) \Delta p + o(\rho), \end{aligned}$$

где $\rho = \sqrt{\Delta p^2 + \Delta p^2} \ = \ \Delta p \sqrt{2} > 0.$

$$\frac{\partial H}{\partial p_1} = -\log p_1 - \log e; \quad \frac{\partial H}{\partial p_2} = -\log p_2 - \log e;$$

$$\begin{aligned} \Delta H &= (\log p_2 - \log p_1) \Delta p + o(\rho) \\ &= \Delta p \log \frac{p_2}{p_1} + o(\rho) > 0, \end{aligned}$$

поскольку $\Delta p > 0$ и $p_2 > p_1$.

Пример. Имеются два ящика, обозначаемые A и B. В каждом ящике содержится по 12 шаров. В ящике A – 3 чёрных шара, 3 белых и 6 красных, а в ящике B – по 4 шара каждого цвета. Два опыта, соответственно A и B, состоят в извлечении шара из ящика A и ящика B. Какова неопределённость исходов?

$H_A = -\frac{3}{12}\log\frac{3}{12} - \frac{3}{12}\log\frac{3}{12} - \frac{6}{12}\log\frac{6}{12} = 1,5.$

$H_B = -3(\frac{4}{12}\log\frac{4}{12}) \approx 1,58.$

Неопределённость опыта B выше, и предсказать его труднее.

2.3 Совместная и условная энтропия

Пусть даны два ансамбля

$$A = \begin{pmatrix} a_1 & a_2 & ... & a_k \\ p_1 & p_2 & ... & p_k \end{pmatrix} \text{ и } B = \begin{pmatrix} b_1 & b_2 & ... & b_l \\ q_1 & q_2 & ... & q_l \end{pmatrix},$$

рассматриваемые как два источника информации. Вероятность события (a_i, b_j), заключающегося в том, что от источника A будет получен символ a_i, а от источника B – символ b_j, обозначим $p(a_i, b_j) = p_{i,j}$.

Пары $(a_i, b_1), (a_i, b_2), \ldots, (a_i, b_l)$ исчерпывают все случаи получения символа от источника B, когда от источника A получен символ a_i. События (a_i, b_j), $j = 1, ..., l$, не пересекаются, поэтому

$$\sum_{j=1}^{l} p(a_i, b_j) = p(a_i), \quad i = 1, ..., k,$$

$$\sum_{i=1}^{k} p(a_i, b_j) = p(b_j), \quad j = 1, ..., l.$$

Совместная энтропия двух источников

$$H(AB) = -\sum_i \sum_j p(a_i, b_j) \log p(a_i, b_j) = -\sum_i \sum_j p_{i,j} \log p_{i,j}.$$

Если источники информации A и B статистически независимы, то

$$p(a_i, b_j) = p(a_i)p(b_j),$$

иначе, если эти источники зависимы, то

$$p(a_i, b_j) = p(a_i)p(b_j/a_i) = p(b_j)p(a_i/b_j).$$

Если известно, что от источника A получен символ a_i, то вероятность получения символа b_j от зависимого от A источника B определяется условной вероятностью $p(b_j/a_i)$.

Величина

$$H(B/a_i) = -\sum_{j=1}^{l} p(b_j/a_i) \log p(b_j/a_i)$$

называется *условной энтропией источника B при условии получения символа a_i* от источника A.

Математическое ожидание величины $H(B/a_i)$ – среднюю условную энтропию

$$H_A(B) = \sum_{i=1}^{k} p(a_i) H(B/a_i)$$

называют *условной энтропией источника B* (при условии получения информации от источника A).

Условная энтропия обладает следующими свойствами.

1^o $H_A(B) \geq 0$, причем $H_A(B) = 0$ в том и только в том случае, когда получение информации от источника A однозначно определяет получаемую информацию от источника B.

Доказательство. Неотрицательность условной энтропии очевидна. Если $H_A(B) = \sum_{i=1}^{k} p(a_i)H(B/a_i) = 0$, то, поскольку все члены суммы неотрицательны, для всех i имеем $p(a_i)H(B/a_i) = 0$. Если при этом условии от источника A получен символ a_i, то $p(a_i) > 0$, следовательно,

$$H(B/a_i) = \sum_{j=1}^{l} p(b_j/a_i) \log p(b_j/a_i) = 0.$$

Но если при этом получен символ b_j от источника B, то $p(b_j/a_i) > 0$, и сумма $H(B/a_i)$ может быть равна нулю только тогда, когда $p(b_j/a_i) = 1$, что означает, что информация a_i от источника A однозначно определяет получение информации b_j от источника B.

2^o Для любых ансамблей – источников A и B выполняется

$$H(AB) = H(A) + H_A(B).$$

Доказательство.

$$H(AB) = -\sum_{i,j} p(a_i, b_j) \log p(a_i, b_j)$$

$$= -\sum_{i,j} p(a_i) p(b_j/a_i) \log p(a_i) p(b_j/a_i)$$

$$= -\sum_{i,j} p(a_i) p(b_j/a_i) \big(\log p(a_i) + \log p(b_j/a_i) \big)$$

$$= \underbrace{-\sum_{i,j} p(a_i) p(b_j/a_i) \log p(a_i)}_{S_1} \underbrace{-\sum_{i,j} p(a_i) p(b_j/a_i) \log p(b_j/a_i)}_{S_2}.$$

$$S_1 = -\sum_i \sum_j p(a_i) p(b_j/a_i) \log p(a_i)$$

$$= -\sum_i p(a_i) \log p(a_i) \underbrace{\sum_j p(b_j/a_i)}_{1} = H(A).$$

$$S_2 = -\sum_i \sum_j p(a_i) p(b_j/a_i) \log p(b_j/a_i)$$

$$= -\sum_i p(a_i) \sum_j p(b_j/a_i) \log p(b_j/a_i)$$

$$= -\sum_i p(a_i) H(B/a_i) = H_A(B).$$

3^o Для любых вещественных $q_1, ..., q_k$: $q_i \geq 0$, $i = 1, ..., k$, $q_1 + ... + q_k = 1$ имеет место неравенство

$$-\sum_{i=1}^{k} p_i \log q_i \geq -\sum_{i=1}^{k} p_i \log p_i,$$

причём равенство достигается при $q_i = p_i$ для всех i.

Доказательство. Пусть все $p_i > 0$. Логарифм – выпуклая вверх функция, поэтому

$$\sum_{i=1}^{k} \alpha_i \log x_i \le \log\Big(\sum_{i=1}^{k} \alpha_i x_i\Big)$$

для произвольных x_i и неотрицательных $\alpha_i : \alpha_1 + ...\alpha_k = 1$.

Взяв $\alpha_i = p_i$ и $x_i = \frac{q_i}{p_i}$ получаем

$$\sum_i p_i \log \frac{q_i}{p_i} \le \log \underbrace{\Big(\sum_i p_i \frac{q_i}{p_i}\Big)}_{1} = 0.$$

$$\sum_i p_i \log q_i - \sum_i p_i \log p_i \le 0; \quad -\sum_i p_i \log q_i \ge -\sum_i p_i \log p_i.$$

4^o Для любых двух ансамблей – источников информации A и B выполняется неравенство

$$H(AB) \le H(A) + H(B),$$

причём равенство имеет место в случае, когда источники A и B независимы.

Доказательство. Обозначим $p(a_i) = p_i;\ p(b_j) = q_j;\ p(a_i, b_j) = r_{ij}$. Тогда

$$p_i = \sum_j r_{ij}; \quad q_j = \sum_i r_{ij};$$

$$H(A) + H(B) = -\sum_i p_i \log p_i - \sum_j q_j \log q_j$$

$$= -\sum_{i,j} r_{ij} \log p_i - \sum_{i,j} r_{ij} \log q_j = -\sum_{i,j} r_{ij} \log(p_i \cdot q_j).$$

$$H(AB) = -\sum_{i,j} r_{ij} \log r_{ij}.$$

В силу свойства **3**^{o}, учитывая, что

$$\sum_i \sum_j p_i \cdot q_j = \sum_i p_i \Big(\sum_j q_j\Big) = 1 \quad \text{и}$$

$$\sum_i \sum_j r_{ij} = \sum_i \sum_j p(a_i, b_j) = \sum_i p(a_i) = 1,$$

получаем

$$H(A) + H(B) = -\sum_{i,j} r_{ij} \log(p_i \cdot q_j) \geq -\sum_{i,j} r_{ij} \log r_{ij} = H(AB).$$

Очевидно, равенство $H(AB) = H(A) + H(B)$ будет иметь место когда $r_{ij} = p_i \cdot q_j$, т.е. когда источники A и B независимы.

5^o Условная энтропия не превосходит безусловную:

$$H_A(B) \leq H(B),$$

причём $H_A(B) = H(B)$, когда источники A и B независимы.

Доказательство. По свойству **4**^{o}

$$H(AB) = H(A) + H_A(B) \leq H(A) + H(B).$$

Удаляя из левой и правой частей последнего неравенства $H(A)$, получаем $H_A(B) \leq H(B)$.

2.4 Информация и её свойства

Разность $H(B) - H_A(B)$ характеризует среднее уменьшение неопределённости источника информации B после получения информации от источника A. Поэтому эту разность определяют как *среднюю величину информации $I(A, B)$, получаемой об источнике B в результате получения информации от источника A*:

$$I(A, B) = H(B) - H_A(B).$$

Справедливы следующие свойства информации $I(A, B)$.

1. $I(A, B) \geq 0$, причём $I(A, B) = 0$, когда источники A и B независимы – не несут информации друг о друге.

2. $I(A, B) \leq H(B)$, причём равенство имеет место тогда, когда исход A однозначно определяет исход B.

Доказательство. $I(A, B) = H(B) - H_A(B) \leq H(B)$, поскольку $H_A(B) \geq 0$. Если же достигается равенство, то

$$H_A(B) = \sum_i p(a_i) H(B/a_i) = 0.$$

Пусть получен какой-либо символ a_i ансамбля A. Тогда он имеет ненулевую вероятность $p(a_i) > 0$. Но тогда

$$H(B/a_i) = -\sum_j p(b_j/a_i) \log p(b_j/a_i) = 0.$$

Поскольку задан ансамбль B, то $\sum_j p(b_j/a_i) > 0$, и тогда найдется j такое, что $p(b_j/a_i) > 0$. Тогда из условия $-\sum_j p(b_j/a_i) \log p(b_j/a_i) = 0$ следует, что $p(b_j/a_i) = 1$, что означает: получение символа a_i однозначно определяет получение символа b_j.

2. $I(B, B) = H(B)$.

Доказательство. $I(B, B) = H(B) - H_B(B)$;

$$H_B(B) = \sum_j p(b_j) H(B/b_j);$$

$$H(B/b_j) = -\sum_j p(b_j/b_j) \log \underbrace{p(b_j/b_j)}_{1} = 0.$$

3. $I(A, B) = I(B, A)$.

Доказательство. $I(A, B) = H(B) - H_A(B)$;

$$H(AB) = H(A) + H_A(B);$$

$$H_A(B) = H(AB) - H(A);$$

$$I(A, B) = H(B) - H_A(B) = H(B) - \big(H(AB) - H(A)\big)$$
$$= H(B) + H(A) - H(AB).$$

$$I(B, A) = H(A) - H_B(A) = H(A) - \big(H(BA) - H(B)\big)$$
$$= H(A) + H(B) - H(BA).$$

$$H(AB) = -\sum_{i,j} p(a_i b_j) \log p(a_i b_j) = H(BA).$$

4. Если A, B, C, D – источники информации (ансамбли), причём пара источников AC независима от пары BD, то

$$I(AB, CD) = I(A, C) + I(B, D).$$

Пояснение. Прежде чем доказывать свойство **4**, дадим его содержательную интерпретацию.

Пусть имеются два канала передачи информации. На входе первого канала – источник A; его выход можно рассматривать как источник информации C. Аналогично, на входе второго канала – источник B, и его выход можно рассматривать как источник информации D (рис. 2.6).

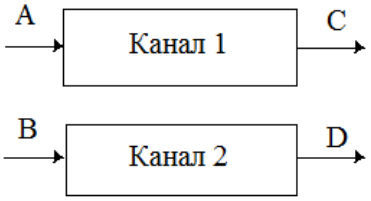


Рис. 2.6: Два независимых канала передачи информации

Тогда при передаче сообщений A и B по двум независимым каналам общая переданная информация равна сумме информации, полученной по первому каналу, и информации, полученной по второму каналу.

Доказательство. Если AC и BD независимы, то A и B, A и D, C и B, C и D – также независимы. Тогда

$$p(a_ib_j) = \sum_{u,v} p(a_ib_jc_ud_v)$$

$$= \sum_{u,v} p(a_ic_u) \cdot p(b_jd_v)$$

$$= \sum_{u} p(a_ic_u) \sum_{v} p(b_jd_v) = p(a_i) \cdot p(b_j).$$

$$I(AB, CD) = H(AB) + H(CD) - H(ABCD)$$

$$= H(A) + H(B) + H(C) + H(D) - H(AC) - H(BD)$$

$$= \underbrace{H(A) + H(C) - H(AC)}_{I(A,C)} + \underbrace{H(B) + H(D) - H(BD)}_{I(B,D)}$$

$$= I(A, C) + I(B, D).$$

5. Пусть имеется некоторый алфавитный оператор φ – преобразователь информации, действующий как показано на рис. 2.7.

Если на входе преобразователя φ появляется случайно выбранный символ источника информации – ансамбля A, то на его выходе

будет случайный символ ансамбля $\varphi(A) = A'$, причём каждому символу a_i ансамбля A соответствует единственный символ ансамбля A'.

Если, например, $\varphi(a_1) = \varphi(a_2) = \varphi(a_3) = a_1'$, то $p(a_1') = p(a_1)+p(a_2)+p(a_3)$[1], и в таком случае будем говорить, что преобразователь φ *склеивает* символы a_1, a_2, a_3 в символ a_1'.

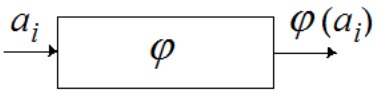


Рис. 2.7: Алфавитный оператор φ

При сформулированных условиях имеет место неравенство

$$I(\varphi(A), B) \leq I(A, B).$$

Доказательство. Пусть φ склеивает a_1 и a_2 в a'. Обозначим $p(a_i) = p_i$; $p(a_i b_j) = r_{ij}$. В этих обозначениях $p(a') = p_1 + p_2$; $p(a' b_j) = r_{1j} + r_{2j}$.

$$I(A, B) = H(B) - H_A(B);$$

$$I(\varphi(A), B) = H(B) - H_{\varphi(A)}(B),$$

поэтому достаточно установить, что $H_A(B) \leq H_{\varphi(A)}(B)$.

$$H_A(B) = \sum_i p(a_i) H(B/a_i);$$

$$H_{\varphi(A)}(B) = \underbrace{p(a')H(B/a')}_{a_1 \text{и} \ \ a_2 \ \text{склеиваются в} \ a'} + \sum_{i \geq 3} p(a_i) H(B/a_i).$$

Теперь достаточно установить, что

$$p(a_1)H(B/a_1) + p(a_2)H(B/a_2) \leq p(a')H(B/a'). \qquad (1)$$

$$H(B/a_i) = -\sum_j p(b_j/a_i) \log p(b_j/a_i) = -\sum_j \frac{r_{ij}}{p_i} \log \frac{r_{ij}}{p_i}$$

(считаем что в ансамбле A таких символов a_i, для которых $p_i = 0$, нет: в противном случае их можно было удалить из ансамбля заранее, как непоявляющиеся).

$$H(B/a') = -\sum_j \frac{r_{1j} + r_{2j}}{p_1 + p_2} \log \frac{r_{1j} + r_{2j}}{p_1 + p_2}.$$

[1]События, соответствующие появлению разных символов на входе преобразователя, несовместны

Перепишем неравенство (1), которое требуется доказать, в виде

$$-p_1 \sum_j \frac{r_{1j}}{p_1} \log \frac{r_{1j}}{p_1} - p_2 \sum_j \frac{r_{2j}}{p_2} \log \frac{r_{2j}}{p_2} \le -(p_1+p_2) \sum_j \frac{r_{1j}+r_{2j}}{p_1+p_2} \log \frac{r_{1j}+r_{2j}}{p_1+p_2}. \quad (2)$$

Рассматривая выпуклую вверх функцию $-x \log x$ в двух точках: $x_1 = \frac{r_{1j}}{p_1}$ и $x_2 = \frac{r_{2j}}{p_2}$ с коэффициентами $\alpha_1 = \frac{p_1}{p_1+p_2}$ и $\alpha_2 = \frac{p_2}{p_1+p_2}$, имеем

$$\frac{p_1}{p_1+p_2}\Big(-\frac{r_{1j}}{p_1} \log \frac{r_{1j}}{p_1}\Big) + \frac{p_2}{p_1+p_2}\Big(-\frac{r_{2j}}{p_2} \log \frac{r_{2j}}{p_2}\Big) \le -\frac{r_{1j}+r_{2j}}{p_1+p_2} \log \frac{r_{1j}+r_{2j}}{p_1+p_2}.$$

Домножая обе части последнего неравенства на p_1+p_2 и суммируя по j, убеждаемся в справедливости (2).

2.5 Энтропия непрерывной информации

Формула для энтропии непрерывного источника информации, являющегося случайной величиной X с плотностью распределения $p(x)$ на $\mathbb{R}$, *определяется* по аналогии с формулой энтропии дискретного источника следующим образом:

$$H(X) = -\int_{-\infty}^{\infty} p(x) \log p(x) dx. \quad (3)$$

1^o Если плотность распределения $p(x)$ равна нулю всюду, кроме промежутка $[a,b]$, то максимум энтропии, равный $\log(a-b)$ достигается для случая равномерного распределения на отрезке $[a,b]$.

Для доказательства понадобится известное неравенство $\ln x \le x-1$, которое с использованием логарифма по основанию 2 будет иметь вид

$$\log x \le (x-1) \log e. \quad (4)$$

Пусть $p(x)$ и $q(x)$ – два произвольных одномерных распределения, равных нулю всюду, кроме промежутка $[a,b]$, на котором эти плотности больше нуля; $\int_a^b q(x)dx = \int_a^b p(x)dx = 1$. Покажем, что

$$\int_a^b p(x) \log \frac{q(x)}{p(x)} dx \le 0, \quad (5)$$

причем равенство достигается при $q(x) = p(x)$.

$$\int_a^b p(x)\log\frac{q(x)}{p(x)}dx \le \int_a^b p(x)(\frac{q(x)}{p(x)}-1)\log e\ dx =$$

$$= \log e\int_a^b p(x)(\frac{q(x)}{p(x)}-1)dx = \log e\Big(\int_a^b q(x)dx - \int_a^b p(x)dx\Big) = 0.$$

Подставим в неравенство (5) равномерную плотность распределения $q(x) = \frac{1}{b-a}$, равную нулю всюду вне промежутка $[a,b]$:

$$\int_a^b p(x)\log\frac{1/(b-a)}{p(x)}dx \le 0. \tag{6}$$

$$\log\frac{1}{b-a}\int_a^b p(x)dx + H(X) \le 0;$$

$$H(X) \le \log(b-a).$$

Вычисляя энтропию равномерного на $[a,b]$ распределения (из условия нормировки в этом промежутке $p(x) = \frac{1}{b-a}$), имеем

$$H_{unif}\,(X) = -\int_a^b \frac{1}{b-a}\log\frac{1}{b-a}dx = \log(b-a).$$

2^o При условии ограниченности дисперсии, максимальной энтропией, равной

$$\log\sigma\sqrt{2\pi e} \tag{7}$$

обладает случайная величина X, имеющая нормальное распределение, плотность которого есть

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\bar{x})^2}{2\sigma^2}}. \tag{8}$$

Доказательство. Покажем, что максимум функционала

$$H(X) = -\int_{-\infty}^{\infty} p(x)\log p(x)dx \tag{9}$$

при ограничениях

$$\int_{-\infty}^{\infty} p(x)dx = 1 \quad \text{и} \quad \int_{-\infty}^{\infty}(x-\bar{x})^2 p(x)dx = \sigma^2 \tag{10}$$

достигается, когда плотность распределения $p(x)$ имеет вид (8). Для этой цели решаем задачу Лагранжа вариационного исчисления на минимум функционала [2]

$$-H(X) = \int_{-\infty}^{\infty} p(x) \log p(x) dx \tag{11}$$

при ограничениях (10). Составляем функцию Лагранжа

$$F(p, \lambda, \mu) = p \log p + \lambda p + \mu (x - \bar{x})^2 p; \quad \frac{\delta F}{\delta p} = 0.$$

Решаем уравнение Эйлера

$$\log p + \log e + \lambda + \mu (x - \bar{x})^2 = 0; \quad p \cdot e = e^{-\lambda - \mu (x - \bar{x})^2};$$

$$p(x) = e^{-1-\lambda-\mu(x-\bar{x})^2};$$

$$p(x) = e^{-\mu(x-\bar{x})^2} / e^{1+\lambda}. \tag{12}$$

Далее понадобятся следующие два интеграла:

$$\int_{-\infty}^{\infty} e^{-\mu(x-a)^2} dx = \sqrt{\frac{\pi}{\mu}}; \tag{13}$$

$$\int_{-\infty}^{\infty} (x-a)^2 e^{-\mu(x-a)^2} dx = \frac{1}{2\mu} \sqrt{\frac{\pi}{\mu}}; \tag{14}$$

Интеграл (13) получается из условия нормировки нормального распределения. Положив $\mu = \frac{1}{2\sigma_*^2}$; $\sigma_* = \frac{1}{\sqrt{2\mu}}$, получим

$$\frac{1}{\sqrt{2\pi}\sigma_*} \int_{-\infty}^{\infty} e^{-\frac{(x-a)^2}{2\sigma_*^2}} dx = 1;$$

$$\int_{-\infty}^{\infty} e^{-\mu(x-a)^2} dx = \sqrt{2\pi}\sigma_* = \sqrt{2\pi} \frac{1}{\sqrt{2\mu}} = \sqrt{\frac{\pi}{\mu}}.$$

Интеграл (14) получается путем сведения к дисперсии нормального распределения:

$$\frac{1}{\sqrt{2\pi}\sigma_*} \int_{-\infty}^{\infty} (x-a)^2 e^{-\mu(x-a)^2} dx = \sigma_*^2;$$

$$\int_{-\infty}^{\infty} (x-a)^2 e^{-\mu(x-a)^2} dx = \sigma_*^2 \sqrt{2\pi}\sigma_* = \frac{1}{2\mu} \sqrt{2\pi} \frac{1}{\sqrt{2\mu}} = \frac{1}{2\mu} \sqrt{\frac{\pi}{\mu}}.$$

[2] энтропия как функционал, определённый в пространстве функций распределения, обладает свойством выпуклости [10, с. 615]

Согласно условию нормировки

$$\int_{-\infty}^{\infty} e^{-1-\lambda-\mu(x-\bar{x})^2}dx = 1; \quad \int_{-\infty}^{\infty} e^{-\mu(x-\bar{x})^2}dx = e^{1+\lambda}; \quad \sqrt{\frac{\pi}{\mu}} = e^{1+\lambda}.$$

Подставляя в (12) вместо $e^{1+\lambda}$ равное значение $\sqrt{\frac{\pi}{\mu}}$, получаем

$$p(x) = \sqrt{\frac{\mu}{\pi}}\, e^{-\mu(x-\bar{x})^2}. \tag{15}$$

Учитывая второе ограничение, получаем

$$\int_{-\infty}^{\infty} (x-\bar{x})^2 \sqrt{\frac{\mu}{\pi}}\, e^{-\mu(x-\bar{x})^2}dx = \sigma^2;$$

$$\int_{-\infty}^{\infty} (x-\bar{x})^2\, e^{-\mu(x-\bar{x})^2}dx = \sigma^2\sqrt{\frac{\pi}{\mu}} = \sigma^2\sqrt{2\pi\sigma^2}; \tag{16}$$

$$\frac{1}{2\mu}\sqrt{\frac{\pi}{\mu}} = \sigma^2\sqrt{\frac{\pi}{\mu}}; \quad \mu = \frac{1}{2\sigma^2}; \quad \sqrt{\mu} = \frac{1}{\sqrt{2}\sigma}.$$

Подставляя найденные выражения для $\sqrt{\mu}$ и μ в (15), получаем

$$p(x) = \sqrt{\frac{\mu}{\pi}}\, e^{-\mu(x-\bar{x})^2} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}}.$$

Подставляя найденную экстремаль $\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}}$ в функционал (11), получаем

$$-H_{extr}(X) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}} \log\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}}\right) dx$$

$$= \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}} \left(\log\frac{1}{\sqrt{2\pi}\sigma} - \frac{(x-\bar{x})^2}{2\sigma^2}\log e\right) dx$$

$$= \frac{1}{\sqrt{2\pi}\sigma}\log\frac{1}{\sqrt{2\pi}\sigma}\sqrt{2\pi\sigma^2} - \frac{1}{\sqrt{2\pi}\sigma}\cdot\frac{\log e}{2\sigma^2}\int_{-\infty}^{\infty}(x-\bar{x})^2 e^{-\frac{(x-\bar{x})^2}{2\sigma^2}}dx$$

$$= \log\frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sqrt{2\pi}\sigma}\cdot\frac{\log e}{2\sigma^2}\cdot\frac{2\sigma^2}{2}\sqrt{2\pi\sigma^2} \quad \text{(см. 16)}$$

$$= \log\frac{1}{\sqrt{2\pi}\sigma} - \frac{\log e}{2} = \log\frac{1}{\sigma\sqrt{2\pi e}}.$$

$$H_{extr}(X) = \log\sigma\sqrt{2\pi e}.$$

3 Передача дискретной информации, пропускная способность канала и теорема Шеннона

3.1 Скорость передачи и пропускная способность канала

Скорость передачи информации (и скорость её порождения источником) может измеряться разными способами, например, числом символов, передаваемых в единицу времени. Если каждый символ передаётся за τ секунд, то скорость передачи $R = \frac{1}{\tau}$ (*бод*), 1 *бод* $= \frac{1 \ \ символ}{1 \ \ сек}$. Эта единица измерения связана с именем изобретателя телеграфа *Ж. Бодо*.

Скорость передачи двоичных символов измеряется в битах, килобитах, мегабитах в секунду и т.д. Если передаются двоичные блоки равной длины по n бит, и каждый – за T секунд, то скорость передачи $R = \frac{n}{T}$ *бит/сек*. Такую передачу называют *блоковой*. Блок может содержать в себе информацию об одном символе или иметь некоторое другое смысловое значение.

Ниже, при рассмотрении линейных двоичных (n, k) кодов, скорость передачи информации определяется как $R = \frac{k}{n}$ (*бит/на время одного такта*), где n – длина кодового блока в битах, а k – число информационных разрядов, содержащихся среди n разрядов передаваемой информации. Естественно полагать, что в этом случае k информационных бит передаются за n тактов (единиц) времени.

Если на входе канала передачи данных имеется источник информации, который характеризуется ансамблем X, а на его выходе – ансамблем Y, то количество передаваемой информации в *среднем на один символ* определяется по формуле

$$I(Y, X) = H(Y) - H_X(Y) \quad (\textit{бит/символ}).$$

Усреднение на символ объясняется тем, как вычисляется энтропия: при её вычислении происходит усреднение информации в битах, соответствующих символам, по всем символам ансамбля. Если при

этом среднее время передачи одного символа равно T, то скорость передачи информации измеряется по формуле

$$R = \frac{I(Y,X)}{T} \quad (\frac{бит}{символ}) : (\frac{сек}{символ}) = (\frac{бит}{сек}).$$

Определение 3.1. *Пропускной способностью канала называется величина*

$$C = \max I(Y,X) = \max(H(Y) - H_X(Y)),$$

характеризующая максимальное количество информации, которое может быть передано по каналу в среднем на один символ (за такт), или

$$C = \frac{\max I(Y,X)}{T} = \frac{\max(H(Y) - H_X(Y))}{T},$$

характеризующая максимальное среднее число бит передаваемой информации в секунду. □

Согласно приведенному определению, пропускная способность является величиной, ограничивающей скорость передачи информации по каналу.

Рассмотрим простейший случай двоичного симметричного канала: входной источник является ансамблем из двух символов – 0 и 1 с одинаковыми вероятностями, равными $\frac{1}{2}$; вероятность искажения любого из этих двух символов (т.е. инвертирования) после передачи по каналу равна $\mathfrak{p}$, а вероятность правильной передачи символа равна $\mathfrak{q} = 1 - \mathfrak{p}$. При такой модели канала при передаче возможны ошибки (говорят: канал с ошибками).

$$I(Y,X) = H(Y) - H_X(Y);$$

$$H_X(Y) = p(X=1)H(Y/X=1) + p(X=0)H(Y/X=0);$$
$$H(Y/X=1) = -p(Y=1/X=1)\log p(Y=1/X=1) -$$
$$-p(Y=0/X=1)\log p(Y=0/X=1);$$
$$H(Y/X=0) = -p(Y=1/X=0)\log p(Y=1/X=0) -$$

$$-p(Y=0/X=0)\log p(Y=0/X=0).$$

Учтем, что

$$p(Y=1/X=0)=p(Y=1/X=0)=\mathfrak{p},$$

$$p(Y=1/X=1)=p(Y=0/X=0)=\mathfrak{q},$$

и тогда

$$H(Y/X=1)=-\mathfrak{q}\log\mathfrak{q}-\mathfrak{p}\log\mathfrak{p}=H(\mathfrak{p}),$$

$$H(Y/X=0)=-\mathfrak{p}\log\mathfrak{p}-\mathfrak{q}\log\mathfrak{q}=H(\mathfrak{p}).$$

$$H_X(Y)=p(X=1)H(\mathfrak{p})+p(X=0)H(\mathfrak{p})=\frac{1}{2}H(\mathfrak{p})+\frac{1}{2}H(\mathfrak{p})=H(p).$$

$$I(Y,X)=H(Y)-H(\mathfrak{p}).$$

Учитывая, что случайная величина на входе принимает только два значения, имеем $H(Y)\le\log 2=1$, откуда следует, что

$$C=\max I(Y,X)\le 1-H(\mathfrak{p}).$$

Здесь $\log 2$ – максимальное возможное значение энтропии $H(Y)$ двухсимвольного ансамбля в случае равновероятного появления символов, поскольку *максимум передаваемой информации в среднем на символ определяется по всевозможным значениям вероятностей появления символов* 1 и 0 – $p(X=1)$ и $p(X=0)=1-p(X=1)$.

Если показать, что при этом также выполняется и равенство

$$C=\max I(Y,X)\ge 1-H(\mathfrak{p}),$$

то будет доказано, что

$$C=1-H(\mathfrak{p}).$$

Для этой цели укажем следующий входной источник рассматриваемого типа: пусть X_0 равновероятно принимает значения 0 и 1 (с вероятностью $\frac{1}{2}$ каждое из этих значений), ошибка при передаче двоичного символа в канале происходит с вероятностью $\mathfrak{p}$, а правильная

передача – с вероятностью $\mathfrak{q} = 1 - \mathfrak{p}$. Обозначим соответствующий указанному входному источнику выход канала Y_0.

$$p(Y_0 = 1) = p(X_0 = 1)\mathfrak{q} + p(X_0 = 0)\mathfrak{p} = \frac{1}{2}(\mathfrak{q} + \mathfrak{p}) = \frac{1}{2}.$$

Аналогично,

$$p(Y_0 = 0) = \frac{1}{2}.$$

Поэтому

$$H(Y_0) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} = 1,$$

и указан вход X_0, для которого $I(Y_0, X_0) = 1 - H(\mathfrak{p})$, поэтому

$$C = \max_{0 \le p(X=1) \le 1} I(Y, X) \ge 1 - H(\mathfrak{p}),$$

и окончательно –

$$C = 1 - H(\mathfrak{p}). \tag{17}$$

Максимальная пропускная способность, равная единице, достигается в двух случаях: при $\mathfrak{p} = 0$ и $\mathfrak{p} = 1$. Если $\mathfrak{p} = 0$, то ошибок в канале нет, и выходная информация тождественна исходной; если $\mathfrak{p} = 1$, то всегда принимается инверсный символ, что также позволяет получать на выходе канала информацию без ошибки. При $\mathfrak{p} = \frac{1}{2}$ принятый на выходе канала символ равновероятно может быть искаженной единицей и искаженным нулём; правильное восстановление невозможно, и пропускная способность в этом случае равна нулю.

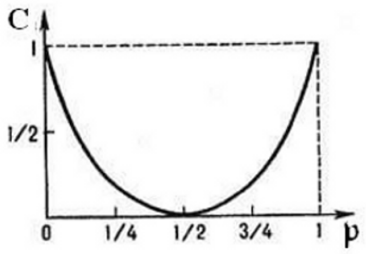


Рис. 3.8: Пропускная способность $C = C(\mathfrak{p})$ двоичного канала с ошибками

В результате можно предположить, что в общем случае пропускная способность канала тем выше, чем меньше вероятность ошибки при передаче информации в канале.

На практике ограничение пропускной способности обычно возникает вследствие физических особенностей канала, например, его

частотных свойств. Наилучшее использование канала предполагает, что информация передаётся со скоростью, равной его пропускной способности. Ответ на вопрос, насколько это равенство возможно, получил К. Шеннон, доказавший, что

сообщение, подлежащее передаче по каналу, можно закодировать таким образом, что будет возможно передавать символы со средней скоростью, сколь угодно близкой к пропускной способности. Но надежная передача информации со скоростью, превышающей пропускную способность канала, невозможна [10].

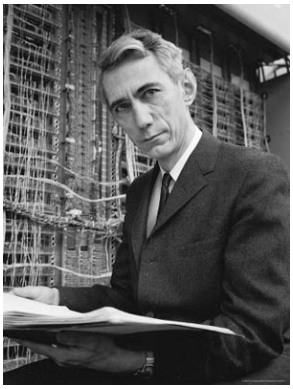


Рис. 3.9: Клод Шеннон, 1916–2001 гг., выдающийся американский инженер и математик

Это утверждение Клода Шеннона справедливо во многих случаях при различных рассматриваемых источниках информации и для различных каналов, что составляет содержание ряда теорем, называемых теоремами Шеннона.

Главную роль для достижения наибольшей скорости передачи символьной информации, как будет показано ниже, играет способ кодирования передаваемых по каналу символов. Поэтому математическая теория передачи информации тесно связана с теорией кодирования.

3.2 Основная теорема Шеннона о кодировании

Теорема 3.1. *При любом заданном конечном ансамбле источника X с энтропией $H(X)$ и для любого варианта кодирования букв алфавита двоичными кодовыми словами, обеспечивающего однозначное декодирование,* [3] *средняя длина кодового слова $\bar{l}$ будет удовлетворять неравенству*

$$\bar{l} \geq H(X). \tag{18}$$

Доказательство. Пусть $p_1, ..., p_i, ...p_k$ – вероятности символов

[3] О кодировании и декодировании см. Приложение А, стр. 204

$x_1, ..., x_i, ..., x_k$ ансамбля X, а $l_1, ..., l_i, ..., l_k$ – длины двоичных кодовых слов этих символов. Покажем, что

$$H(X) - \bar{l} \leq 0.$$

$$H(X) - \bar{l} = \sum_{i=1}^{k} p_i \log \frac{1}{p_i} - \sum_{i=1}^{k} p_i l_i;$$

$$H(X) - \bar{l} = \sum_{i=1}^{k} p_i \log \frac{1}{p_i} - \sum_{i=1}^{k} p_i \log 2^{l_i};$$

$$H(X) - \bar{l} = \sum_{i=1}^{k} p_i (\log \frac{1}{p_i} - \log 2^{l_i}); \quad H(X) - \bar{l} = \sum_{i=1}^{k} p_i \log \frac{2^{-l_i}}{p_i}.$$

Применяя легко проверяемое неравенство $\log z \leq (z-1)\log e$ (кривая логарифма лежит ниже линии прямой $(z-1)\log e$, имея с ней только одну общую точку в единице), при $z = \frac{2^{-l_i}}{p_i} > 0$, получаем

$$H(X) - \bar{l} \leq \log e \sum_{i=1}^{k} p_i \left(\frac{2^{-l_i}}{p_i} - 1 \right);$$

$$H(X) - \bar{l} \leq \log e \left(\sum_{i=1}^{k} 2^{-l_i} - \sum_{i=1}^{k} p_i \right);$$

$$H(X) - \bar{l} \leq \log e \left(\sum_{i=1}^{k} 2^{-l_i} - 1 \right).$$

Используя неравенство Макмиллана (см. приложение А, стр. 204) для однозначно декодируемых кодов, получаем

$$\sum_{i=1}^{k} 2^{-l_i} \leq 1; \quad \sum_{i=1}^{k} 2^{-l_i} - 1 \leq 0,$$

и окончательно –

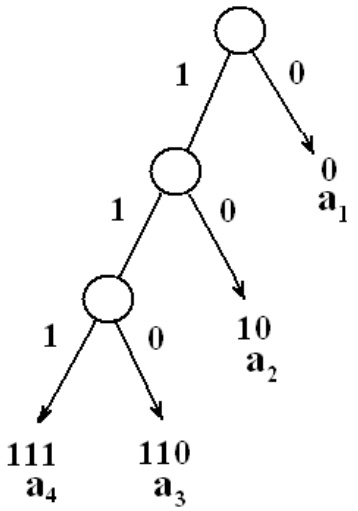
$$H(X) - \bar{l} \leq 0; \quad \bar{l} \geq H(X). \quad \square$$

Итак, установлено, что *средняя длина двоичных кодов букв однозначно декодируемого кода не превышает энтропии ансамбля источника информации. Следующий пример показывает: возможно такое кодирование, что средняя длина кодов букв будет равна энтропии ансамбля источника.*

Пример. Пусть задан следующий ансамбль четырехбуквенного источника

$$\begin{pmatrix} a_1 & a_2 & a_3 & a_4 \\ p_1 = 1/2 & p_2 = 1/4 & p_3 = 1/8 & p_3 = 1/8 \end{pmatrix}.$$

Дерево кодирования по методу Хаффмана [4] букв источника, таблица вероятностей и построенных кодов представлена на рис. 3.10



Буква	Вероятность	Код
a_1	$p_1 = 1/2$	0
a_2	$p_2 = 1/4$	10
a_3	$p_3 = 1/8$	110
a_4	$p_4 = 1/8$	111

Рис. 3.10: Построение кода Хаффмана

Средняя длина кода равна

$$\bar{l} = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + 3 \cdot \frac{1}{8} = 1.75.$$

Энтропия ансамбля равна такой же величине

$$H(X) = \frac{1}{2}\log 2 + \frac{1}{4}\log 4 + \frac{1}{8}\log 8 + \frac{1}{8}\log 8 = 1.75.$$

[4] О коде Хаффмана см. в приложении А на стр. 210

Приведенный пример позволяет сделать вывод, что *минимальная средняя длина кода символов при алфавитном кодировании может быть равна энтропии источника информации.*

Теорема 3.2 (Основная теорема Шеннона для канала без шума)**.** *Пусть источник имеет энтропию H бит на символ, а канал имеет пропускную способность C бит в секунду. Тогда сообщения на выходе источника можно закодировать таким образом, чтобы передавать символы по каналу со средней скоростью $\frac{C}{H}-\varepsilon$ символов в одну секунду, где ε – сколь угодно мало. Но передача со средней скоростью, большей $\frac{C}{H}$, невозможна.*

Доказательство. Выше (теорема 3.1) доказана достижимая нижняя оценка средней длины кода символа $\bar{l} \geq H$. Если канал имеет пропускную способность C бит в секунду, то при минимальной средней длине символа, равной H, максимальная средняя скорость передачи будет равна $\frac{C}{\bar{l}_{min}} = \frac{C}{H}$ $(\frac{бит}{сек})$: $(\frac{бит}{символ})$ = $(\frac{символ}{сек})$, и превысить её, действительно, невозможно.

Рассмотрим все сообщения источника, состоящие из N символов, идущих подряд. Длина таких сообщений определяется способом кодирования символов алфавита – сколько бит будет отведено на каждый его символ. Пусть число таких возможных различных сообщений, составленных из N символов, равно n. Будем считать, что каждое из указанных сообщений имеет ненулевую вероятность появления; в противном случае сообщение с нулевой вероятностью просто не вошло бы в число n сообщений.

Расположим все эти n сообщений по невозрастанию вероятностей их появления, введя соответствующую нумерацию и, не теряя общности, считая, что $p_1 \geq p_2 \geq ... \geq p_s ... \geq p_n$. Пусть $P_1 = 0$; $P_s = \sum_{i=1}^{s-1} p_i$ – сумма вероятностей от p_1 до p_{s-1} включительно, $2 \leq s \leq n$.

Двоичный код сообщения с номером $s = 1$ положим равным нулю и имеющим длину 1; двоичный код сообщения с номером $s \geq 2$ построим путём разложения P_s как двоичного числа до разряда с номером m_s, где m_s удовлетворяет соотношениям

$$\log \frac{1}{p_s} \leq m_s < 1 + \log \frac{1}{p_s}. \tag{19}$$

Эти неравенства можно преобразовать так:

$$\frac{1}{2^{m_s}} \leq p_s < \frac{1}{2^{m_s-1}} \tag{20}$$

и убедиться, что высоковероятные сообщения будут представляться короткими кодами, а маловероятные – длинными.

Итак, длины m_s кодов сообщений определены. Двоичное слово, соответствующее коду s-той последовательности, получим из представления P_s как двоичного числа. Заметим, что $P_s < 1$, и его двоичное представление имеет вид

$$P_s = 0, d_1 d_2 d_3 ... d_{m_s} ... \ , \tag{21}$$

где $d_i,\ i = 1, 2, ...$ означает одну из двоичных цифр 0 или 1. В качестве кода сообщения будем брать m_s первых двоичных цифр после запятой двоичного представления (21).

Пусть код первого по порядку сообщения будет $C_1 = 0$, код второго сообщения будет $C_2 = 1\alpha$, где α – двоичная цифра 0 или 1, а код s-го сообщения, $s \geq 3$, будет

$$C_s = d_1 d_2 ... d_{m_s}.$$

Код следующего сообщения C_{s+1} будет иметь длину $m_s + 1$ и будет получаться путём добавления к двоичному числу P_s значения вероятности p_s в двоичном представлении:

$$\begin{aligned}
P_s = &\quad 0, d_1 d_2 d_3 ... d_{m_{s-1}} d_{m_s} d_{m_{s+1}} ... \\
&+ \\
p_s = &\quad 0, 0\ 0\ 0\ ...\ 0 \qquad 1 \quad b_{m_{s+1}} ... \\
&= \\
P_{s+1} = &\ 0, h_1 h_2 h_3 ... h_{m_{s-1}} h_{m_s} h_{m_{s+1}} ...,
\end{aligned}$$

где $d_i,\ b_i, h_i,\ i = 1, 2, ...,$ обозначают произвольную двоичную цифру.

Вид второго слагаемого объясняется тем, что в силу неравенств (20), в двоичном представлении вероятности p_s в дробной части в разряде с номером m_s должна быть единица, т. к. $p_s \geq 2^{-m_s}$.

В итоге код s-го сообщения

$$C_s = d_1 d_2 ... d_{m_s}$$

не будет являться префиксом кода $s+1$-го сообщения

$$C_{s+1} = h_1 h_2 ... h_{m_s} h_{m_{s+1}},$$

поскольку в результате приведенного выше сложения двоичные слова $d_1 d_2 ... d_{m_s}$ и $h_1 h_2 ... h_{m_s}$ не могут совпадать.

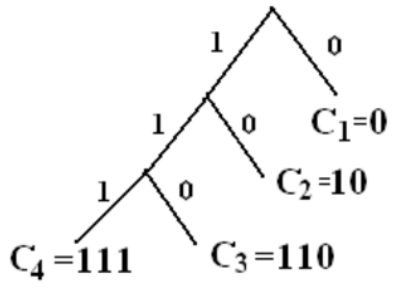


Рис. 3.11: Пример построения кодов сообщений

Тогда, используя то, что каждый предыдущий код C_s не является префиксом последующего кода C_{s+1}, а длины m_s кодов последовательно возрастают, можно сопоставить указанному выше процессу последовательного построения кодов сообщений растущее бинарное дерево (например, как показано на рисунке 3.11), что показывает однозначность декодирования[5].

В результате построенного кодирования среднее число бит на символ исходного сообщения можно оценить как

$$\bar{L} = \frac{1}{N} \sum_{s=1}^{n} m_s p_s.$$

Умножая неравенства (19) на p_s, суммируя по s и деля их на N, получаем

$$\frac{1}{N} \sum_{s=1}^{n} p_s (\log \frac{1}{p_s}) \leq \frac{1}{N} \sum_{s=1}^{n} m_s p_s < \frac{1}{N} \sum_{s=1}^{n} p_s (1 + \log \frac{1}{p_s}). \qquad (22)$$

В левой части цепочки неравенств записана энтропия исходного источника сообщений на символ (напомним, что рассматриваются сообщения, состоящие в точности из N символов, а энтропия обладает свойством аддитивности):

$$\bar{H} = \frac{1}{N} \sum_{s=1}^{n} p_s (\log \frac{1}{p_s}),$$

[5]см. приложение А на стр. 205

поэтому из (22) получаем

$$\bar{H} \le \bar{L} < \bar{H} + \frac{1}{N}.$$

С ростом N величина $\varepsilon_1 = \varepsilon_1(N) = \frac{1}{N}$ может стать сколь угодно малой, $\bar{L} \to \bar{H},\ \varepsilon_1 \to 0$ при $N \to \infty$, и можно обеспечить среднюю длину символов передаваемого сообщения лишь на ε_1 больше энтропии исходного сообщения (на символ). И тогда будет достигнута скорость передачи

$$\frac{C}{\bar{H} + \varepsilon_1} = \frac{C}{\bar{H}} - \varepsilon \left(\frac{\textit{символов}}{\textit{сек}} \right),$$

где

$$\varepsilon = \frac{C\varepsilon_1}{\bar{H}^2 + \bar{H}\varepsilon_1} \to 0 \textit{ при } N \to \infty \textit{ и } \varepsilon_1 \to 0. \ \square$$

Далее рассмотрим блоковую передачу двоичных данных, полагая, что каждый передаваемый блок информации содержит n разрядов, из которых k – информационных и $(n-k)$ – контрольных. Контрольные разряды вычисляются как суммы по модулю два информационных разрядов по определённым правилам, например, как в кодах Хэмминга. В общем случае такие коды называют линейными (n,k)-кодами.

Теорема 3.3 (Шеннона о блоковом кодировании)**.** *Для любого числа R, меньшего пропускной способности канала C, и любого $\varepsilon > 0$ существует способ блоковой передачи со скоростью не меньшей R и вероятностью ошибки, не превосходящей ε.*

Под ошибкой понимается событие, состоящее в том, что переданный блок информации не совпадает с блоком, принятым на выходе канала.

Доказательство теоремы 3.3 приведено в приложении В, см. стр. 212 .

И теорема 3.3, и её доказательство не дают конструктивного способа построения блокового линейного кода, обеспечивающего передачу информации со скоростью R, близкой к пропускной способности канала C, а лишь указывают на существование такого способа

кодирования. При этом скорость передачи не превышает пропускную способность: $R < C$.

При доказательстве теоремы 3.3 полагается, что скорость передачи $R = \frac{k}{n}$ ($\frac{бит}{такт}$), а пропускная способность канала $C = 1 - H(p)$ – максимально возможное среднее число передаваемых бит на символ (см. формулу (17)). Но поскольку в рассматриваемом случае символы являются двоичными, то естественно считать, что один двоичный символ передаётся за один такт. Поэтому пропускная способность также измеряется в битах на такт. Если в канале ошибок нет, т.е. $p = 0$ и $H(p) = 1$, то пропускная способность максимальна: 1 бит за один такт. Пропускная способность будет меньше единицы, например, если за n тактов передачи данных может быть принято только $k < n$ бит – информационных разрядов, а $n - k$ тактов отведены для передачи вспомогательной (контрольной) информации.

Следующая теорема Шеннона, которую иногда называют обращением теоремы о кодировании, утверждает, что *надежная передача* со скоростью $R > C$ невозможна.

Теорема 3.4. *Если скорость передачи информации превышает пропускную способность канала ($R > C$), то существует значение ε_0, $0 < \varepsilon_0 < 1$, $\varepsilon_0 = \varepsilon_0(R, C)$, такое, что при любом способе блоковой передачи со скоростью, не меньшей R, вероятность ошибки передачи $p(E) \geq \varepsilon_0$.*

Под ошибкой, как и в предыдущей теореме, понимается событие E, состоящее в том, что произвольный передаваемый блок информации на входе канала будет отличаться от блока, принятого на выходе канала.

4 Передача непрерывной информации

4.1 Сигналы и их спектры

В современных информационных системах происходит передача и обработка не только дискретной – символьной информации, но также и непрерывной информации, например, речевых сообщений и изображений. Математическим представлением сигнала в таких случаях являются, как правило, непрерывные вещественные функции вещественного аргумента (обычно – времени), которые в процессе обработки могут определённым образом преобразовываться в дискретные последовательности с целью последующей компьютерной обработки.

Детерминированным *сигналом* называется вещественная функция времени $f(t)$.

Периодическим называется сигнал, удовлетворяющий условию $f(t) = f(t+T)$ для любого $t: \ -\infty \leq t \leq \infty$, где T – константа, называемая периодом функции $f(t)$.

Примером простейшего периодического сигнала является так называемое *гармоническое колебание* – тригонометрическая функция времени $f(t) = a\cos(\omega t - \varphi)$, где a – амплитуда, $\omega = \frac{2\pi}{T}$ – угловая частота, φ – начальная фаза.

Периодическая функция, имеющая период T и удовлетворяющая условию Дирихле (конечное число экстремумов в любом промежутке ширины T и непрерывность на нём, за исключением конечного числа точек разрыва первого рода), может быть представлена рядом Фурье

$$f(t) = a_0 + \Sigma_{k=1}^{\infty} A_k \cos(k\omega_1 t - \varphi_k), \tag{23}$$

где a_0 – постоянная составляющая; A_k – амплитуда $k-$той гармонической составляющей, $k\omega_1$ – её частота и φ_k – её фаза.

Гармонические составляющие (*гармоники*) имеют кратные частоты: $\omega_1 = \frac{2\pi}{T}$; $\omega_2 = 2\omega_1$; $... \ \omega_k = k\omega_1, ...$. Совокупность амплитуд гармонических составляющих называют *амплитудным спектром* сигнала. Ряд (23) может быть представлен в виде

$$f(t) = a_0 + \Sigma_{k=1}^{\infty}(a_k \cos k\omega_1 t + b_k \sin k\omega_1 t), \tag{24}$$

где $a_k = A_k cos\varphi_k$, $b_k = A_k sin\varphi_k$, $A_k = \sqrt{a_k^2 + b_k^2}$, $\operatorname{tg}\varphi_k = \frac{b_k}{a_k}$.

Коэффициенты a_k и b_k находятся по формулам

$$a_k = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\cos k\omega_1 t dt, \tag{25}$$

$$b_k = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\sin k\omega_1 t dt. \tag{26}$$

Можно заметить, что для чётной функции $f(t)$ коэффициенты $b_k = 0$ для всех k, а для нечётной функции в ноль обращаются все коэффициенты a_k. Постоянная составляющая вычисляется по формуле

$$a_0 = \frac{1}{T}\int_{-T/2}^{T/2} f(t)dt. \tag{27}$$

Средняя мощность сигнала определяется формулой

$$\overline{f^2(t)} = \frac{1}{T}\int_{-T/2}^{T/2} f^2(t)dt. \tag{28}$$

В информационных системах физически сигналы представляются электрическим напряжением или током, и величину (28) можно интерпретировать как среднюю мощность сигнала, приходящуюся на нагрузку в 1 *ом*.

Пример. Вычислим спектр периодического сигнала, представленного на рис. 4.12.

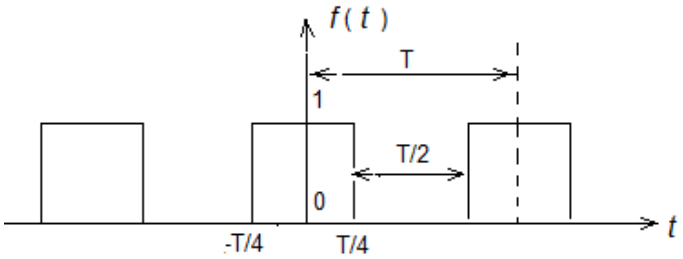


Рис. 4.12: Периодическая последовательность импульсов

$$a_0 = \frac{1}{T}\int_{-T/4}^{T/4} dt = \frac{1}{T}\cdot\frac{T}{2} = \frac{1}{2}.$$

$$a_k = \frac{2}{T}\int_{-T/2}^{T/2} f(t)\cos k\omega_1 t dt = \frac{2}{Tk\omega_1}\sin k\omega_1 t\,\Big|_{-T/4}^{T/4} = \frac{2}{Tk\omega_1}2sin\frac{k\omega_1 T}{4} = \frac{sin\frac{k\omega_1 T}{4}}{\frac{k\omega_1 T}{4}}.$$

Подставляя $\omega_1 = \frac{2\pi}{T}$; $\omega_1 T = 2\pi$; $\frac{k\omega_1 T}{4} = k\frac{\pi}{2}$, получаем $a_k = \frac{sin\frac{k\pi}{2}}{\frac{k\pi}{2}}$.

$$f(t) = 1 + \sum_{k=1}^{\infty}\frac{sin\frac{k\pi}{2}}{\frac{k\pi}{2}}cos\omega_1 kt; \quad f(t) = 1 + \frac{2}{\pi}\sum_{k=1}^{\infty}(-1)^{i+1}\frac{cos\omega_1(2i-1)t}{2i-1}.$$

Поскольку рассматриваемая функция является чётной, $b_k = 0$. *Амплитуды*

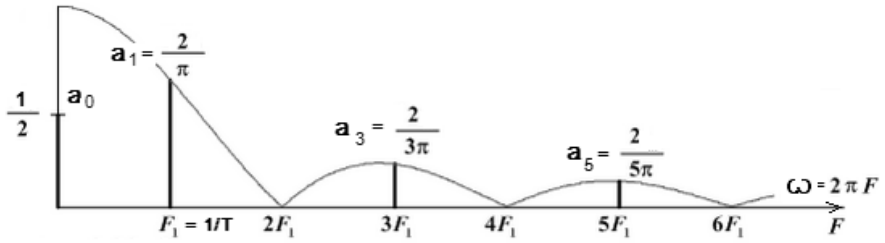


Рис. 4.13: Амплитудный спектр периодических импульсов

составляющих спектра быстро убывают — пропорционально $\frac{1}{k}$ с ростом k. На рис. 4.13 изображен линейчатый дискретный амплитудный спектр рассматриваемой периодической последовательности импульсов ширины $T/2$ с единичной амплитудой и периодом T. На рис. 4.14 показано, как с ростом числа слагаемых гармоник уточняется представление сигнала их суммой. □

На практике часто используют приближение сигналов суммой ряда Фурье с конечным числом членов, учитывая при этом совокупность тех гармоник, амплитуды которых больше некоторого заданного значения. В таком случае спектр приближения становится ограниченным, но сигнал, определяемый приближённым рядом, является в некоторой степени искаженным.

Ограниченность спектра имеет место при передаче сигналов в электронных информационных системах в силу физических свойств источников и приемников информации. Поэтому *идеализация, связанная с ограничением спектра сигналов некоторыми допустимыми значениями, представляется приемлемой в большинстве случаев применения спектрального анализа на практике.*

При разложении периодического сигнала в ряд Фурье, как следует из формулы (23), функция $f(t)$ представляется суммой бесконечного числа гармоник, отстоящих последовательно одна от другой

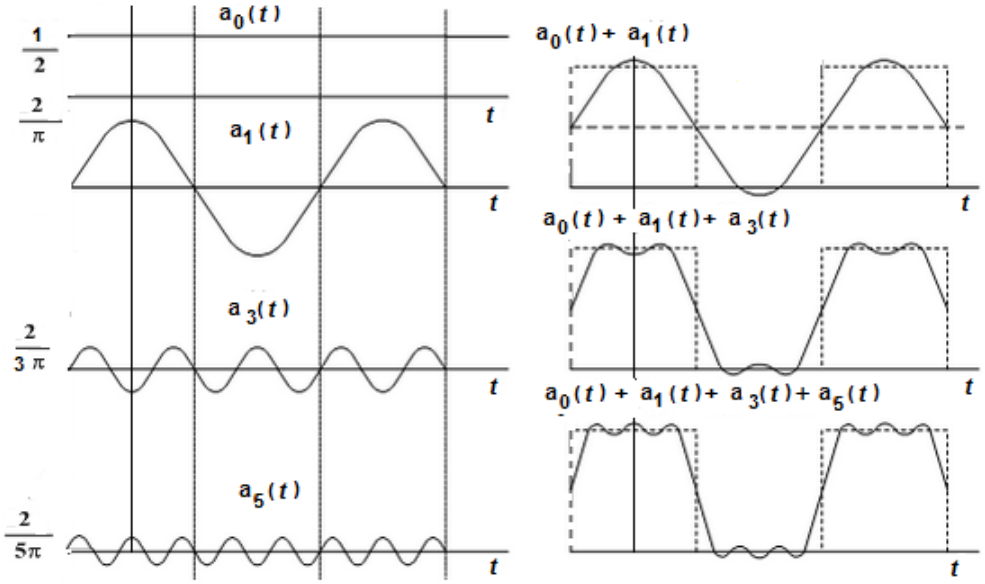


Рис. 4.14: Приближение импульсного периодического сигнала одной, двумя и тремя гармониками

по частоте на величину $\omega_1 = \frac{2\pi}{T}$. Число гармоник бесконечное, счётное, дискретное.

Если представить непериодический сигнал как предельный случай периодического, когда период $T \to \infty$, расстояние между частотами гармоник будет стремиться к нулю: $\frac{2\pi}{T} \to 0$. В этом случае можно предположить, что дискретный спектр "перейдет" в непрерывный. Именно в таком смысле можно понимать спектральное преобразование непериодических сигналов.

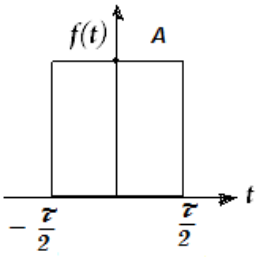


Рис. 4.15: Одиночный импульс с амплитудой A и длительностью τ

Если функция $f(t)$ удовлетворяет условиям Дирихле на любом конечном отрезке оси x и абсолютно интегрируема вдоль всей оси ($\int_{-\infty}^{\infty} |f(t)|dt$ сходится), то имеет место пара преобразований Фурье

$$S(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt, \quad f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S(\omega)e^{i\omega t}d\omega. \qquad (29)$$

Пример. Найдем спектр одиночного прямоугольного импульса (рис.4.15) по формуле (29). Функция $f(t)$ равна A внутри отрезка $[-\frac{\tau}{2}, \frac{\tau}{2}]$ и нулю – всюду вне этого отрезка.

$$S(\omega) = \int_{-\frac{\tau}{2}}^{\frac{\tau}{2}} Ae^{-i\omega t}dt = \frac{A}{-i\omega}(e^{-i\omega\frac{\tau}{2}} - e^{i\omega\frac{\tau}{2}})$$
$$= \frac{A}{-i\omega}(-2i)sin\omega\frac{\tau}{2} = \frac{2A}{\omega}\sin\omega\frac{\tau}{2} = A\tau\frac{\sin\omega\frac{\tau}{2}}{\omega\frac{\tau}{2}}.$$

Полученный непрерывный спектр имеет нули в точках $\omega = \frac{2\pi k}{\tau}$, $k = 1, 2, \ldots$, и быстро убывает с ростом частоты ω. На рис. 4.16 представлен модуль полученного спектра одиночного прямоугольного импульса с шириной τ в области положительных частот.

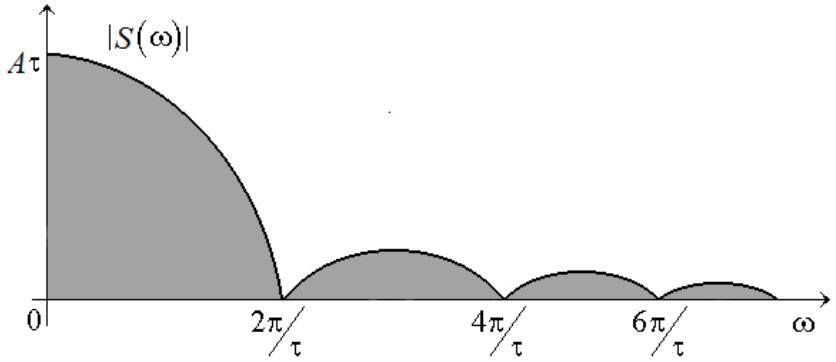


Рис. 4.16: Модуль спектра одиночного импульса

4.2 Дискретизация сигналов. Теорема Котельникова

Теорема 4.1. *Если функция $f(t)$ удовлетворяет условиям Дирихле, абсолютно интегрируема вдоль всей оси t, и её спектр $S(\omega)$ также удовлетворяет условиям Дирихле и имеет вид*

$$S(\omega) = \begin{cases} \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt, & \text{если } |\omega| \le 2\pi F; \\ 0, & \text{если } |\omega| > 2\pi F \end{cases}$$

для некоторого числа $F > 0$, то она полностью определяется своими значениями в точках $t = \frac{k}{2F}$, $k \in \mathbb{Z}$, по формуле

$$f(t) = \sum_{k=-\infty}^{\infty} f(k\Delta t)\frac{\sin 2\pi F(t - k\Delta t)}{2\pi F(t - k\Delta t)}, \qquad (30)$$

где $\Delta t = \frac{1}{2F}$.

Доказательство. Сначала приведем формулы представления ряда Фурье в комплексной форме (см. [9], с. 509), равносильные ранее использованным формулам (24),(25),(26).

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{ik\frac{2\pi t}{T}};$$

$$c_k = \frac{1}{T}\int_{-T/2}^{T/2} f(\xi) e^{-ik\frac{2\pi\xi}{T}} d\xi. \tag{31}$$

Напомним, что T – обозначение периода функции $f(t)$.

Для спектра $S(\omega)$, как вещественной функции, удовлетворяющей условию Дирихле и заданной на конечном интервале $-2\pi F \le \omega \le 2\pi F$, можно построить ряд Фурье, который будет совпадать с функцией $S(\omega)$ на указанном интервале $-2\pi F \le \omega \le 2\pi F$. Это можно сделать, бесконечно продолжая функцию $S(\omega)$ путём периодического повторения влево и вправо её ненулевого участка значений. Таким образом будет получена *функция $S^*(\omega)$ с периодом $T = 4\pi F$, для которой ряд Фурье будет иметь вид*

Рис. 4.17: Академик В.А. Котельников, 1908–2005 гг., советский учёный, радиоинженер, специалист в области теории связи

$$S^*(\omega) = \frac{1}{2}\sum_{k=-\infty}^{\infty} A_k e^{\frac{ik\omega}{2F}}; \tag{32}$$

$$A_k = \frac{1}{2\pi F}\int_{-2\pi F}^{2\pi F} S(\omega) e^{-ik\frac{\omega}{2F}} d\omega. \tag{33}$$

Для удобства здесь, по сравнению с формулами (31), $A_k = 2c_k$.

Обратное преобразование Фурье для получения $f(t)$ по спектру $S^*(\omega)$ будет иметь вид

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S^*(\omega) e^{i\omega t} d\omega$$

$$= \frac{1}{2\pi} \int_{-2\pi F}^{2\pi F} S(\omega) e^{i\omega t} d\omega. \tag{34}$$

Положив в (34) $t = \frac{k}{2F}$, получим

$$f(\frac{k}{2F}) = \frac{1}{2\pi} \int_{-2\pi F}^{2\pi F} S(\omega) e^{i\omega \frac{k}{2F}} d\omega. \tag{35}$$

Сравнивая (35) с A_k (33), нетрудно заметить, что

$$A_k = f(-\frac{k}{2F}) \cdot \frac{1}{F} = 2\Delta t \cdot f(-k\Delta t).$$

Подставляя A_k в (32), получаем

$$S^*(\omega) = \frac{1}{2} \sum_{k=-\infty}^{\infty} 2\Delta t \cdot f(-k\Delta t) e^{\frac{ik\omega}{2F}} = \sum_{k=-\infty}^{\infty} \Delta t \cdot f(k\Delta t) e^{\frac{-ik\omega}{2F}} \tag{36}$$

(произведена замена переменной суммирования k на $-k$).

Поскольку функция однозначно определяется своим спектром, имеем

$$f(t) = \frac{1}{2\pi} \int_{-2\pi F}^{2\pi F} S(\omega) e^{i\omega t} d\omega$$

$$= \frac{1}{2\pi} \int_{-2\pi F}^{2\pi F} \left(\sum_{k=-\infty}^{\infty} \Delta t \cdot f(k\Delta t) e^{\frac{-ik\omega}{2F}} \right) e^{i\omega t} d\omega$$

$$= \frac{\Delta t}{2\pi} \sum_{k=-\infty}^{\infty} \left(f(k\Delta t) \int_{-2\pi F}^{2\pi F} e^{i\omega(t-k\Delta t)} d\omega \right)$$

$$= \frac{\Delta t}{2\pi} \sum_{k=-\infty}^{\infty} \left(f(k\Delta t) \frac{(e^{i2\pi F(t-k\Delta t)} - e^{-i2\pi F(t-k\Delta t)})}{i(t-k\Delta t)} \right)$$

$$= \frac{\Delta t}{2\pi} \sum_{k=-\infty}^{\infty} \left(f(k\Delta t) \frac{2i \sin 2\pi F(t-k\Delta t)}{i(t-k\Delta t)} \right)$$

$$= \frac{1}{4\pi F} \sum_{k=-\infty}^{\infty} f(k\Delta t) \frac{2 \sin 2\pi F(t-k\Delta t)}{(t-k\Delta t)}$$

$$= \sum_{k=-\infty}^{\infty} f(k\Delta t) \frac{\sin 2\pi F(t - k\Delta t)}{2\pi F(t - k\Delta t)}.$$

Получено точное разложение функции с ограниченным спектром

$$f(t) = \sum_{k=-\infty}^{\infty} f(k\Delta t) \frac{\sin 2\pi F(t - k\Delta t)}{2\pi F(t - k\Delta t)}. \quad \square$$

Теорема Котельникова теоретически обосновывает возможность передачи непрерывной информации – сигнала с ограниченным спектром – в виде последовательности числовых значений этого сигнала, взятых через промежутки времени $\Delta t = \frac{1}{2F}$. Но на практике может быть использовано только ограниченное число отсчётов: если время передачи сигнала составляет промежуток длительности $\mathfrak{T}$, то число отсчётов будет равно $[\frac{\mathfrak{T}}{\Delta t}] + 1$, и по этим отсчётам восстановить сигнал можно будет только приближённо. Несмотря на это обстоятельство, дискретизация сигналов по времени через промежутки $\Delta t = \frac{1}{2F}$, обоснованная разложением (30), широко применяется при вводе непрерывных сигналов в компьютеры с целью дальнейшей цифровой обработки.

Каждый отсчёт сигнала – значение $f(k\Delta t)$ – является вещественным числом, поэтому отсчёты для возможности ввода в компьютер также должны быть подвержены дискретизации, но теперь уже не по времени, а по уровню значений. Если в результате дискретизации информация о сигнале поступает в двоичный n разрядный регистр, то допустима дискретизация каждого отсчёта не более чем по 2^n уровням (включая нулевой).

Далее будет показано, как происходит дискретизация сигнала по времени и по уровням значений отдельных отсчётов. Шаг дискретизации по оси X определяется теоремой Котельникова, а шаг дискретизации значений отсчётов по оси Y определяется числом выбранных уровней квантования.

4.3 Аналого-цифровое и цифро-аналоговое преобразование информации

Как было показано выше, и периодические и непериодические сигналы $f(t)$ имеют спектры $S(\omega)$, в которых амплитуды составляющих (или мощности в частотных промежутках) быстро убывают с ростом частоты ω. На практике это позволяет указать такую частоту $\omega_{\text{в}}$, что почти вся мощность сигнала будет сосредоточена в его частотном отрезке от 0 до $\omega_{\text{в}}$. Выбор достаточно малого значения ε приводит к нахождению частоты $\omega_{\text{в}}$:

$$\frac{\int_{\omega_{\text{в}}}^{\infty} |S(\omega)|^2 d\omega}{\int_{0}^{\omega_{\text{в}}} |S(\omega)|^2 d\omega} < \varepsilon.$$

На рис. 4.18 показан модуль спектра некоторого непериодического сигнала и отмечена частота $\omega_{\text{в}}$, "выше" которой спектр сигнала быстро "затухает". В электронике такое затухание сигналов с ростом частоты является неизбежным следствием физических свойств среды, и ограничение спектра частотой $\omega_{\text{в}}$ в этом смысле может оказаться вполне естественным.

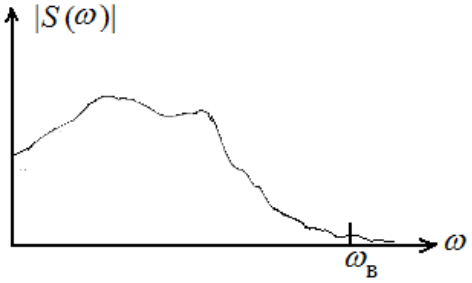


Рис. 4.18: Выбор частоты $\omega_{\text{в}} = \omega_{\text{в}}(\varepsilon)$

Аппроксимация модуля спектра его частью в конечном отрезке $[0, \omega_{\text{в}}]$ позволяет применить теорему дискретизации Котельникова, выбрав шаг квантования сигнала равным $\Delta t = \frac{1}{2F_{\text{в}}}$, где $F_{\text{в}} = \frac{\omega_{\text{в}}}{2\pi}$. Исходный сигнал $f(t)$ и сигнал, полученный в результате квантования с шагом $T = \Delta t$, представлены на рис. 4.19.

Кроме квантования сигнала по времени, для ввода информации, представляемой сигналом, в компьютер, необходимо также его квантование по амплитуде или, как часто говорят, *по уровню*. Квантование по уровню необходимо для того, чтобы заменить непрерывные значения амплитуд конечным числом их рациональных приближе-

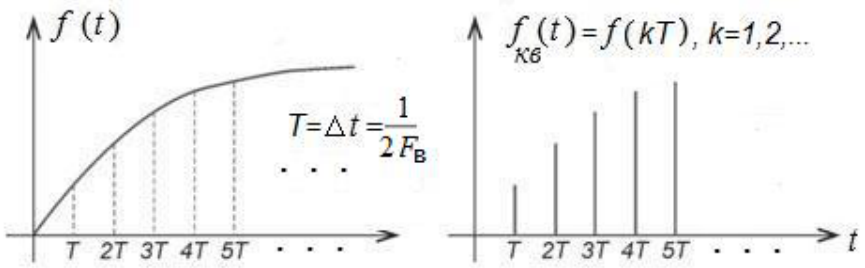


Рис. 4.19: Сигнал до и после квантования

ний, которые могут быть записаны в ячейки памяти компьютера.

Физическое устройство, обеспечивающее квантование сигналов с целью их ввода в компьютер, называется *аналого-цифровым преобразователем* (АЦП). С ознакомительной целью приведём один из принципов построения АЦП, основанный на сравнении с уровнем.

Для сравнения с уровнем используется *компаратор* – физический элемент, реализующий функцию

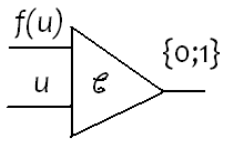


Рис. 4.20: Обозначение компаратора

$$\mathscr{C}(f(t),u)=\begin{cases}1, & \text{если } f(t)\geq u;\\ 0, & \text{если } f(t)<u,\end{cases}$$

где $f(t)$ – значение сигнала в момент t, сравниваемое с задаваемым уровнем u. На электронных схемах компаратор обозначается как показано на рис.4.20. На выходе компаратора могут быть только два состояния: ноль или единица. Нулевое состояние заменяется единичным, как только сигнал достигает уровня u или становится больше его. Единичное состояние заменяется нулевым, как только сигнал становится меньше уровня u.

На рис.4.21 представлена схема с тремя компараторами, позволяющая осуществлять квантование сигнала так, что на её выходе могут быть только четыре дискретных значения: $0,\ u_1,\ u_2=2u_1,\ u_3=3u_1$. При этом полагается, что $0\leq f(t)<u_4=4u_1$.

Делитель напряжения на четырёх резисторах делит опорное напряжение u_4 на равные части, задавая тем самым уровни $0,\ u_1,\ u_2=2u_1,\ u_3=3u_1$ (полагаем ток в цепи резисторов условно равным единице). Дешифратор ДШ преобразует тройку выходов y_1,y_2,y_3 компараторов $\mathscr{C}_1,\ \mathscr{C}_2,\ \mathscr{C}_3$ в двоичное слово β_2,β_1.

Дешифратор ДШ реализует две логические функции $\beta_1=\beta_1(y_1,y_2,y_3)$ и

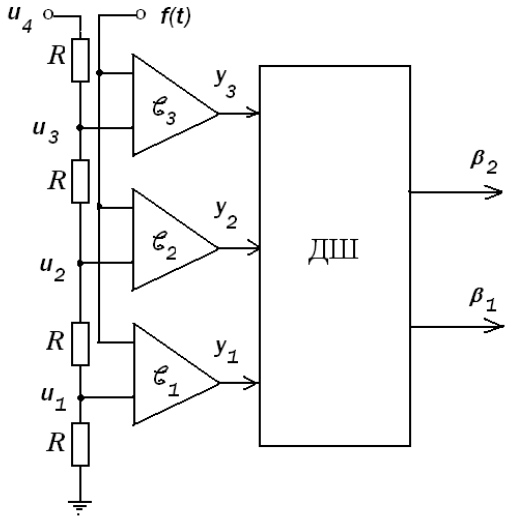


Рис. 4.21: Схема простейшего АЦП

$\beta_2 = \beta_2(y_1, y_2, y_3)$. Эти функции несложно определить по таблице

y_1	y_2	y_3	β_1	β_2
0	0	0	0	0
1	0	0	0	1
1	1	0	1	0
1	1	1	1	1

следующим образом: $\beta_1 = y_2$; $\beta_2 = y_1\bar{y_2} \vee y_3$.

На рис. 4.22 показано, как формируются значения выходных сигналов компараторов $y_3y_2y_1$. Следует учесть, что кроме квантования по уровню для ввода преобразованного сигнала в компьютер происходит квантование также и по времени с шагом Δt, определяемым теоремой Котельникова. Поэтому будут использованы значения квантованного по амплитуде сигнала только в точках $t = 0, \Delta t, 2\Delta t, 3\Delta t \ldots$.

Для достижения большей точности необходимо увеличить число используемых компараторов (сравнений). Если число уровней квантования (включая нулевой) равно $s + 1$, где s – число компараторов, то, очевидно, число двоичных выходных разрядов АЦП и, соответственно, регистра, принимающего полученную в результате дискретизации информацию, должно быть равным $\lceil \log_2(s + 1) \rceil$.

Реализацию квантования по уровню можно осуществить двумя способами, называемыми *усечение* и *округление*. Для пояснения этих способов приведен рис. 4.23.

Обозначим k – номер шага квантования по времени; $y(k) = f(k\Delta t)$ – значение сигнала в момент $k\Delta t$; $y_{\text{кв}}(k)$ – значение, получаемое в результате

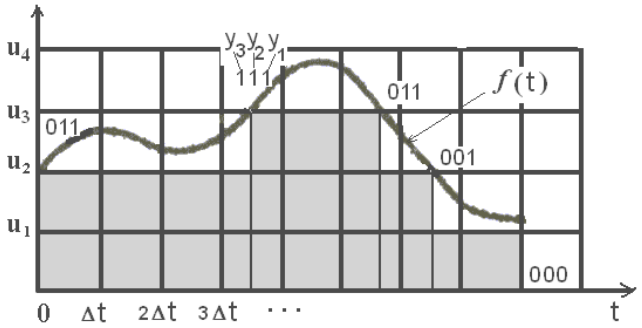


Рис. 4.22: Формирование выходов компараторов

квантования в момент времени $k\Delta t$; Δy расстояние между уровнями или шаг квантования.

Вычисление по методу усечения производится по формуле

$$y_{\text{кв}}(k) = \lfloor y(k)/\Delta y \rfloor \cdot \Delta y,$$

а по методу округления – по формуле

$$y_{\text{кв}}(k) = \lfloor y(k)/\Delta y + 0,5 \rfloor \cdot \Delta y.$$

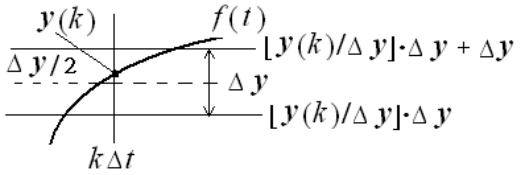


Рис. 4.23: Усечение и округление

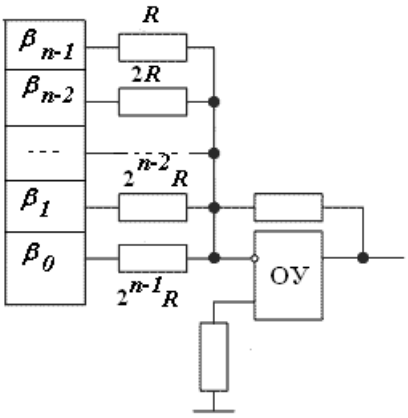


Рис. 4.24: Схема ЦАП

Ошибкой квантования является величина

$$e(k) = y_{\text{кв}}(k) - y(k).$$

Для метода усечения получается

$$\Delta y < e(k) \leq 0,$$

а для метода округления

$$-\frac{\Delta y}{2} < e(k) < \frac{\Delta y}{2}.$$

Физическое устройство, обеспечивающее преобразование дискретных величин в непрерывные, называется *цифро-аналоговым преобразователем* (ЦАП). Выше было дано описание АЦП, основанного на принципе деления напряжений и сравнении с уровнями. Рассмотрим построение ЦАП, основанного па принципе суммирования токов.

Двоичный код, подлежащий преобразованию в аналоговое значение, имеет вид $\tilde{\beta} = \beta_{n-1}\beta_{n-2}\ldots\beta_1\beta_0$. Определяемое этим кодом десятичное число, в соответствии с двоичной позиционной системой счисления, имеет вид

$$N(\tilde{\beta}) = 2^{n-1}\beta_{n-1} + 2^{n-2}\beta_{n-2} + \cdots + 2^1\beta_1 + 2^0\beta_0 \;=\; \sum_{i=0}^{n-1} 2^i\beta_i.$$

Из этой формулы усматривается принцип работы ЦАП на основе суммирования токов, схема которого приведена на рис. 4.24.

Операционный усилитель ОУ суммирует токи. Если $\beta_{n-1} = 1$, то входящий в сумму ток в соответствующей цепи, обратно пропорциональный сопротивлению в этой цепи, будет в 2^{n-1} раз больше, чем ток, возникающий при единичном значении младшего двоичного разряда β_0.

4.4 Пропускная способность непрерывного канала с аддитивной помехой с учётом дискретизации сигнала

Рассмотрим передачу сигнала $a(t)$ по каналу передачи информации с учётом действующей помехи $\xi(t)$, приводящей к тому, что сигнал $x(t)$ на выходе канала может не совпадать с сигналом на его входе. Будем полагать, что канал "пропускает" сигнал только в полосе частот от 0 до F, что определяет ограничение спектра входного сигнала, и вследствие чего в соответствии с теоремой Котельникова сигнал $a(t)$ передаётся отдельными значениями через промежутки времени $\Delta t = \frac{1}{2F}$. Тогда за время T число переданных отсчётов сигнала будет $n = \frac{T}{\Delta t} = 2TF$ (для упрощения дальнейшей записи формул будем считать t кратным Δt).

И входной сигнал, и сигнал на выходе канала будут представлены последовательными наборами чисел (отсчётов)

$$a_1, a_2, ..., a_i, ..., a_n \quad \text{и} \quad x_1, x_2, ..., x_i, ..., x_n$$

как показано на рис. 4.25.

Обозначим эти последовательности следующим образом: A – источник информации на входе канала и X – источник информации на выходе канала. Источник помехи обозначим Ξ. Будем считать, что помеха действует независимо на каждый из отсчётов

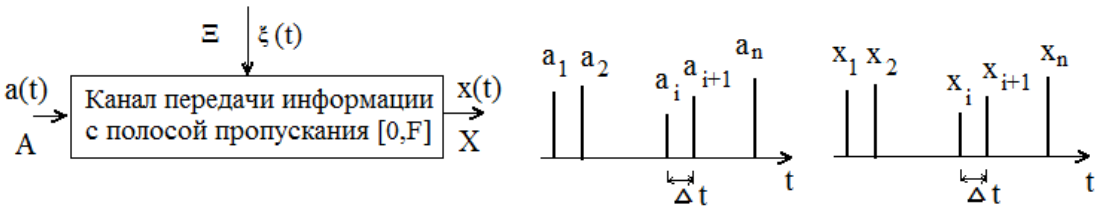


Рис. 4.25: Схема передачи информации по каналу с помехой

$a_1, a_2, ..., a_i, ..., a_n$, являясь одномерной случайной величиной ξ с дисперсией $\sigma_{\text{п}}$. Тогда источник помех можно рассматривать как вектор $\Xi = (\xi_1, ..., \xi_i, ..., \xi_n)$, в котором все компоненты ξ_i – одинаковые случайные величины: $\xi_i = \xi$. Каждый выходной отсчёт x_i будет случайной величиной, определяемой отсчётом сигнала и воздействием помехи.

Пусть существуют плотности распределения, входящие в приводимые ниже формулы, определяющие энтропии этих источников и условную энтропию:

$$H(A) = \int\int\cdots\int p(a_1, a_2, ..., a_n) \log p(a_1, a_2, ..., a_n) da_1 da_2 \cdots da_n;$$

$$H(X) = \int\int\cdots\int p(x_1, x_2, ..., x_n) \log p(x_1, x_2, ..., x_n) dx_1 dx_2 \cdots dx_n;$$

$$H(X/A) = \int\int\cdots\int p(x_1, ..., x_n; a_1, ..., a_n) \log p(x_1, ..., x_n/a_1, ..., a_n) dx_1 dx_2 \cdots dx_n.$$

Энтропия определялась как среднее количество информации в битах, приходящееся на один символ источника информации. В рассматриваем нами случае энтропия будет средним количеством бит на передаваемый набор из n числовых значений: $H(\cdot)(\frac{\text{бит}}{n \text{ значений}})$. Как было определено выше, T – время передачи n значений-отсчётов сигнала $a_1, ..., a_n$. Таким образом имеем "размерность" T $(\frac{\text{сек. передачи}}{n \text{ значений}})$. Тогда размерность отношения $\frac{H(\cdot)}{T}$ будет $(\frac{\text{бит}}{\text{сек}})$.

Скорость R передачи информации по каналу определяется выражением

$$R = \frac{1}{T}(H(X) - H(X/A)) \quad \left(\frac{\text{бит}}{\text{сек}}\right).$$

Пропускная способность канала равна

$$C = \max_{P(A)} \frac{1}{T}(H(X) - H(X/A)) \quad \left(\frac{\text{бит}}{\text{сек}}\right),$$

где максимум берется по всевозможным вероятностным распределениям источника информации A.

В канале действует помеха Ξ, которая статистически не зависит от входного сигнала $a(t)$, вследствие чего условная энтропия $H(X/A)$ равна условной энтропии входного сигнала и помехи

$$H(X/A) = H(A/A) + H(\Xi/A).$$

$H(A/A) = 0$, поскольку неопределённость источника A при условии получения всей информации от источника A равна нулю. $H(\Xi/A) = H(\Xi)$, так как помеха статистически не зависит от передаваемого сигнала. Поэтому

$$H(X/A) = H(\Xi).$$

Помеха $\Theta = (\xi_1, ..., \xi_n)$ действует случайно и независимо на каждый отсчёт входного сигнала $a_1, ..., a_n$. Поэтому $H(\Xi) = nH(\xi_i)$, $\xi_i = \xi$ для всех $i = 1, 2..., n$, где ξ – одномерная случайная величина, имеющая гауссово распределение с дисперсией $\sigma^2_{\text{п}}$. Учитывая формулу для энтропии нормального распределения (7), получаем

$$H(X/A) = H(\Xi) = n \log \sqrt{2\pi e}\sigma_{\text{п}}.$$

С учётом полученного значения энтропии $H(X/A)$, можно записать

$$C = \max_{P(A)} \frac{1}{T}(H(X) - n \log \sqrt{2\pi e}\sigma_{\text{п}}).$$

Максимум энтропии $H(X)$ достигается в случае, когда совокупность отсчётов сигнала и воздействующей случайной аддитивной помехи имеет нормальное распределение с суммарной дисперсией $\sigma_c^2 + \sigma^2_{\text{п}}$, где σ_c^2 – дисперсия одного отсчёта. Поэтому

$$\max_{P(A)} H(X) = n \log \sqrt{2\pi e}\sqrt{\sigma_c^2 + \sigma^2_{\text{п}}},$$

$$C = \frac{1}{T}\left(n \log \sqrt{2\pi e}\sqrt{\sigma_c^2 + \sigma^2_{\text{п}}} - n \log \sqrt{2\pi e}\sigma_{\text{п}}\right).$$

Учитывая, что $T = n\Delta t = \frac{n}{2F}$, получаем

$$C = 2F\left(\log\sqrt{2\pi e}\sqrt{\sigma_c^2+\sigma_{\text{п}}^2} - \log\sqrt{2\pi e}\sigma_{\text{п}}\right);$$

$$C = 2F\log\frac{\sqrt{2\pi e}\sqrt{\sigma_c^2+\sigma_{\text{п}}^2}}{\sqrt{2\pi e}\sigma_{\text{п}}} = F\log\frac{\sigma_c^2+\sigma_{\text{п}}^2}{\sigma_{\text{п}}^2};$$

$$C = F\log\left(\frac{\sigma_c^2}{\sigma_{\text{п}}^2}+1\right). \tag{37}$$

В итоге, в соответствии с полученной формулой (37), можно утверждать, что пропускная способность канала тем выше, чем шире его полоса пропускания (чем больше F) и чем больше отношение сигнал/шум $\frac{\sigma_c^2}{\sigma_{\text{п}}^2}$.

Квадрат дисперсии σ_c^2 характеризует мощность сигнала, а $\sigma_{\text{п}}^2$ – мощность помехи.

В завершение этого раздела приведем без доказательства ещё одну теорему К. Шеннона, касающуюся скорости передачи информации и пропускной способности, в которой фигурирует оценка (37).

Теорема 4.2. *Пусть P – средняя мощность источника информации и Q – средняя мощность помехи в промежутке частот $[0, F]$. Тогда можно применить такое двоичное кодирование, что можно будет передавать информацию по каналу со скоростью*

$$C = F\log\left(\frac{P}{Q}+1\right)$$

и со сколь угодно малой частотой ошибок. Но никакой метод кодирования не позволяет осуществить передачу со скоростью, превышающей C, со сколь угодно малой частотой ошибок.

5 Алгоритмическая обработка информации

5.1 Логические основания математики и алгоритмы

Под алгоритмом, применимым для решения некоторого класса однотипных задач (*называемого проблемой*), традиционно понимается некоторое общее правило, если таковое существует, при помощи которого решение любой задачи указанного класса может быть найдено механически, путём конечной последовательности точно определённых действий (шагов решения).

Пожалуй, самым известным является алгоритм Евклида нахождения наибольшего общего делителя двух натуральных чисел.

Слово "алгоритм" происходит от имени арабского математика Муххамеда **аль-Хорезми** (783-850 гг.), который внёс существенный вклад в начальное развитие подобных пошаговых методов арифметических вычислений. Имя этого математика постепенно трансформировалось в слово **алгоритм**, которое и поныне обозначает пошаговые, строго определённые и выполняемые "механически" вычисления.

Прогресс в развитии таких алгоритмических методов породил предположение о том, что для любой поставленной математической задачи должно найтись её алгоритмическое решение. Сторонниками этого предположения были многие знаменитые математики, например, Рене Декарт, Готфрид Лейбниц и даже Давид Гильберт. Развивая аналитическую геометрию, Декарт намеревался сделать её доступной алгебраическим методам вычислений и тем самым продвинуться к её полной алгоритмизации. Великий Лейбниц пытался придумать специально приспособленную для решения математических задач автоматически работающую машину. Гильберт стимулировал исследования в этом направлении, призывая математиков к нахождению алгоритмического метода решения некоторых задач, в частности, выдвижением в 1900 г. на втором Международном конгрессе математиков в Париже своей 10-й (из 23-х представленных

им) проблемы:

"*Пусть задано диофантово уравнение с произвольными неизвестными и целыми числовыми коэффициентами. Указать способ, при помощи которого возможно после конечного числа операций установить, разрешимо ли это уравнение в целых рациональных числах*".

В этой проблеме речь идет о целочисленном решении уравнения

$$P(x_1, x_2, ..., x_n) = 0,$$

где P – многочлен с целыми коэффициентами и целыми показателями степеней. Степень уравнения равна степени многочлена P.

Рис. 5.26: Курт Гёдель, 1906–1978 гг., знаменитый австрийский математик

По-видимому, Гильберт считал, что искомый метод существует и рано или поздно будет найден. Вопрос о том, что *такой метод может в принципе не существовать*, во времена Гильберта не стоял.

В 1931 г. в одном из немецких научных журналов появилась статья: "О формально неразрешимых противоречиях *Principia Mathematica* и родственных систем". Автором её был 25-летний математик из Венского университета Курт Гёдель. Эта работа явилась одним из величайших достижений математики. В ней была впервые доказана алгоритмическая неразрешимость некоторых математических проблем.

Точнее, было показано, что многие известные математические проблемы не могут быть решены с помощью алгоритмов из некоторого точно определённого заданного класса алгоритмов. При этом значимость результатов Гёделя зависела от степени совпадения выбранного для доказательства класса алгоритмов со всеми алгоритмами, понимаемыми на интуитивном уровне. Поэтому *потребовалось строгое определение алгоритма, на основе которого можно бы было исследовать разрешимость произвольной заданной проблемы.*

Используемый для доказательства неразрешимости класс алгоритмов (не называвшийся именно так Гёделем) представлял собой *рекурсивные функции* – специально разработанный для этой цели

и сформировавшийся в переписке Курта Гёделя с Жаком Эрбраном математический аппарат, причём главная роль в создании этого формализма принадлежала Жаку Эрбрану.

5.2 Рекурсивные функции

Рекурсивные функции являются подклассом функций вида

$$f : \mathbb{N}_0 \times \cdots \times \mathbb{N}_0 \to \mathbb{N}_0,$$

где $\mathbb{N}_0 = \{0, 1, 2, ...\}$, и определяются путём задания базисных функций и операций, порождающих из базисных функций все остальные в рассматриваемом классе.

Рис. 5.27: Жак Эрбран, 1908–1931 гг., знаменитый французский математик. Именно он создал аппарат рекурсивных функций. Трагически погиб в 23 года. Считался выдающимся молодым математиком

Базисные функции:

$\mathscr{B}1$) Константа $0 \in \mathbb{N}_0$;

$\mathscr{B}2$) $S(x) = x + 1$ – функция добавления единицы к аргументу $x \in \mathbb{N}_0$ (функция следования);

$\mathscr{B}3$) $I_m^n(x_1, ..., x_{m-1}, x_m, x_{m+1}, ..., x_n) = x_m$, $1 \leq m \leq n$, – функции, возвращающие $m-$й аргумент из n заданных аргументов.

Операции:

$\mathscr{O}1$) C – суперпозиции (подстановки на места аргументов одной функции выражений вида $f(x_1, ..., x_r)$ – других функций с указанными аргументами).

$\mathscr{O}2$) Операция Πp – примитивной рекурсии, которая задаётся следующим образом.

Сначала подчеркнём, что Πp – функциональная операция (оператор) ставит в соответствие двум заданным функциям – третью: $f = \Pi p(\varphi, \psi)$, причём функция f строится "пошагово" вслед за изменением одной выделенной переменной – $0, 1, 2, ...$.

Пусть заданы две функции:

$$\varphi = \varphi(x_1, ..., x_n),$$

$$\psi = \psi(x_1, ..., x_n, x_{n+1}, x_{n+2}).$$

Функция $f = f(x_1, ..., x_n, x_{n+1})$ определяется рекуррентным выражением

$$\begin{cases} f(x_1, ..., x_n, 0) = \varphi(x_1, ..., x_n); \\ f(x_1, ..., x_n, y+1) = \psi(x_1, ..., x_n, y, f(x_1, ..., x_n, y)). \end{cases} \tag{38}$$

Выражение (38) называют *схемой примитивной рекурсии.* Эта схема определяет следующий процесс.

Для произвольного набора аргументов $\alpha_1, ..., \alpha_n, \alpha_{n+1}$ сначала полагается

$$f(\alpha_1, ..., \alpha_n, 0) = \varphi(\alpha_1, ..., \alpha_n).$$

Таким образом определяется начальное значение при переменной, по которой происходит рекурсия, изначально равной нулю.

Далее определяются значения

$$f(\alpha_1, ..., \alpha_n, 1) = \psi(\alpha_1, ..., \alpha_n, 0, f(\alpha_1, ..., \alpha_n, 0)),$$

$$f(\alpha_1, ..., \alpha_n, 2) = \psi(\alpha_1, ..., \alpha_n, 1, f(\alpha_1, ..., \alpha_n, 1)),$$

$$\dots$$

$$f(\alpha_1, ..., \alpha_n, y+1) = \psi(\alpha_1, ..., \alpha_n, y, f(\alpha_1, ..., \alpha_n, y)).$$

Наконец, при $y = \alpha_{n+1}$ будет получено значение $f(\alpha_1, ..., \alpha_n, \alpha_{n+1})$, если для всех $y = 0, 1, ..., \alpha_n$ функция $f(\alpha_1, ..., \alpha_n, y)$ определена, либо, в противном случае, значение $f(\alpha_1, ..., \alpha_n, \alpha_{n+1})$ останется не определённым.

В схеме (38) переменные $x_1, ..., x_n$, вообще говоря, могут быть фиктивными или отсутствовать. Но одна переменная – по которой проводится рекурсия – должна быть всегда. В простейших случаях схема примитивной рекурсии может иметь, например, такой вид:

$$\begin{cases} f(0) = a; & // \ a - \textit{константа;} \\ f(y+1) = \psi(y, f(y)). \end{cases}$$

При этом выражение $\psi(y, f(y))$ в частном случае также может быть заменено некоторой константой.

$\mathscr{O}3$) Операция μ – минимизации определяется следующим образом.

Используя заданную функцию g, операция μ порождает функцию f: $f = \mu(g)$. Пусть дана функция $g = g(x_1, ..., x_{n-1}, x_n)$, и $\alpha_1, ..., \alpha_n$ – набор значений переменных $x_1, ..., x_n$.

На основе функции g составляется уравнение

$$g(\alpha_1, ..., \alpha_{n-1}, y) = \alpha_n, \tag{39}$$

где y – неизвестное. Если это уравнение имеет решения, обозначим μ_y – наименьшее (числовое значение) из этих решений.

Если при этом определены все значения

$$g(\alpha_1, ..., \alpha_{n-1}, 0),\ g(\alpha_1, ..., \alpha_{n-1}, 1),\ ...\, g(\alpha_1, ...\ , \alpha_{n-1}, \mu_y - 1), \tag{40}$$

то полагается

$$f(\alpha_1, ...\ , \alpha_{n-1}, \alpha_n) = \mu_y.$$

Если же уравнение (39) не имеет решений или хотя бы одно из значений (40) не определено, то значение функции $f(\alpha_1, ...\ , \alpha_{n-1}, \alpha_n)$ также полагается не определённым.

Часто используют следующее мнемоническое обозначение:

$$f(x_1, ..., x_n) = \mu_y(g(x_1, ..., x_{n-1}, y) = x_n),$$

которое показывает, что функция f получена из функции g путём применения операции μ минимизации на основе составления уравнения (39).

Обозначив

$$h(x_1, ..., x_{n-1}, y, x_n) = |g(x_1, ..., x_{n-1}, y) - x_n|,$$

можно представить операцию минимизации в виде

$$f(x_1, ..., x_n) = \mu_y(h(x_1, ..., x_{n-1}, y, x_n) = 0).$$

Определение 5.1.

Классом примитивно рекурсивных функций называется множество всех функций, которые могут быть получены из базисных функций $0, S, I_m^n$ *путём применения операций суперпозиции* $\mathbf{C}$ *и примитивной рекурсии* Πp.

Классом частично рекурсивных функций называется множество всех функций, которые могут быть получены из базисных функций $0, S, I_m^n$ *путём применения операций суперпозиции* $\mathbf{C}$*, примитивной рекурсии* Πp *и минимизации* μ*.*

Классом общерекурсивных (*или просто рекурсивных*) *функций называется подмножество всех частично рекурсивных функций, которые определены всюду* (*на всех допустимых значениях переменных*)*.*

Согласно определению, для того, чтобы показать, что функция является примитивно рекурсивной, нужно привести последовательность её построения путём использования суперпозиции и примитивной рекурсии.

Покажем примитивную рекурсивность следующих функций.

Константы: $1 = S(0);\ 2 = S(1) = S(S(0)), \ldots$.

Сложение: $f(x_1, x_2) = x_1 + x_2$:

$$\begin{cases} f(x_1, 0) = I_1^1(x_1, x_2); \\ f(x_1, y+1) = S(f(x_1, y)). \end{cases}$$

Для упрощения записи вместо функциональных символов удобно использовать знаки операций, а вместо записи функции выделения переменной использовать просто выделяемую переменную.

Например, схема для сложения может быть записана так:

$$\begin{cases} x_1 + 0 = x_1; \\ x_1 + (y+1) = (x_1 + y) + 1. \end{cases}$$

Умножение:

$$\begin{cases} x_1 \times 0 = 0; \\ x_1 \times (y+1) = x_1 \times y + x_1. \end{cases}$$

Степень двойки:

$$\begin{cases} 2^0 = 1; \\ 2^{y+1} = 2^y \times 2. \end{cases}$$

Усеченная разность: $x_1 \dot{-} x_2$ по определению равна просто разности $x_1 - x_2$, если $x_1 \geq x_2$, и равна нулю, если $x_2 > x_1$. Для того, чтобы показать примитивную рекурсивность усечённой разности, сначала покажем что примитивно рекурсивной является функция $x \dot{-} 1$:

$$\begin{cases} 0 \dot{-} 1 = 0; \\ (x+1) \dot{-} 1 = x. \end{cases}$$

$$\begin{cases} x_1 \dot{-} 0 = x_1; \\ x_1 \dot{-} (x_2 + 1) = (x_1 \dot{-} x_2) \dot{-} 1. \end{cases}$$

Модуль разности:

$$|x_1 - x_2| = (x_1 \dot{-} x_2) + (x_2 \dot{-} x_1).$$

Определим рекурсивные *характеристические функции* $Sg(x)$ и $\overline{Sg}(x)$:

$$Sg(x) = \begin{cases} 0, & x = 0; \\ 1, & x > 0. \end{cases}$$

$$\begin{cases} Sg(0) = 0; \\ Sg(x+1) = 1. \end{cases}$$

$$\overline{Sg}(x) = 1 \dot{-} Sg(x).$$

Покажем частичную рекурсивность следующих функций.

Функция $\left\lfloor \frac{x}{y} \right\rfloor$ – целая часть от деления x на y не определена при $y = 0$. Будем рассматривать вместо неё функцию

$$\left\lfloor \frac{x}{y} \right\rfloor^* = \begin{cases} x, & y = 0; \\ \lfloor \frac{x}{y} \rfloor, & y > 0. \end{cases}$$

При $y > 0$ величина $\lfloor \frac{x}{y} \rfloor$ будет равна наименьшему z такому, что

$$y(z+1) > x \Leftrightarrow y(z+1) \geq x + 1 \Leftrightarrow (x+1) \dot{-} y(z+1) = 0.$$

Поэтому при $y > 0$ значение $z = \lfloor \frac{x}{y} \rfloor$ при любом x может быть определено путём применения операции минимизации

$$\mu_z((x+1)\dot{-}y(z+1) = 0).$$

Если же $y = 0$, то уравнение

$$(x+1)\dot{-}y(z+1) = 0$$

не будет иметь корня: его левая часть будет больше нуля. Чтобы учесть, что при $y = 0$ $\left\lfloor \frac{x}{y} \right\rfloor^* = x$, дополним это уравнение множителем $(x\dot{-}z)$, и тогда

$$\left\lfloor \frac{x}{y} \right\rfloor^* = \mu_z((x\dot{-}z)((x+1)\dot{-}y(z+1)) = 0).$$

Остаток от деления x на $y > 0$:

$$Ocm(x,y) = x\dot{-}y\left\lfloor \frac{x}{y} \right\rfloor^*.$$

Делимость x на y, равная единице, если x делится нацело на y, и равная нулю, если x не делится на y:

$$\mathscr{D}el(x,y) = \overline{Sg}(Ocm(x,y)).$$

Число делителей x, равное 1 при $x = 0$ и числу делителей натурального x при $x > 0$:

$$n\mathscr{D}el(x) = \sum_{y=0}^{x} \overline{Sg}(Ocm(x,y).$$

Чётность x, равная единице, если если x делится нацело на 2, и равная нулю в противном случае – при нечетном x:

$$\mathscr{P}ar(x) = \mathscr{D}el(x,2).$$

Функция $Cm_2(x)$, равная максимальному показателю степени, с которым число 2 входит в разложение на множители числа x; иначе говоря, если $x = 2^t(2y+1)$, то $Cm_2(x) = t$.

Применяя оператор минимизации, можно записать

$$Cm_2(x) = \mu_t(\mathscr{D}el(x, 2^{t+1}) = 0),$$

откуда следует принадлежность функции Cm_2 классу частично рекурсивных функций.

Чётность показателя степени двойки в разложении на множители числа x:

$$\mathscr{P}arCm_2(x) = \mathscr{P}ar(Cm_2(x)).$$

Биективная функция нумерации пар p ставит во взаимно однозначное соответствие паре чисел (x, y) одно число n:

$$p(x, y) = n,$$

и, поскольку p – биекция, должны существовать обратные функции

$$l(n) = x; \quad r(n) = y.$$

Укажем частично рекурсивные функции p, l, r, обеспечивающие биективную нумерацию пар.

$$p(x, y) = 2^x(2y + 1)\dot{-}1;$$

Если $2^x(2y + 1)\dot{-}1 = n$, то

$$x = l(n) = Cm_2(n + 1),$$

и должно выполняться

$$\frac{n+1}{2^{x+1}} = y + \frac{1}{2}.$$

Но y – целое, откуда следует, что

$$y = \left\lfloor \frac{n+1}{2^{x+1}} \right\rfloor,$$

$$y = r(n) = \left\lfloor \frac{n+1}{2^{Cm_2(n+1)+1}} \right\rfloor.$$

При помощи рекурсивной нумерации пар можно получить рекурсивную нумерацию троек, четверок и т.д.:

$$P_3(x,y,z) = p(p(x,y),z); \quad P_4(x,y,z,w) = p(P_3(x,y,z),w).$$

Любая функция $f : \mathbb{N}_0 \times \dots \mathbb{N}_0 \to \mathbb{N}_0$, *равная нулю всюду, за исключением конечного числа точек, является примитивно рекурсивной.*

Действительно, пусть $f(x_1,...,x_n) = f(\tilde{x}) \neq 0$ только в точках $\tilde{a}_1,...,\tilde{a}_j,...,\tilde{a}_k$ и принимает в этих точках значения $y_1,...,y_j,...,y_k$. Тогда

$$f(x_1,...,x_n) = \sum_{j=1}^{k} y_j \times \chi_{\tilde{a}_j}(x_1,...,x_n),$$

где

$$\chi_{\tilde{a}_j}(\tilde{x}) = \begin{cases} 1, & \tilde{x} = \tilde{a}_j; \\ 0, & \tilde{x} \neq \tilde{a}_j; \end{cases}$$

– характеристическая функция точки $\tilde{a}_j = (a_{j1}, a_{j2}, \dots, a_{ji}, \dots, a_{jn})$.

$$\chi_{\tilde{a}_j}(\tilde{x}) = \overline{Sg}(|x_1 - a_{j1}|) \times \dots \overline{Sg}(|x_i - a_{ji}|) \dots \overline{Sg}(|x_n - a_{jn}|).$$

Как следствие, имеем *примитивную рекурсивность матриц*, состоящих из неотрицательных целых чисел, как функций двух переменных на конечном множестве.

5.3 Формальные (аксиоматические) теории

Широко применяемый в математике аксиоматический метод построения теории может быть строго формализован.

Формальная (аксиоматическая) теория $\mathfrak{T}$ *гильбертова типа* определяется совокупностью следующих множеств:

1^o Конечное или счётное множество A символов – *алфавит* теории $\mathfrak{T}$.

Конечные последовательности символов алфавита образуют множество $\mathscr{W}(A)$ выражений (*слов*) теории $\mathfrak{T}$.

2^o Подмножество $\mathscr{F} \subset \mathscr{W}(A)$ выражений теории $\mathfrak{T}$, называемых (правильно построенными) *формулами теории* $\mathfrak{T}$. Обычно построение формул задаётся формально и полагается, что существует *эффективная процедура*[6](алгоритм), позволяющая по каждому выражению теории определить: является оно формулой или нет.

3^o Подмножество $\mathscr{A} \subset \mathscr{F}$ формул, называемых *аксиомами теории* $\mathfrak{T}$. Если существует эффективная процедура, позволяющая установить, является ли данная формула теории аксиомой, то теория $\mathfrak{T}$ называется эффективно *аксиоматизированной* или *аксиоматической* теорией.

4^o Конечное множество $\mathscr{R}_1, ..., \mathscr{R}_n$ отношений конечной местности между формулами, называемое *правилами вывода.*

Если отношение $\mathscr{R}_i$ имеет местность j, и последовательность формул $\varphi_1, \varphi_2, ..., \varphi_{j-1}, \varphi_j$ теории $\mathfrak{T}$ принадлежит отношению $\mathscr{R}_i$, то говорят, что формула φ_j непосредственно *выводится* из формул $\varphi_1, \varphi_2, ..., \varphi_{j-1}$ (является их следствием) по правилу вывода $\mathscr{R}_i$.

Выводом в теории $\mathfrak{T}$ называется такая последовательность формул $\varphi_1, ..., \varphi_k, ...$, в которой для каждого k формула φ_k есть либо аксиома теории $\mathfrak{T}$, либо непосредственное следствие каких-либо предыдущих формул по одному из правил вывода.

Формула Ф называется *теоремой* теории $\mathfrak{T}$, если существует вывод этой формулы Ф в теории $\mathfrak{T}$, в котором последней формулой является Ф; именно такой вывод называется *выводом формулы* Ф.

Согласно данному определению аксиоматической формальной теории, её аксиомы "распознаются" некоторыми алгоритмами. Но для произвольной формулы теории, не являющейся аксиомой, вообще говоря, может не существовать алгоритма, который распознаёт: существует вывод этой формулы в теории, т.е. является ли она теоремой теории. Если это так, то формальная теория называется (алгоритмически) *неразрешимой*; в противном случае, если для любой формулы, не являющейся аксиомой, существует алгоритм, позволяющий установить: является ли она теоремой теории, то эта теория

[6]Эффективной процедурой математики изначально называли именно то, что в настоящее время формализовано как алгоритм тезисом Чёрча-Тьюринга. В современной теории вычислимости эффективными называют алгоритмы, имеющие полиномиальную временную сложность – в отличие от алгоритмов экспоненциальной сложности

называется (алгоритмически) *разрешимой.*

Выводимость формулы Φ (доказуемость теоремы Φ) в теории $\mathfrak{T}$ обозначают $\vdash_{\mathfrak{T}} \Phi$ или, если понятно о какой теории идет речь, $\vdash \Phi$.

Для обозначения выводимости в теории $\mathfrak{T}$ формулы φ_j из последовательности формул $\varphi_1, \varphi_2, ..., \varphi_{j-1}$ используется аналогичная запись $\varphi_1, \varphi_2, ..., \varphi_{j-1} \vdash_{\mathfrak{T}} \varphi_j$ или $\varphi_1, \varphi_2, ..., \varphi_{j-1} \vdash \varphi_j$.

Формальные теории подробно изучаются в курсе математической логики. В рамках излагаемого курса теоретических основ информатики важно проследить логико-математические основания и истоки возникновения формального определения понятия алгоритма, введенного тезисом Чёрча-Тьюринга, и уделить особое внимание понятию алгоритмической неразрешимости, которое изначально возникло при изучении формальных теорий.

Поскольку понятие алгоритма и собственно алгоритмический метод решения задач обработки информации являются центральными в информатике как науке, важно определить возможности и ограничения применения алгоритмического метода. В этом смысле "подвластными" алгоритмическому методу являются только алгоритмически разрешимые проблемы. При этом алгоритмически неразрешимые проблемы нужно и очень важно научиться распознавать, поскольку для них традиционные компьютеры, реализующие исключительно алгоритмы, становятся бесполезными.

Для дальнейшего изложения понадобятся формальная теория L исчисления высказываний и формальная теория предикатов первого порядка. Первая из этих теорий алгоритмически разрешима, а вторая − нет.

Формальная аксиоматическая теория L определяется следующим образом.

1^o Алфавит теории: символы $\neg$, $\supset$, (,) и буквы с индексами $A_1, A_2, ..., A_i, ...$ Круглые скобки, как указано, являются символами алфавита. Обозначение букв алфавита с индексами используется для удобства изложения: буквы можно было бы обозначать $A1, A2, ..., Ai$ или любым другим способом, считая каждую букву отдельным символом.

(Пропозициональные) буквы принципиально отличаются от (примитивных) связок $\neg$, $\supset$ и скобок. Напомним, что набор связок $\{\neg, \supset\}$ является полным, что позволяет выражать конъюнкцию, дизъюнкцию, эквивалентность и все прочие логические функции.

2^o Формулы теории.

a) Любая (пропозициональная) буква есть формула.

b) Если $\mathscr{A}$ и $\mathscr{B}$ – формулы, то $(\neg\mathscr{A})$ и $(\mathscr{A} \supset \mathscr{B})$ тоже формулы.

c) Выражение теории L является формулой в том и только в том случае, когда оно может быть получено при помощи определённых выше пунктов (a) и (b).

3^o Аксиомы теории.

Для любых формул $\mathscr{A}, \mathscr{B}, \mathscr{C}$ теории L аксиомами являются следующие формулы:

a1) $(\mathscr{A} \supset (\mathscr{B} \supset \mathscr{A}))$;

a2) $((\mathscr{A} \supset (\mathscr{B} \supset \mathscr{C})) \supset ((\mathscr{A} \supset \mathscr{B}) \supset (\mathscr{A} \supset \mathscr{C})))$;

a3) $((\neg\mathscr{B} \supset \neg\mathscr{A}) \supset ((\neg\mathscr{B} \supset \mathscr{A}) \supset \mathscr{B}))$.

4^o Правило вывода в теории L только одно: непосредственным следствием формул $\mathscr{A}$ и $(\mathscr{A} \supset \mathscr{B})$ (для любых формул $\mathscr{A}, \mathscr{B}$) является формула $\mathscr{B}$. Это правило называется *modus ponens* [7](MP). Иногда правило MP обозначают так:

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

Напомним, что теория исчисления высказываний строится на основе алгебры логики, в которой буквы обозначают логические переменные, принимающие значения истина или ложь, а пропозициональные связки инверсия $(\neg)$ и импликация $(\supset)$ определяются логическими таблицами истинности.

Любая **теория первого порядка** [8] является расширением теории L – добавлением символов, формул, аксиом, правил вывода. В зависимости от того, какие добавления произведены, фиксируется конкретная теория первого порядка.

Обозначим $\mathfrak{T}_{\mathrm{I}}$ произвольную теорию первого порядка. Для её определения используются все элементы теории L исчисления вы-

[7] modus – способ; ponens – положения, вывода (лат.)

[8] Основным отличительным свойством, определяющим теории первого порядка, является то, что в них допускается навешивание кванторов только на предметные переменные

сказываний.

К *символам теории* L добавляются: знак квантора $\forall$, запятая, счётное множество (предметных) переменных $x_1, x_2, ...$ и предметных констант $a_1, a_2, ...$, конечное или счётное множество предикатных букв A_j^n, где натуральные j и n служат, соответственно, для обозначения номера предикатного символа и местности предиката; конечное или счётное множество функциональных букв f_j^n для обозначения n-местных функций с номерами $j = 1, 2, ...$. В теориях с равенством в алфавит теории включается символ "=".

Формулами теории $\mathfrak{T}_\mathrm{I}$, в дополнение ко всем формулам теории L, объявляются выражения, для построения которых используются *термы*. Терм – это предметная переменная, предметная константа или выражение $f_j^n(t_1, t_2, ..., t_n)$, где $t_1, t_2, ..., t_n$ – термы. Дополнительными формулами теории $\mathfrak{T}_\mathrm{I}$ объявляются выражения:

d1)$A_j^n(t_1, t_2, ..., t_n)$, где A_j^n – предикатный символ, $t_1, t_2, ..., t_n$ – термы (их называют элементарными формулами).

d2) $\forall x_i \mathscr{F}(x_i)$, где x_i – предметная переменная, $\mathscr{F}$ – формула.

Формула $\neg(\forall x_i(\neg\mathscr{F}(x_i))$ может служить определением формулы с квантором существования $\exists x_i \mathscr{F}(x_i)$.

Формулы теории L: Если $\mathscr{A}$ и $\mathscr{B}$ − формулы, то $(\neg\mathscr{A})$ и $(\mathscr{A} \supset \mathscr{B})$ тоже являются формулами. $(\mathscr{A} \vee \mathscr{B}$ является обозначением формулы $((\neg\mathscr{A}) \supset \mathscr{B})$; $(\mathscr{A} \wedge \mathscr{B})$ является обозначением формулы $\neg(\mathscr{A} \supset (\neg\mathscr{B}))$.

Если $\mathfrak{T}_\mathrm{I}$ – теория с равенством, то формулой является выражение

d3) $t_1 = t_2$, где t_1 и t_2 – термы.

Аксиомы теории $\mathfrak{T}_\mathrm{I}$ разделяются на два подмножества – логических и собственных аксиом.

К *логическим* аксиомам относятся аксиомы теории L с добавлением следующих двух аксиом:

a4) $\forall x_i \mathscr{A}(x_i) \supset \mathscr{A}(t)$, где $\mathscr{A}(x_i)$ – формула теории, t – терм, свободный для переменной x_i в формуле $\mathscr{A}(x_i)$ [9]

Поскольку формула $\mathscr{A}(x_i)$ вообще не содержит кванторов, то

[9]Терм t называется свободным для переменной x_i в формуле $\mathscr{A}$, если никакое свободное вхождение x_i в $\mathscr{A}$ не лежит в области действия никакого квантора по переменной, входящей в терм t. Например, терм y свободен для переменной x в формуле $\mathscr{A}(x)$, но тот же терм не свободен для переменной x в формуле $\forall y \mathscr{A}(x)$. См. также примечание [10].

сама переменная x_i является термом, свободным для x_i в формуле $\mathscr{A}(x_i)$, откуда получается аксиома $\forall x_i \mathscr{A}(x_i) \supset \mathscr{A}(x_i)$.

a5) $\forall x_i(\mathscr{A} \supset \mathscr{B}) \supset (\mathscr{A} \supset \forall x_i \mathscr{B})$, если формула $\mathscr{A}$ не содержит свободных вхождений [10] переменной x_i.

Собственные, специфические аксиомы формулируются для каждой теории по-разному. Если теория первого порядка не содержит собственных аксиом, то она называется *исчислением предикатов первого порядка.*

Правила вывода теории первого порядка:

r1) *Modus ponens* (MP): из $\mathscr{A}$ и $\mathscr{A} \supset \mathscr{B}$ следует $\mathscr{B}$.

r2) *Generalization, правило обобщения* (Gen): из $\mathscr{A}$ следует $\forall x_i \mathscr{A}$.

Теории первого порядка часто называют формализованными языками, которые могут описывать различные математические теории. Множество предметных переменных, предметных констант, функциональных символов и предикатных символов называется *сигнатурой* формального языка и является его специфической частью.

Под *интерпретацией* понимается задание (или фиксация) сигнатуры формального языка, а именно:

– задание множества D, называемого *областью* интерпретации;

– задание соответствия, сопоставляющего каждому предикатному символу A_j^n некоторое n-местное отношение $\mathscr{R} \subset D^n$;

– сопоставление каждой предметной константе некоторого элемента из области интерпретации D;

– сопоставление каждой функциональной букве f_j^n некоторой функции $\varphi : D^n \to D$.

В заданной интерпретации любая замкнутая формула [11] является высказыванием, которое истинно или ложно, а любая формула, содержащая свободные переменные, выражает некоторое отношение на области интерпретации. Когда набор значений переменных принадлежит этому отношению, такая формула становится истинной; в

[10] Вхождение переменной в формулу называется связанным, если оно находится в области действия квантора, использующего эту переменную, или же оно является вхождением в этот квантор. Вхождение переменной в формулу называется свободным, если оно не является связанным.

[11] Замкнутой называется формула, не содержащая свободных (не лежащих в поле действия кванторов) переменных.

противном случае – ложной.

Моделью теории первого порядка $\mathfrak{T}_{\mathrm{I}}$ называется такая интерпретация, в которой истинны все аксиомы теории.

Напомним, что теория называется *непротиворечивой*, если в ней не существует формулы $\mathscr{A}$ такой, что $\mathscr{A}$ и $\neg\mathscr{A}$ одновременно являются теоремами (выводимы) в этой теории.

Тавтологией называется тождественно истинная (при любой интерпретации предикатных и пропозициональных переменных) формула. Тавтологии также называют *общезначимыми* формулами.

Основные свойства теорий первого порядка.

1^o Любое исчисление предикатов первого порядка непротиворечиво.

2^o (Теорема Гёделя о полноте) В любом исчислении предикатов первого порядка теоремами являются те и только те формулы, которые общезначимы.

3^o Множество всех выражений любой теории первого порядка счётно.

4^o Любая непротиворечивая теория первого порядка имеет счётную модель (модель со счётной областью D).

5.4 Формальный вывод теорем и алгоритмы. Теорема Гёделя о неполноте формальной арифметики

Гёдель арифметизировал формальную теорию первого порядка $\mathfrak{T}_{\mathrm{I}}$ с равенством путём введения так называемых *гёделевых номеров* – различающихся натуральных чисел – номеров всех символов теории $\mathfrak{T}_{\mathrm{I}}$, затем номеров всех её правильно построенных формул и даже номеров доказательств – последовательностей вывода формул теории. Можно сказать, что Гёдель ввел специальную функцию нумерации, которая взаимно однозначно отображает множество всех символов, выражений и конечных последовательностей выражений во множество натуральных чисел. Любое доказательство (вывод в теории $\mathfrak{T}_{\mathrm{I}}$) при этом также получило уникальный гёделевский но-

мер. Это позволило вместо заданных определением объектов формальной теории "работать" с представляющими их натуральными числами.

Именно теория рекурсивных функций позволяет оперировать с натуральными числами (константами) и арифметическими функциями вида $f : \mathbb{N}_0 \to \mathbb{N}_0$, применяя последовательно элементарные операции суперпозиции, примитивной рекурсии и минимизации (забегая вперед, – рассматриваемые как шаги некоторой алгоритмической модели).

В гёделевской нумерации каждому символу u произвольной теории первого порядка $\mathfrak{T}_{\mathrm{I}}$, включая разделители – скобки запятые и все прочие специальные знаки, – поставлен в соответствие натуральное число $g(u)$, называемое гёделевым номером символа u:

$$g(\ (\ ) = 3;\ g(\ )\ ) = 5;\ g(\ ,\ ) = 7;$$

$$g(\ \neg\ ) = 9; g(\ \supset\ ) = 11;\ g(\forall) = 13;\ g(=) = 15;$$

$$g(x_k) = 9 + 8k; \quad k = 1, 2, ...; \quad // \text{ Номера переменных теории;}$$

$$g(a_k) = 11 + 8k; \quad k = 1, 2, ...; \quad // \text{ Номера констант теории;}$$

$$g(f_k^n) = 13 + 8(2^n 2^k) \quad k, n \geq 1; \quad // \text{ Номера функциональных символов;}$$

$$g(A_k^n) = 15 + 8(2^n 2^k) \quad k, n \geq 1; \quad // \text{ Номера предикатных символов;}$$

Для произвольного выражения $u_0, u_1, ..., u_r$ – последовательности символов теории $\mathfrak{T}_{\mathrm{I}}$ – вводится его гёделевский номер следующим образом:

$$g(u_0, u_1, ..., u_r) = 2^{g(u_0)} 3^{g(u_1)} \cdot \cdot \cdot p_r^{g(u_r)},$$

где $p_0 = 2$, а p_i при $i > 0$ есть i-е простое число.

Различные выражения при такой нумерации (в том числе – для каждого отдельного символа) получат различные номера.

Аналогично через степени простых чисел строится гёделев номер последовательности выражений. Если $v_0, v_1, ..., v_q$ – последовательность выражений, то её гёделев номер есть

$$g(v_0, v_1, ..., v_q) = 2^{g(v_0)} \cdot 3^{g(v_1)} \cdot \ ... \cdot p_q^{g(v_q)}.$$

Вывод в теории – это некоторая последовательность выражений этой теории, поэтому все выводы получают свой гёделевский номер.

Можно заметить, что гёделевские номера символов – нечётные, номера выражений – чётные, а номера последовательностей выражений отличаются от предыдущих конструкций тем, что имеют чётный показатель степени при 2.

Построенная функция g взаимно однозначно отображает множество всех символов, выражений и конечных последовательностей выражений во множество целых положительных чисел.

Обратно, по заданному целому положительному числу можно рекурсивно определить, является ли оно гёделевским номером предметной константы, функциональной или предикатной буквы, номером выражения или последовательности выражений (вывода) и даже определить по гёделевым номерам соответствующие им символы, выражения, выводы.

Для этого можно использовать рекурсивные функции целочисленной делимости, чётности, чётности показателя степени двойки, усечённой разности и другие. Так, например, предикат, определяющий, что число k является последовательностью выражений теории, может быть представлен формулой

$$\mathscr{P}ar(Cm_2(k));$$

предикат, определяющий, что число k есть номер функционального символа – формулой

$$\mathscr{D}el(k\dot{-}13, 8).$$

Номер символа переменной по её гёделевому номеру k можно определить при помощи рекурсивной функции $\lfloor\frac{k\dot{-}9}{8}\rfloor^*$ и т.д.

Заметим, что нумерацию, подобную гёделевской, можно произвести не единственным способом; требуется только взаимная однозначность построения номеров объектов теории.

Используемая далее теория первого порядка S имеет единственный предикатный символ A_1^2, единственную предметную константу a_1, три функциональных символа f_1^1, f_1^2, f_2^2. Предикат A_1^2 выражает равенство, поэтому вместо $A_1^2(t, s)$ можно писать $t = s$. Константа a_1 есть ноль, и записывается как 0. Функциональным символам

f_1^1, f_1^2, f_2^2 устанавливаются в соответствие функции $t^{'}$, $t+s$, $t \cdot s$. Последние три выражения допускается использовать вместо $f_1^1(t)$, $f_1^2(t,s)$, $f_2^2(t,s)$.

Теория S строится на основе аксиом арифметики Пеано, согласно которым:

1) 0 есть натуральное число;

2) для любого натурального числа k существует другое натуральное число $k^{'}$, непосредственно следующее за k;

3) $0 \neq k^{'}$ ни для какого натурального k;

4) Если $k^{'} = m^{'}$, то $k = m$;

5) Пусть $P -$ свойство (одноместный предикат), которым может обладать или не обладать произвольное натуральное число k. Тогда если 0 обладает свойством P ($P(0)$ – истинно) и вместе с тем из истинности $P(k)$ следует истинность $P(k)^{'}$, то $P(k)$ истинно для любого натурального числа k (принцип математической индукции).

Построенные на основе аксиом Пеано, собственные аксиомы теории S имеют следующий вид.

AS1) $k_1 = k_2 \supset (k_1 = k_3 \supset k_2 = k_3)$;

AS2) $k_1 = k_2 \supset k_1^{'} = k_2^{'}$;

AS3) $\forall k(\neg(0 = (k)^{'}))$;

AS4) $k_1^{'} = k_2^{'} \supset k_1 = k_2$;

AS5) $k + 0 = k$;

AS6) $k_1 + k_2^{'} = (k_1 + k_2)^{'}$;

AS7) $k \cdot 0 = 0$;

AS8) $k_1 \cdot (k_2^{'}) = (k_1 \cdot k_2) + k_1$;

AS9) $\mathscr{A}(0) \wedge (\forall k(\mathscr{A}(k) \supset \mathscr{A}(k^{'}))) \supset \forall k \mathscr{A}(k)$, где $\mathscr{A}(k)$ – формула теории S.

Приведем несколько базовых определений, связанных с применением рекурсивных функций к арифметизированной теории S. Напомним, что функции, область определения и множество значений которых состоят из чисел множества $\mathbb{N}_0$, относятся к семейству арифметических функций. Арифметическими называются также отношения и предикаты, заданные на множестве $\mathbb{N}_0$.

$\mathscr{D}$**1.** Отношение $R = R(x_1, ..., x_n)$ называется *примитивно рекурсивным* (*рекурсивным*) отношением, если примитивно рекурсивной (рекурсивной) является его характеристическая функция C_R :

$$C_R(x_1,...,x_n)=\begin{cases}1, & \text{если набор } x_1,...,x_n \text{ принадлежит отношению;}\\ 0, & \text{если набор } x_1,...,x_n \text{ не принадлежит отношению.}\end{cases}$$

В частности, некоторое множество A натуральных чисел является рекурсивным, если рекурсивной является его характеристическая функция $C_A(x)$.

Заметим, что характеристическая функция C_R является предикатом, принимающим значение 1 тогда и только тогда, когда набор $x_1,...,x_n$ принадлежит отношению R. Поэтому *рекурсивные предикаты соответствуют определяемым ими рекурсивным отношениям.*

Отношение $x_1 = x_2$ примитивно рекурсивно: его характеристическая функция есть $\overline{Sg}(|x_1 - x_2|)$.

Отношение $x_1 < x_2$ примитивно рекурсивно: $C_{x_1<x_2} = Sg(x_2 \dot{-} x_1)$.

Отношение делимости $x_1 \vdots x_2$ примитивно рекурсивно: $C_{x_1 \vdots x_2} = \mathscr{D}el(x_1, x_2))$.

Отношение $Prime(x)$ "x – простое число" примитивно рекурсивно: $C_{Prime}(x) = \overline{Sg}(|n\mathscr{D}el(x) - 2| + \overline{Sg}(|x - 1|) + \overline{Sg}(x))$ (число является простым, если оно имеет ровно два делителя и отличается от нуля и единицы).

Функция $PrN(k)$, возвращающая для целого $k \geq 0$ *простое число с этим номером* (в упорядочении по номерам $0, 1, 2...$), является частично рекурсивной:

$$\begin{cases} PrN(0) = 2; \\ PrN(k+1) = \mu_x(\underbrace{Sg((PrN(k)+1)\dot{-}x)}_{x>PrN(k)} + \underbrace{\overline{Sg}(Prime(x))}_{x-\text{простое}} = 0). \end{cases}$$

Любое положительное целое число x, отличное от 0 и 1, однозначно разложимо в произведение степеней простых чисел $p_0, p_1, ..., p_k$:

$$x = p_0^{a_0} p_1^{a_1} ... p_k^{a_k}.$$

Обозначим $(x)_i$ функцию, определяющую значение показателя a_i в указанном разложении. Если $x = 0$, то при любом i положим

$(x)_i = 0$; при $x = 1$ для любого i также положим $(x)_i = 0$.

$$(x)_i = \mu_y(\underbrace{(Sg((x\dot{-}1)\dot{-}y))}_{\mu_y=0 \text{ при } x=0\vee 1}\underbrace{(\overline{Sg}(\mathscr{D}el(x, PrN(i)^y))}_{=0\Leftrightarrow\text{делится на } PrN(i)^y} + \underbrace{Sg(\ \mathscr{D}el(x, PrN(i)^{y+1})\ )}_{=0\Leftrightarrow\text{не делится на } PrN(i)^{y+1}}) = 0).$$

Обозначим $nNE(k)$ число отличных от нуля показателей степеней в разложении натурального k на простые множители и положим $nNE(0) = 0$. Функция $nNE(k)$ примитивно рекурсивна:

$$nNE(k) = \sum_{i\le k} Sg\ (Sg(k) \times \mathscr{D}el(k, i) \times Prime(i))$$

(сумма случаев, когда, не являясь нулём, k делится на простое число i).

Конкатенация выражений. Обозначим $x * y$ выражение

$$2^{a_0} \cdot 3^{a_1} \cdot\ \ldots\ \cdot p_k^{a_k} \cdot p_{k+1}^{b_0} \cdot p_{k+2}^{b_1} \cdot\ \ldots\ \cdot p_{k+m+1}^{b_m},$$

где $x = 2^{a_0} \cdot 3^{a_1} \cdot\ \ldots\ \cdot p_k^{a_k},\ y = 2^{b_0} \cdot 3^{b_1} \cdot\ \ldots\ \cdot p_m^{b_m}$. Тогда $k+1$ можно выразить рекурсивно как $nNE(x),\ m+1$ – как $nNE(y);\ b_j$ как $(y)_j$.

$$x * y = x \times \prod_{j=0}^{nNE(y)-1} p_{nNE(x)+j}^{(y)_j}\ .$$

Правило modus ponens (MP) – выражение с гёделевым номером c непосредственно следует из выражений с гёделевыми номерами a и b формально выражается равенством

$$b = 2^3 * a * 2^{11} * c * 2^5\ .$$

Действительно, если по правилу MP для формул a, b, c имеет место $\frac{a,b}{c}$, то выражение b есть $a \supset c$. Характеристическая функция отношения MP

$$C_{MP}(a, b, c) = |b - 2^3 * a * 2^{11} * c * 2^5|$$

рекурсивна. Поэтому *правило* (отношение) MP *рекурсивно.*

Предикат $OOV(x)$ – x есть *гёделев номер выражения, состоящего только из одной переменной – рекурсивен.* Гёделев номер переменной x есть $9+8k$ для некоторого k, что может быть выражено формулой

$$\exists k(1 \leq k) \wedge (k < x) \wedge (x = 9 + 8k).$$

Соответствующее этой формуле отношение рекурсивно:

$$C_{OOV}(x) = \mu_k(\overline{Sg}(k) + \overline{Sg}(x \dot{-} k) + |x - (9 + 8k)| = 0).$$

Правило $Gen(x, y)$ (генерализации) "выражение с гёделевым номером b непосредственно следует из выражения с гёделевым номером a по правилу генерализации $(\frac{\mathscr{A}}{\forall x \mathscr{A}})$" формально может быть представлено в виде

$$OOV(v) \ \wedge \ (v < b) \wedge \ (b = 2^3 * 2^{13} * v * 2^3 * a * 2^5 * 2^5),$$

где v – гёделев номер любой переменной .

Правило Gen выражает "стандартную практику рассуждений: мы доказываем какое-то утверждение $\mathscr{A}$ со свободной переменной x, после чего заключаем, что мы доказали $\forall x \mathscr{A}$, так как x было произвольным" [2, с. 135]. Но в таком случае переменная x входит в формулу $\mathscr{A}$, и тогда её гёделев номер должен быть меньше гёделевого номера выражения $\mathscr{A}$ (равным быть не может, поскольку x – терм, а не формула). Это объясняет включение в выражение отношения $(v < b)$.

Характеристическая функция отношения Gen рекурсивна:

$$C_{Gen} = OOV(v) \times Sg(b \dot{-} v) \times \overline{Sg}(|b - 2^3 * 2^3 * 2^{13} * v * 2^5 * a * 2^5|).$$

Следовательно *правило* (отношение) Gen *рекурсивно.*

Приведенных выше арифметизированных рекурсивных представлений элементов теории S – символов, выражений, выводов, правил вывода – достаточно, чтобы убедиться в том, что язык теории рекурсивных функций (которые далее будут объявлены формальным определением алгоритмов) позволяет полностью описать теорию S первого порядка с равенством. Для теоретической информатики этот результат является особенно важным, поскольку позволяет убедиться, что вывод теорем в формальных системах осуществляется именно алгоритмами.

$\mathscr{D}$**2.** Арифметическое отношение (предикат) $R(x_1, ..., x_n)$ называется *выразимым в теории* S, если существует формула $\mathscr{A}(x_1, ..., x_n)$ теории S с n свободными переменными такая, что для любых натуральных чисел $k_1, ..., k_n$

1) если $R(k_1, ..., k_n)$ истинно, то формула $\mathscr{A}(\bar{k}_1, ..., \bar{k}_n)$ выводима в теории S;

2) если $R(k_1, ..., k_n)$ ложно, то в теории S выводима формула $\neg\mathscr{A}(\bar{k}_1, ..., \bar{k}_n)$.

Здесь и в дальнейшем для натурального числа k запись $\bar{k}$ обозначает рекурсивное представление этого числа k, т.е. применение k раз к константе ноль функции добавления единицы (корректная запись формул системы S требует использования в них констант в рекурсивном представлении).

$\mathscr{D}$**3.** Арифметическая функция $f(x_1, ..., x_n)$ называется *представимой* в теории S, если существует формула $\mathscr{A}(x_1, ..., x_n, x_{n+1})$ теории S с $n+1$ свободными переменными такая, что для любых натуральных чисел $k_1, ..., k_n, k_{n+1}$ выполняется:

1) если $f(k_1, ..., k_n) = k_{n+1}$, то в теории S выводима формула $\mathscr{A}(\bar{k}_1, ..., \bar{k}_n, \bar{k}_{n+1})$;

2) выводима формула $\exists!x_{n+1}\mathscr{A}(\bar{k}_1, ..., \bar{k}_n, x_{n+1})$.

$\exists!x\mathscr{A}(x)$ означает формулу $(\exists x\mathscr{A}(x)) \wedge ((\forall x\forall y(\mathscr{A}(x)) \wedge \mathscr{A}(y) \supset (x = y)))$.

Гёдель доказал следующие важные утверждения, касающиеся арифметизированной теории S.

$\mathscr{U}$**1.** *Всякая рекурсивная функция представима в теории* S.

𝒴 **2.** *Всякое рекурсивное отношение (предикат) выразимо* в S.

𝒴 **3.** *Рекурсивным является предикат* $W(\mathbf{u}, v)$, *истинный тогда и только тогда, когда* $\mathbf{u}$ *есть гёделев номер формулы* $\mathscr{A}(x_1)$, *содержащей свободную переменную* x_1, *а* v *есть номер вывода в* S *формулы* $\mathscr{A}(\bar{\mathbf{u}})$.

𝒴 **4.** *Всякая функция* $f(x_1, ..., x_n)$, *представимая в* S, *рекурсивна.*

𝒴 **5.** *Всякий заданный на множестве натуральных чисел предикат* $R(x_1, ..., x_n)$ *рекурсивен тогда и только тогда, когда он выразим в теории* S.

Предикат $W(\mathbf{u}, v)$ из утверждения *𝒴* **3** *выразим* в теории S в силу утверждения *𝒴* **2**. Это означает существование формулы $\mathscr{W}(x_1, x_2)$ такой, что из истинности $W(\mathbf{u}, v)$ следует *выводимость формулы* $\mathscr{W}(\bar{\mathbf{u}}, \bar{v})$ в теории S, а из истинности $\neg W(\mathbf{u}, v)$ следует выводимость формулы $\neg\mathscr{W}(\bar{\mathbf{u}}, \bar{v})$ в теории S.

Формула

$$\forall x_2 \neg \mathscr{W}(x_1, x_2)$$

используется далее для построения формулы, невыводимой в теории S.

Для доказательства представленной ниже теоремы Гёдель построил следующую *невыводимую* (*недоказуемую*) в теории S замкнутую формулу [12]

$$\Psi = \forall x_2 \neg \mathscr{W}(\bar{m}, x_2) \tag{41}$$

арифметической формальной системы S, где:

m – гёделев номер формулы $\forall x_2 \neg \mathscr{W}(x_1, x_2)$, свободной по первой переменной;

$\mathscr{W}(x_1, x_2)$ – формула, выражающая в S предикат $W(x_1, x_2)$ и выводимая тогда и только тогда, когда $x_1 = u$ есть гёделев номер вывода некоторой формулы $\mathscr{A}(x_1)$, содержащей свободную переменную x_1 в S, а $x_2 = v$ есть номер вывода в S формулы $\mathscr{A}(\bar{u})$.

Формула $\mathscr{W}(\bar{m}, \bar{v})$ выводима тогда и только тогда, когда v есть гёделев номер вывода в системе S формулы $\forall x_2 \neg \mathscr{W}(x_1, x_2)$. Можно сказать, что формула $\mathscr{W}(\bar{m}, \bar{v})$ в определённом смысле выражает

[12] Замкнутость формулы с кванторами означает отсутствие в ней свободных переменных – находящихся вне действия некоторого квантора

применимость некоторой формулы к её собственному гёделевому номеру и говорить в этом смысле о самоприменимости этой формулы.

Формальная теория (формальная система) называется *непротиворечивой*, если в ней не существует замкнутой формулы $\mathscr{A}$ такой, что $\mathscr{A}$ и $\neg\mathscr{A}$ одновременно выводимы в этой теории – не существует *неразрешимого предложения*. Если выводима либо замкнутая формула $\mathscr{A}$, либо $\neg\mathscr{A}$, то формула $\mathscr{A}$ называется *разрешимым предложением*. Формальную систему называют *полной*, если в ней не существует неразрешимого предложения, иначе – называют *неполной*.

Теорема 5.1 (теорема Гёделя для теории S)**.**

Если формальная система (*теория*) S *непротиворечива, то формула* Ψ *невыводима в* S.

Доказательство. Предположим, что теория S является непротиворечивой и в ней выводима формула $\forall x_2\neg\mathscr{W}(\bar{m},x_2)$. Пусть k – гёделев номер какого-нибудь вывода в теории S формулы $\forall x_2\neg\mathscr{W}(\bar{m},x_2)$. Тогда, используя логическую аксиому теории S о замене переменной термом

$$(\forall x_2\neg\mathscr{W}(\bar{m},x_2)) \supset (\neg\mathscr{W}(\bar{m},\bar{k})),$$

и применяя правило *modus ponens*

$$\frac{\forall x_2\neg\mathscr{W}(\bar{m},x_2),\ (\forall x_2\neg\mathscr{W}(\bar{m},x_2)) \supset (\neg\mathscr{W}(\bar{m},\bar{k}))}{\neg\mathscr{W}(\bar{m},\bar{k})},$$

из формулы $\forall x_2\neg\mathscr{W}(\bar{m},x_2)$ *можно вывести формулу* $\neg\mathscr{W}(\bar{m},\bar{k})$.

В то же время, поскольку предикат $W(\bar{m},\bar{k})$ является истинным, а формула $\mathscr{W}$ выражает предикат W, должна быть *выводимой формула* $\mathscr{W}(\bar{m},\bar{k})$.

Это приводит к противоречию: выводимости одновременно формул $\mathscr{W}(\bar{m},\bar{k})$ и $\neg\mathscr{W}(\bar{m},\bar{k})$. Поэтому либо система S противоречива, либо формула (41) невыводима. $\square$

Таким образом, если система S непротиворечива, то она неполна (содержит невыводимую формулу), и Ψ служит примером такой неразрешимой формулы.

Формулу Ψ (41) иногда называют *гёделевой неразрешимой формулой*.

Формальная теория первого порядка S, основанная на аксиомах Пеано и достаточно подробно рассмотренная выше, позволяет вывести все основные результаты элементарной арифметики. Поэтому она представляет собой один из вариантов построения теории Ar, называемой *формальной арифметикой*. Это позволяет переформулировать предыдущую теорему в следующем виде:

Теорема 5.2 (теорема Гёделя о неполноте формальной арифметики)**.**

Если формальная арифметика (*система* Ar) *непротиворечива, то она неполна.*

Развивая свои исследования, Гёдель показал, что, исключая простейшие случаи, доказательство непротиворечивости формальной теории невозможно провести средствами одной лишь этой теории.

Более того, Гёдель доказал неполноту гильбертовой системы, обнаружив внутри неё ряд неразрешимых проблем – одной из них и является проблема непротиворечивости. Этим было показано, что возможности аксиоматического метода ограничены, причём даже обычная арифметика натуральных чисел не может быть полностью аксиоматизирована. Отсюда следует, что процесс математического доказательства не всегда сводится к использованию аксиоматического метода, и возможности человеческого мышления гораздо шире.

5.5 Алгоритмическая модель (машина) Тьюринга

В своей работе "*On Computable Numbers, with an Application to the Entscheidungsproblem*"(О вычислимых числах с приложением к проблеме разрешения), опубликованной 12 ноября 1936 года, Тьюринг переформулировал теорему Гёделя о неполноте, *заменив универсальный формальный арифметический язык Гёделя на другой формальный язык алгоритмической модели, которая впоследствии стала известна как машина Тьюринга. Он предположил, что его машина способна произвести любые математические вычисления, представимые в виде алгоритма.* Далее Тьюринг показал, что не

существует решения проблемы *Entscheidungsproblem* (проблемы разрешимости), предварительно доказав, что проблема остановки для машины Тьюринга неразрешима: в общем случае невозможно алгоритмически определить, остановится ли когда-нибудь данная машина Тьюринга, начав работу над произвольным заданным входным словом.

Подход Алана Тьюринга является более доступным и конструктивным, чем гёделевский подход. Идея "Универсальной Машины", которая способна выполнять функции любой другой машины или, другими словами, вычислить всё, что можно в принципе вычислить, была особенно интересна математикам. Джон фон Нейман признал, что концепция современного компьютера основана на упомянутой выше работе Алана Тьюринга.

Рис. 5.28: Выдающийся английский математик Алан Тьюринг (1912-1954)

Машины Тьюринга по-прежнему являются основным инструментом исследований в теории алгоритмов.

С сентября 1936 года по июль 1938 Тьюринг работал под руководством Алонзо Чёрча в Принстоне, где в июне 1938 года защитил докторскую диссертацию "Логические системы, основанные на ординалах".

Алгоритмическая модель – машина Тьюринга может быть определена следующим образом.

Пусть $A = \{a_0, ..., a_{n-1}\}$ – конечный алфавит, $S(A)$ – множество слов, которые могут быть составлены из букв алфавита A; $Q = \{q_0, ..., q_{r-1}\}$ – так называемый конечный алфавит *памяти*, или алфавит *состояний*. Модель Тьюринга является динамической и описывается при помощи динамической переменной *времени* или *шага* $t = 0, 1, 2, ...$. На каждом шаге машина Тьюринга может находиться ровно в одном состоянии из алфавита Q. Состояние, в котором находится машина Тьюринга на шаге 0, называется *начальным*, и если после выполнения некоторого конечного числа шагов машина останавливается, то состояние, в котором она останавливается

называется *конечным*.

Далее для сокращения определения будем обозначать машину Тьюринга MT. Любая MT реализует отображение (алфавитную функцию) $F_{MT} : S(A) \to S(A)$, то есть, если задано входное слово $X = (x_1, ..., x_n)$ из множества $S(A)$, то MT ставит в соответствие этому слову единственное слово $Y = (y_1, ..., y_n)$ из множества $S(A)$.

Предполагается, что задана бесконечная пустая строка (ее называют также *лентой*), которая нумеруется *индексами* или номерами *ячеек* ленты. Пустой называется строка, все символы которой являются некоторым выделенным в алфавите A *пустым* символом Λ. Не теряя общности, можно нумеровать бесконечную строку-ленту, начиная с произвольно выбранной нулевой позиции (индекса) влево (отрицательными) и вправо (положительными) целыми индексами:

$$\begin{pmatrix} \text{Символы: } ... & \Lambda & \Lambda & \Lambda & \Lambda & \Lambda & \Lambda & ... \\ \text{Индексы: } ... & -2 & -1 & 0 & 1 & 2 & 3 & ... \end{pmatrix}.$$

Машина Тьюринга на каждом шаге "обозревает" одну ячейку ленты в том смысле, что фиксируется на некотором индексе i записанной на ленту строки $(\alpha_1, ..., \alpha_i, ..., \alpha_s) \in S(A)$. В такой ситуации говорят, что МТ обозревает ячейку с номером i, "видит" в ней символ α_i, находясь при этом в состоянии q_t (имея в памяти символ $q_t \in Q$).

$$\begin{pmatrix} ... & \Lambda & \Lambda & \Lambda & \alpha_1 & \alpha_2 & ... & \alpha_i & \alpha_{i+1} & ... & \alpha_s & ... \\ ... & -2 & -1 & 0 & 1 & 2 & ... & i & i+1 & ... & s & ... \\ & & & & & & & \Delta & & & & \\ & & & & & & & q_t & & & & \end{pmatrix}.$$

Полагается, что в начальном состоянии МТ обозревает именно первый символ заданного входного слова – представленной для обработки строки. Все символы ленты слева и справа от заданной строки полагаются пустыми.

Если на ленте записана некоторая строка символов длины s, и машина Тьюринга на шаге t в состоянии q_t обозревает ячейку с номером i, то это можно представить так называемой *конфигурацией* МТ на шаге t:

$$K_t = \alpha_1\alpha_2...\alpha_{i-1}q_t\alpha_i\alpha_{i+1}...\alpha_s.$$

Конфигурация, таким образом, это строка, в которую вставлен символ состояния МТ, указывающий на обозреваемый, непосредственно примыкающий к нему справа символ входной строки.

Логика одного шага работы МТ заключается в следующем. Находясь в конфигурации K_t и в зависимости только от α_i и q_t МТ

а) считывает символ α_i;

в) записывает на его место другой символ α_i^* входного алфавита A (или, в частном случае, тот же самый, т.е. оставляемый без изменения символ);

с) переходит в новое состояние – символ алфавита Q (или, в частном случае, не меняет своё состояние);

d) осуществляет сдвиг по ленте-строке влево или вправо, уменьшая или увеличивая на единицу номер обозреваемой ячейки (в частном случае – не изменяя позиции, что соответствует нулевому сдвигу).

Один шаг работы МТ, таким образом, описывается уравнениями

$$\begin{cases} \alpha_i^* = F(\alpha_i, q_{t-1}); \\ q_t = G(\alpha_i, q_{t-1}); \\ \Delta(i) = D(\alpha_i, q_{t-1}), \end{cases}$$

где $(t-1)$ – предыдущий, а t – последующий номера шагов; q_t – новое состояние, в которое перейдет МТ из состояния q_t; а $\Delta(i)$ – сдвиг номера ячейки (индекса), который может принимать три значения: $\{-1, 0, +1\}$. Новый индекс определяется сдвигом в соответствии с операцией присваивания

$$i := i + \Delta(i).$$

Поскольку алфавиты A и Q конечны, МТ полностью описывается шестеркой

$$< A, Q, q_0 \in Q, F, G, D >,$$

где q_0 – начальное состояние МТ, а F, G и D – бинарные операции или задаваемые соответствующими матрицами функции

$$\begin{cases} F : A \times Q \to A; \ (\text{ Функция выходов }) \\ G : A \times Q \to Q; \ (\text{ Функция переходов }) \\ D : A \times Q \to \{-1, 0, +1\}. \ (\text{ Функция сдвигов }) \end{cases}$$

Можно заметить, что МТ на каждом шаге выполняет локальное преобразование текущей строки – в окрестности только одного символа. Это позволяет задать ещё один формализм – текущую выполняемую команду:

$$C_t = q_{t-1}\alpha_i q_t \alpha_i^* \Delta(i),$$

которую нужно понимать следующим образом: находясь в состоянии q_{t-1} на шаге преобразования текущей строки и обозревая в позиции (ячейке) с номером i символ α_i, МТ переходит в состояние q_t, перезаписывает вместо α_i символ α_i^* и сдвигается на позицию ленты $i := i + \Delta(i)$. При $t = 1$ машина Тьюринга начинает работать, находясь в начальном состоянии q_0. Если некоторая команда МТ переводит её в конечное (завершающее) состояние, то МТ останавливается, и результатом её работы полагается записанное на ленте слово.

Таблица 1: Таблица выходов, определяющая значения функции $F : A \times Q \to A$

Вход \ *Состояние*	q_0	q_1	...	q_j	...	q_{r-1}
a_0	y_{00}	y_{01}	...	y_{0j}	...	$y_{0,(r-1)}$
a_1	y_{10}	y_{11}	...	y_{1j}	...	$y_{1,(r-1)}$
...	...	...	...	...	...	...
a_i	y_{i0}	y_{i1}	...	y_{ij}	...	$y_{i,(r-1)}$
...	...	...	...	...	...	...
a_{n-1}	$y_{(n-1),0}$	$y_{(n-1),1}$	...	$y_{(n-1),j}$	...	$y_{(n-1),(r-1)}$

Таблица 1 представляет функцию выходов в табличном виде: $y_{i,j} = F(a_i, q_j)$. Она описывается $n \times r$ значениями символов, записываемых на ленту в каждой из $n \times r$ ситуаций обработки информации на отдельном шаге МТ.

Таблица 2 имеет такую же структуру и представляет функцию переходов: $\delta_{i,j} = G(a_i, b_j)$. Таблица 3 определяет сдвиг по обрабатываемому слову (ленте) влево (-1), вправо $(+1)$ или, как частный случай, – нулевой сдвиг, который используется на очень коротких циклах локальной обработки слов.

Если задана строка, подлежащая обработке, то полагается, что стандартное начальное положение МТ таково, что первый символ

Таблица 2: Таблица переходов, определяющая значения функции $G : A \times Q \to Q$

Вход \ *Состояние*	q_0	q_1	...	q_j	...	q_{r-1}
a_0	δ_{00}	δ_{01}	...	δ_{0j}	...	$\delta_{0,(r-1)}$
a_1	δ_{10}	δ_{11}	...	δ_{1j}	...	$\delta_{1,(r-1)}$
...	...	...	...	...	...	...
a_i	δ_{i0}	δ_{i1}	...	δ_{ij}	...	$\delta_{i,(r-1)}$
...	...	...	...	...	...	...
a_{n-1}	$\delta_{(n-1),0}$	$\delta_{(n-1),1}$	...	$\delta_{(n-1),j}$	...	$\delta_{(n-1),(r-1)}$

этой строки расположен в ячейке с номером 1, МТ находится в состоянии q_0, и тогда достаточно только автоматически определять по указанным трем таблице вычислительный процесс.

Таблица 3: Таблица сдвигов, определяющая значения функции $D : A \times Q \to \{-1, 0, +1\}$

Вход \ *Состояние*	q_0	q_1	...	q_j	...	q_{r-1}
a_0	Δ_{00}	Δ_{01}	...	Δ_{0j}	...	$\Delta_{0,(r-1)}$
a_1	Δ_{10}	Δ_{11}	...	Δ_{1j}	...	$\Delta_{1,(r-1)}$
...	...	...	...	...	...	...
a_i	Δ_{i0}	Δ_{i1}	...	Δ_{ij}	...	$\Delta_{i,(r-1)}$
...	...	...	...	...	...	...
a_{n-1}	$\Delta_{(n-1),0}$	$\Delta_{(n-1),1}$	...	$\Delta_{(n-1),j}$	...	$\Delta_{(n-1),(r-1)}$

Можно совместить три указанные выше таблицы, описывающие работу МТ, в одну, и тогда строке a_i и столбцу q_j будет соответствовать клетка таблицы с записанной в ней объединённой информацией $y_{i,j}\delta_{i,j}\Delta_{i,j}$. Тогда команду МТ, выполняемую на одном шаге её работы можно записать так:

$$C_{i,j} = a_i q_j y_{i,j} \delta_{i,j} \Delta_{i,j},$$

а все команды можно упорядочить: $i = 0, 1, ..., n-1;\ j = 0, 1, ..., r-1$, получив конечную программу МТ из $n \times r$ команд, точно и одно-

значно описывающую конкретную МТ:

$$\{C_{i,j};\ \ i = 0, 1, ..., (n-1);\ j = 0, 1, ..., (r-1)\}.$$

Определение 5.2. *Две машины Тьюринга*

$$< A, Q, q_0 \in Q, F, G, D >,\ A = \{a_0, ..., a_{n-1}\}, Q = \{q_0, ..., q_{r-1}\},\ и$$

$$< \mathcal{A}, \mathcal{Q}, g_0 \in \mathcal{Q}, \mathcal{F}, \mathcal{G}, \mathcal{D} >,\ \mathcal{A} = \{\alpha_0, ..., \alpha_{n-1}\}, \mathcal{Q} = \{g_0, ..., g_{r-1}\}$$

называются изоморфными, если существуют биекции

$$\varphi : A \to \mathcal{A}\ \ и\ \ \psi : Q \to \mathcal{Q}$$

такие, что для всех допустимых пар символов a_i, b_j

$$g_0 = \psi(q_0),$$

$$\mathcal{F}(\varphi(a_i), \psi(g_j)) = \varphi(F(a_i, q_j)),$$
$$\mathcal{Q}(\varphi(a_i), \psi(g_j)) = \psi(Q(a_i, q_j)),$$
$$\mathcal{D}(\varphi(a_i), \psi(g_j)) = D(a_i, q_j).$$

Далее можно рассматривать класс МТ с точностью до изоморфизма, основанного на взаимно однозначном переименовании переменных.

Поскольку отношение изоморфизма на парах машин Тьюринга, что очевидно, – рефлексивно, симметрично и транзитивно, оно порождает разбиение множества всех МТ на классы эквивалентности. Далее, изучая машины Тьюринга, из каждого класса эквивалентных изоморфных МТ можно рассматривать только одну, применяя универсальное кодирование символов ее алфавитов.

Применим следующую общую кодировку для машин Тьюринга (таблица 4), в соответствии с которой все символы, используемые для описания МТ, можно закодировать попарно различными *натуральными числами*, начинающимися с единицы. Если a – некоторый символ, то его шифр – соответствующее натуральное число – будет обозначаться $\mathbb{C}(a)$. Например, шифр состояния q_0 будет натуральным числом $\mathbb{C}(q_0) = 100000$. Определим шифр одной команды МТ

$$C_{i,j} = a_i q_j y_{i,j} \delta_{i,j} \Delta_{i,j}$$

как конкатенацию входящих в неё символов:

$$\mathbb{C}(C_{i,j}) = \mathbb{C}(a_i)\mathbb{C}(q_j)\mathbb{C}(y_{i,j})\mathbb{C}(\delta_{i,j})\mathbb{C}(\Delta_{i,j})$$

и, наконец, программу МТ полностью –

$$\{C_{i,j};\ \ i = 0, 1, ..., (n-1);\ j = 0, 1, ..., (r-1)\}$$

как конкатенацию всех её команд:

$$\mathbb{C}(MT) = \mathbb{C}(C_{1,1})\mathbb{C}(C_{1,2})...\mathbb{C}(C_{(n-1),(r-1)}). \qquad (42)$$

Теорема 5.3. *Число неизоморфных машин Тьюринга не более чем счётно.*

Доказательство. Согласно универсальному кодированию, машине Тьюринга из каждого класса изоморфизма поставлено в соответствие натуральное число (42), составленное из единиц и нулей и начинающееся с единицы, что доказывает теорему. □

Таблица 4: Таблица цифровых кодов символов МТ

Типы символов	*Символ*	*Шифр символа*	*Число нулей в коде символа*
Символы сдвига	-1	10	1
	0	100	2
	+1	1000	3
Входной алфавит	a_0	10000	4
	a_1	1000000	6
	...	...	...
	a_i	10...00	$2i+4$
	...	...	...
Алфавит состояний	q_0	100000	5
	q_1	10000000	7
	...	...	...
	q_j	10...000	$2j+5$
	...	...	...

5.6 Машины Тьюринга и рекурсивные функции

Пример универсального кодирования машин Тьюринга при помощи шифров показывает, что вместо произвольных алфавитов для описания тьюринговской модели можно использовать натуральные числа. Также можно рассмотреть входное слово на ленте МТ как набор натуральных чисел – аргументов, а выходное слово – как натуральное число, являющееся выходным словом. В этом смысле *МТ реализует или, что более привычно математикам, вычисляет некоторую функцию.*

Определение 5.3. *Функция называется вычислимой по Тьюрингу, если существует МТ, которая, начав работу над словом на ленте, представляющем допустимое значение аргумента этой функции, за конечное число шагов выдаст на ленту значение функции от заданного аргумента и остановится.*

Естественно возникает вопрос: отличается ли класс функций, вычислимых по Тьюрингу, от класса частично рекурсивных функций, которые были предложены Эрбраном и Гёделем и использованы для формализации процессов доказательства теорем в формальных теориях?

Оказывается, что эти классы совпадают (*см. теоремы 5.4 и 5.5*) *, и установление этого факта исторически можно считать отправной точкой развития новой науки – алгоритмической или компьютерной математики.*

Если говорить о любой математической задаче, теореме, алгоритме, то трудно представить иную форму её задания, чем пару предложений на некотором языке, первое из которых (условно) начинается словом "*дано*" и последовательностью символов, которая описывает условие, а второе – словом "*найти*" и последовательностью символов, описывающей что нужно сделать: доказать, отыскать корень уравнения или другое.

Алан Тьюринг предложил именно такую модель, а её эквивалентность классу частично рекурсивных функций усилила значение следующих предложений:

Тезис Тьюринга. *Класс задач, которые можно решить при помощи алгоритмов, понимаемых в интуитивном смысле, совпа-*

дает с классом задач, которые могут быть решены машиной Тьюринга.

Тезис Чёрча. *Класс задач*[13]*, которые можно решить при помощи алгоритмов, понимаемых в интуитивном смысле, совпадает с классом частично рекурсивных функций.*

Определение 5.4. *Вычислительной проблемой называется совокупность однотипных (индивидуальных) задач, отличающихся друг от друга только допустимой вариацией начальных данных.*

Например, проблема разрешимости диофантовых уравнений предполагает установление существования или несуществования алгоритма решения произвольного диофантова уравнения. Известно, что для некоторых частных случаев диофантовых уравнений (индивидуальных задач) можно указать алгоритмы их решения, но в общем случае, для любого диофантова уравнения, решающего алгоритма не существует. Это доказал в 1970 г. советский математик Ю. В. Матиясевич.

Определение 5.5. *Проблема называется алгоритмически разрешимой, если существует машина Тьюринга, при помощи которой можно получить решение каждой индивидуальной задачи этой проблемы.*

Теорема 5.4. *Если функция является частично рекурсивной, то она вычислима по Тьюрингу.*

Доказательство. Сначала изложим идею доказательства. Любая частично рекурсивная функция по определению порождается из базисных функций путём применения операций суперпозиции, примитивной рекурсии и минимизации. Поэтому, если показать, что на МТ реализуемы константа ноль, функция следования, функция выделения аргумента, а также что операции $C, \Pi p, \mu$ можно реализовать на МТ, то теорема будет доказана.

1. *Базисные функции.* Представление константы *ноль* на ленте МТ и программу прибавления единицы к числу на ленте даже не нужно обсуждать.

Если на ленте записан набор из n натуральных чисел, то "подвод" головки МТ к m-му числу со стиранием предыдущих $m-1$

[13]задач как отображений "исходные данные $\rightarrow$ результат"

чисел, пропуск m-го числа и стирание всех остальных – очевидно, позволяет получить функцию $I_m^n(x_1, ..., x_m, ..., x_n) = x_m$.

2. *Суперпозиция* функций реализуется на машинах Тьюринга следующим образом. Первая МТ вычисляет результат, соответствующий первой функции, и записывает его на ленту. Вторая МТ (соответствующая второй функции суперпозиции $f_1 \circ f_2$) имеет своим начальным состоянием конечное состояние первой МТ и в качестве входного слова получает результат первой МТ.

3. На машинах Тьюринга реализуемо *ветвление* (условный переход). Действительно, пусть $U : X \to \{0, 1\}$ – произвольное условие (предикат). Пусть три машины Тьюринга – МТ1, МТ2, МТ3 таковы, что МТ1 вычисляет $U(x)$ по входному слову x, выдаёт на ленту значение σ, равное 1 или 0 в зависимости от выполнения или невыполнения условия, и переходит на начальное состояние машины МТ2, если $\sigma = 1$, или на начальное состояние МТ3, если $\sigma = 0$.

4. На машинах Тьюринга реализуем *цикл*. Действительно, цикл организуется путём применения ветвления, когда по надлежащему условию осуществляется переход либо на МТ2, либо снова на МТ1.

5. *Операция примитивной рекурсии* Пp реализуема на МТ.

Для упрощения записи доказательства сначала рассмотрим операцию Пp только в простейшем случае:

$$\begin{cases} f(0) = a; \\ f(y+1) = \varphi(y, f(y)), \end{cases}$$

где f – определяемая операцией Пp функция; a – начальное значение; y – переменная, по которой производится рекурсия; φ – функция, используемая для шагов выполнения рекурсии.

Пусть требуется вычислить $f(x)$. Пусть МТ1 заносит на ленту три целых неотрицательных числа (блока): $0 \smile a \smile x$, где $\prime \smile \prime$ – некоторый разделитель. Обозначим эти три блока $B1, B2, B3$.

Определим следующие МТ.

МТ2 вычисляет функцию φ так: $B2 := \varphi(B1, B2)$.

МТ3 добавляет единицу к $B1$.

МТ4 вычитает единицу из $B3$.

МТ5 реализует ветвление по значению $B3$: переход на М6, если $B3 = 0$, иначе – на МТ2.

МТ6 стирает блоки $B1$, $B3$ и возвращает результат в блоке $B2$.

В общем случае операция Πp представляется схемой

$$\begin{cases} f(x_1, ..., x_n, 0) = \psi(x_1, ..., x_n); \\ f(x_1, ..., x_n, y+1) = \varphi(x_1, ..., x_n, y, f(x_1, ..., x_n, y)). \end{cases}$$

Можно заметить, что рекурсия всегда выполняется только по одной переменной, а переменные $x_1, ..., x_n$ остаются без изменений, следовательно, могут просто сохраняться на ленте. А для вычисления $\psi(x_1, ..., x_n)$ потребуется дополнительная МТ7.

6. *Операция минимизации* μ реализуема на МТ. Рассмотрим операцию минимизации $f = \mu(g)$, применяемую к заданной функции g одной переменной:

$$f(x) = \mu_y(g(y) = x).$$

Минимизация всегда производится по одной переменной y. Поэтому ее можно реализовать следующим образом: начиная от значения $y = 0$ и добавляя к нему последовательно по единице, проверять условие $g(y) = x$. Если это условие выполнится, то найденное значение y будет искомым значением для $f(x)$.

Пусть МТ1 заносит на ленту три целых неотрицательных числа (блока): $0 \smile g(0) \smile x$, где $/ \smile /$ – некоторый разделитель. Обозначим эти три блока $B1, B2, B3$.

Машина МТ2 добавляет к блоку $B1$ единицу.

Машина МТ3 вычисляет значение функции $g(B1)$ и заносит его в $B2$.

Машина МТ4 вычисляет предикат $(B2 = B3)$.

Машина МТ5 реализует переход по условию $B2 = B3$ на МТ6 или, в противном случае, возврат на МТ2.

Машина МТ6 возвращает $B1$ как результат $f(x)$.

Реализация операции μ в общем случае, согласно выражению

$$f(x_1, ..., x_n) = \mu_y(g(x_1, ..., x_{n-1}, y) = x_n),$$

отличается только дополнительным размещением на ленте переменных, которые не изменяются в процессе выполнения операции.

Теорема 5.5. *Если функция вычислима по Тьюрингу, то она является частично рекурсивной.*

Доказательство. Идея состоит в том, чтобы показать: процесс переработки информации любой данной машиной Тьюринга эквивалентен вычислению некоторой частично рекурсивной функции.

Будем считать, что кодирование всех символов, используемых при определении МТ, осуществляется натуральными числами.

Рассмотрим уравнения, описывающие шаги работы произвольной МТ с начальным состоянием $q0$ и конечным состоянием q_{stop}.

$$\begin{cases} q_0 = q0; & // \textit{ начальное состояние;} \\ \alpha_0 = x_1; & // \textit{ первый символ входного слова;} \\ \Delta(0) = 0; & // \textit{ начальный сдвиг – ноль;} \\ i = 1; & // \textit{ начальная позиция на ленте – ячейка номер 1;} \\ \alpha_i^* = F(\alpha_i, q_{t-1}); \\ q_t = G(\alpha_i, q_{t-1}); \\ \Delta(i) = D(\alpha_i, q_{t-1}); \\ i = i + D(\alpha_i, q_{t-1}). \end{cases}$$

Введем фиктивные переменные для функций F, G, D и функцию $\mathscr{I}$, вычисляющую номер i:

$$\begin{cases} q_0 = q0; & // \textit{ начальное состояние;} \\ \alpha_0 = x_1; & // \textit{ первый символ входного слова;} \\ \Delta(0) = 0; & // \textit{ начальный сдвиг – ноль;} \\ i = 1; & // \textit{ начальная позиция на ленте – ячейка номер 1;} \\ \alpha_i^* = F(\alpha_i, q_{t-1}, \Delta(i-1), i-1); \\ q_t = G(\alpha_i, q_{t-1}, \Delta(i-1), i-1); \\ \Delta(i) = D(\alpha_i, q_{t-1}, \Delta(i-1), i-1); \\ i = \mathscr{I}(\alpha_i, q_{t-1}, \Delta(i-1), i-1). \end{cases}$$

Сравним полученные уравнения со схемой примитивной рекурсии

$$\begin{cases} f(x_1, ..., x_n, 0) = \psi(x_1, ..., x_n); \\ f(x_1, ..., x_n, y+1) = \varphi(x_1, ..., x_n, y, f(x_1, ..., x_n, y)). \end{cases}$$

Основное отличие состоит в числе уравнений: четыре – в схеме работы МТ, а схеме Πp – только одно (уравнения, задающие начальные

условия, в этом подсчёте не учитываются).

Сведем рекурсивно четыре уравнения, описывающие работу МТ, к одному.

Известно, например, что рекурсивной является биекция нумерации пар (см. стр. 64-65)

$$n \xrightarrow{P} (x, y),$$

где функция P ставит в соответствие паре (x, y) число n:

$$P(x, y) = 2^x(2y + 1)\dot{-}1,$$

и существуют обратные функции p_1 и p_2, которые позволяют по значению n найти элементы пары x и y:

$$x = Ct_2(n + 1) = p_1(n),$$

$$y = \left[\frac{n + 1}{2^{Ct_2(n+1)+1}}\right] = p_2(n).$$

Имея рекурсивную биекцию для пар, легко построить рекурсивную биекцию для троек и четверок:

$$P_3(x, y, z) = P(P(x, y), z);$$

$$P_4(x, y, z, u) = P(P_3(x, y, z), u) = P(P(P(x, y), z), u).$$

Если $\eta = P_4(x, y, z, u)$, то $u = p_2(\eta); P(P(x, y), z) = p_1(\eta)$; $z = p_2(p_1(\eta))$; $P(x, y) = p_1(p_1(\eta))$; $x = p_1(p_1(p_1(\eta)))$; $y = p_2(p_1(p_1(\eta)))$.

Таким образом для четверок (x, y, z, u) получена рекурсивная биекция $\mathscr{P} = P_4$ и соответствующие ей обратные функции, которые мы обозначим $\gamma_1, \gamma_1, \gamma_3, \gamma_4$, такие, что при $\mathscr{P}(x, y, z, u) = \eta$

$$\gamma_1(\eta) = x;\ \gamma_2(\eta) = y;\ \gamma_3(\eta) = z;\ \gamma_4(\eta) = u.$$

Применим частично рекурсивную биекцию $\mathscr{P}$, ставящую в соответствие четвёрке (α, q, Δ, i) натуральное число η.

$$\begin{cases} \eta(0) = \mathscr{P}(\alpha_0, q_0, \Delta(0), 1); \\ \eta(t + 1) = \mathscr{F}(\eta(t)).; \end{cases}$$

$$\mathscr{F}(\eta(t)) =$$

$= \mathscr{P}(F(\gamma_1(\eta(t-1))), G(\gamma_2(\eta(t-1))), D(\gamma_3(\eta(t-1))), \mathscr{I}(\gamma_4(\eta(t-1))))$.

Поскольку функции F, G, D задаются конечными таблицами, легко убедиться,что они примитивно рекурсивны как и функция $\mathscr{I}$, определяемая сложением. Поэтому $\mathscr{P}$ и $\mathscr{F}$ – рекурсивные функции, и шаги МТ эквивалентны выполнению операции Πp примитивной рекурсии.

Остановка МТ происходит тогда, когда она переходит в заключительное состояние, которое обозначим q_{stop}. Поэтому шаг t_{stop}, на котором МТ должна остановиться может быть описан оператором минимизации:

$$t_{stop} = \mu_t(q(t) = q_{stop}),$$

$$t_{stop} = \mu_t(\gamma_2(\eta(t)) = q_{stop}).$$

Таким образом, доказано, что процесс переработки информации машиной Тьюринга соответствует последовательности применения к наборам натуральных чисел операций суперпозиции, примитивной рекурсии и минимизации, т.е. процессу вычисления частично рекурсивной функции.

5.7 Алгоритмическая модель А. А. Маркова

Нормальные алгоритмы Маркова (так называют рассматриваемые далее модели алгоритмов) уточняют интуитивное понятие алгоритма на основе алфавитно-словарного подхода. Они имеют большое теоретическое и прикладное значение и лежат в основе многих систем и языков программирования, предназначенных для обработки символьной информации.

Пусть A – конечный алфавит, $\rightarrow$ и $\bullet$ – вспомогательные дополнительные символы. Обозначим $S(A)$ – множество всевозможных слов над A. Число символов, входящих в слово, как обычно, будем называть его длиной. Пустым называется слово, не содержащее символов.

Будем обозначать слова так:

$$W = a_1 a_2 \ldots a_r,$$

где W – символ, определяющий данное слово, $a_1a_2\dots a_r$ – идущие друг за другом символы, составляющие это слово.

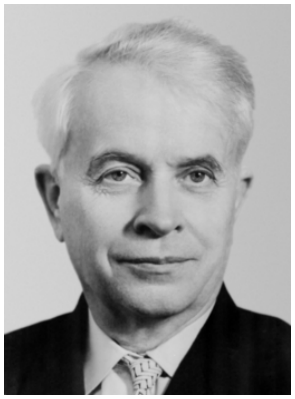


Рис. 5.29: А. А. Марков (младший), 1903–1979 гг., русский математик, основоположник советской школы конструктивной математики

Если $W = a_1a_2\dots a_r$, $B = b_1b_2\dots b_k$, $C = c_1c_2\dots c_m$, а

$$V = a_1a_2\dots a_rb_1b_2\dots b_kc_1c_2\dots c_m,$$

то говорят, что слово W является *префиксом*, а слово C – *окончанием* слова V. В таком случае используется запись

$$V = WBC$$

и говорят, что слова W, B и C *входят* в слово V.

Префикс и суффикс могут быть пустыми словами.

Если в слове V заменить любое входящее в него слово, например B, на какое-нибудь другое слово, например, на слово $D = d_1d_2\dots d_s$, то будет получено новое слово

$$G = WDC = a_1a_2\dots a_rd_1d_2d_s\dots b_kc_1c_2\dots c_m.$$

Такой переход от слова к другому слову путём замены является единственным оператором рассматриваемой алгоритмической модели, называемым *марковской подстановкой* и обозначаемым

$$B \to D.$$

Подстановка понимается следующим образом:

1) если обрабатываемое слово содержит в себе слово B, то оно заменяется в обрабатываемом слове на слово D (подстановка применима);

2) если в обрабатываемом слове слово B не содержится, то обрабатываемое слово не изменяется (подстановка неприменима).

Алгоритм Маркова заключается в последовательном применении подстановок, причём подстановки, имеющие специальное обозначение $\to \bullet$, называются *заключительными*, указывают на остановку алгоритма после их выполнения и имеют вид

$$B \to \bullet D,$$

где B и D – некоторые слова, называемые соответственно левая и правая часть подстановки.

Чтобы записывать все подстановки единым образом, вводится формализм

$$B \to \gamma D,$$

где γ может быть пробелом или точкой: $\gamma \in \{\bullet, \Lambda\}$; Λ – обозначение пробела.

Конечная непустая упорядоченная система подстановок

$$\begin{cases} L_1 \to \gamma_1 R_1; \\ L_2 \to \gamma_2 R_2; \\ \ldots\ldots\ldots\ldots \\ L_k \to \gamma_k R_k; \end{cases}$$

называется *нормальной схемой* в алфавите A.

Описание процесса выполнения марковских алгоритмов.

Пусть P – текущее обрабатываемое алгоритмом слово.

1^0 Начиная с первой подстановки упорядоченной схемы подстановок, отыскивается подстановка с левой частью, входящей в слово P. Если таких подстановок нет – алгоритм заканчивает свою работу, и полученным результатом считается само слово P, иначе выполняется пункт 2^0.

2^0 Если подстановки, левая часть которых входит в слово P, имеются, то берется самая "верхняя" подстановка в схеме, обладающая этим свойством. Выбранная подстановка применяется к слову P, начиная с проверки возможности её выполнения с первого символа слова P, и если она не является заключительной, до тех пор, пока её применение возможно. В результате получается некоторое новое слово P_1, которое становится текущим обрабатываемым словом: $P := P_1$.

После первого же выполнения заключительной подстановки алгоритм сразу заканчивает работу.

В противном случае осуществляется переход к следующей по порядку подстановке схемы.

3^0 Если рассмотрена последняя подстановка схемы, перейти на шаг 1^0. $\square$

Если процесс подстановок некоторой схемы никогда не заканчивается, то говорят, что определяемый этой схемой нормальный алгоритм *неприменим* к заданному слову P.

Пример. Нормальный алгоритм над алфавитом $\{|, +\}$, предназначенный для обработки слов вида $||\ldots|+||\ldots|$, задан схемой

$$\begin{cases} |+| \to +|| \\ +| \to \bullet| \end{cases}$$

и определяет сложение двух чисел в унарной системе счисления.

Рассмотрим применение этого алгоритма к слову $P = ||+|||$.

Применение первой подстановки приводит к получению слова $P = |+||||$. Первая подстановка снова применима, что приводит к слову $P = +|||||$. Теперь применима только вторая, заключительная подстановка, и в результате получается слово $P = |||||$. $\square$

Очевидно, что нормальный алгоритм, как и машина Тьюринга, определяет некоторую словарную функцию $f : S(A) \to S(A)$. Заменяя каждую букву конечного $k-$элементного алфавита A соответствующей по порядку $k-$значной цифрой, можно получить схему вычисления некоторой функции $f : \mathbb{N}_0 \to \mathbb{N}_0$.

Определение 5.6. *Функция называется вычислимой по Маркову, если существует нормальный (марковский) алгоритм, позволяющий по любому заданному допустимому значению аргумента найти значение этой функции.*

Далее мы докажем, что функция вычислима по Маркову тогда и только тогда, когда она вычислима по Тьюрингу. Следствием этого факта будет то, что функция вычислима по Маркову тогда и только тогда, когда она является частично рекурсивной, и более того:

> *можно предположить, что формальный вывод (доказательство) в исчислении существует тогда и только тогда, когда существует нормальный алгоритм, определяющий по слову – совокупности аксиом – другое слово – доказуемую теорему.*

Теорема 5.6. *Если функция вычислима по Тьюрингу, то она вычислима по Маркову.*

Доказательство. Пусть функция вычислима по Тьюрингу.

Применим следующий формализм. Определим объединённый алфавит $\mathscr{A} = A \cup Q$ из символов, записываемых на ленту МТ и символов её состояний. Текущее состояние МТ будем представлять словом в алфавите $\mathscr{A}$, причём символ текущего состояния будем помещать точно перед обозреваемым символом на ленте:

$$a_1 ... a_{i-2} a_{i-1} q_t a_i a_{i+1} ... a_r. \tag{43}$$

На произвольном шаге t МТ выполняет одну из следующих команд:

$$q_t a_i q_* a_* L \quad (L);$$

$$q_t a_i q_* a_* S \quad (S);$$

$$q_t a_i q_* a_* R \quad (R).$$

Команда L приводит к замене символа a_i на символ a_*, изменению состояния q_t на состояние q_* и сдвигу влево на одну ячейку ленты (на один индекс символа обрабатываемого слова). Следствием выполнения этой команды будет преобразование слова (43) в слово (44):

$$a_1 ... a_{i-2} q_* a_{i-1} a_* a_{i+1} ... a_r. \tag{44}$$

Очевидно, что выполнение команды приводит лишь *к локальному* изменению слова (43), к замене в нём трехбуквенного подслова на другое слово, что точно описывается подстановкой

$$a_{i-1} q_t a_i \to q_* a_{i-1} a_*.$$

Команды S и R эквивалентны подстановкам

$$q_t a_i \to q_* a_*;$$

$$q_t a_i \to a_* q_*.$$

Таким образом, последовательность выполняемых команд МТ можно заменить эквивалентной последовательностью подстановок, что доказывает теорему.

Теорема 5.7. *Если функция вычислима по Маркову, то она вычислима по Тьюрингу.*

Доказательство. Пусть произвольная функция f вычислима по Маркову. Функция f осуществляет алфавитное отображение некоторого исходного слова в результирующее слово путём последовательного выполнения конечного числа подстановок – замен некоторых подслов на другие слова. Но все такие действия осуществимы, что легко проверить, при помощи суперпозиции следующих машин Тьюринга:

$M1$ – обеспечивает определение номера позиции вхождения заданного слова в другое слово (и возвращает ноль в случае невхождения);

$M2$ – вычисляет длину заданного слова;

$M3$ – вставляет необходимое число пробелов в заданное слово с нужной позиции;

$M4$ – удаляет необходимое число пробелов в заданном слове с нужной позиции;

$M5$ – удаляет подслово в другом слове с заданной позиции.

$M6$ – вставляет слово в другое слово с заданной позиции.

5.8 Алгоритмическая разрешимость

Доказанная эквивалентность классов функций, вычислимых по Тьюрингу, вычислимых по Маркову, частично-рекурсивных функций, – является убедительной аргументацией в пользу правильности выбора точного определения алгоритма как строго описанной последовательности обработки информации в любой из рассмотренных алгоритмических моделей. Более того, известны и другие аналогичные модели – например Э. Поста, М. Минского, но они оказались эквивалентными тьюринговской модели и всем остальным.

Таким образом, в современной информатике и математике алгоритм, как одно из базовых понятий, определен строго.

Алгоритм решения задачи существует тогда и только тогда, когда существует машина Тьюринга, правильно решающая эту задачу за конечное число шагов.

Определение 5.7. *Функция называется вычислимой, если она вычислима по Тьюрингу.*

Таким образом, понятие вычислимости базируется на строгой алгоритмической модели Тьюринга (на семействе эквивалентных математических моделей).

Возникает вопрос: существуют ли функции, не являющиеся вычислимыми (не реализуемые алгоритмически)?

Теорема 5.8. *Существуют функции, не являющиеся вычислимыми.*

Доказательство. Возьмём сравнительно узкое семейство функций одного аргумента

$$\mathscr{F} = \{f : \mathbb{N}_0 \to \{0, 1\}\}$$

и покажем, что даже в этом семействе $\mathscr{F}$ содержатся невычислимые функции.

Сначала покажем, что семейство $\mathscr{F}$ шире счётного множества. Действительно, если предположить, что число функций в семействе $\mathscr{F}$ является счётным: $\mathscr{F} = \{f_0, f_1, f_2, ..., f_k, ...\}$, то все функции можно перечислить следующей бесконечной таблицей:

Таблица 5: Таблица функций семейства $\mathscr{F}$ в предположении их счётности

$x \setminus f(x)$	f_0	f_1	f_2	...	f_k	...
0	y_{00}	y_{01}	y_{02}	...	$y_{0,k}$	...
1	y_{10}	y_{11}	y_{12}	...	y_{1k}	...
2	y_{20}	y_{21}	y_{22}	...	y_{2k}	...
...	...	...	...	...	...	...
k	y_{k0}	y_{k1}	y_{k2}	...	y_{kk}	...
...	...	...	...	...	...	...

Но можно указать функцию $\varphi : \mathbb{N}_0 \to \{0, 1\}$, которая отличается от всех функций счётного множества $\{f_0, f_1, f_2, ..., f_k, ...\}$.

Действительно, функция

$$\varphi(x) = 1 - y_{xx}, \quad x = 0, 1, 2, .., \ ,$$

где y_{xx} – диагональные значения в таблице 5, будет отличаться от каждой функции множества $\{f_0, f_1, f_2, ..., f_k, ...\}$: от f_0 – при значении аргумента $x = 0$, от f_1 – при значении аргумента $x = 1$, ..., от

f_k – при значении аргумента $x = k$ и так далее. Поэтому семейство функций $\mathscr{F}$ шире счётного множества.

Ранее было показано, что число попарно неизоморфных машин Тьюринга не более чем счётно. Поэтому множество вычислимых функций не шире счётного. Но класс $\mathscr{F}$, как показано, шире счётного множества. Следовательно, в классе $\mathscr{F}$ не все функции вычислимы. $\square$

Выше были построены шифры для машин Тьюринга, при помощи которых удалось перечислить все попарно неизоморфные МТ. Шифр МТ, согласно этому построению, – двоичное натуральное число, начинающееся с единицы.

Определение 5.8. *Пусть $\mathbb{C}(M)$ – шифр машины Тьюринга M. Если в качестве начального слова на ленте МТ M взять слово $x = \mathbb{C}(M)$ – её собственный шифр, то возможны два исхода:*

1) начав работу над собственным шифром $x = \mathbb{C}(M)$, МТ M выполнит конечное число шагов и остановится; в этом случае говорят, что МТ M самоприменима;

2) начав работу над собственным шифром $x = \mathbb{C}(M)$, МТ M будет работать бесконечно долго – никогда не остановится; в этом случае говорят, что МТ M несамоприменима.

Проблема самоприменимости состоит в следующем. Для произвольной МТ M определить: является эта МТ M самоприменимой или нет?

Можно определить предикат $S_{app} = S_{app}(M)$, принимающий значение 1, если МТ M самоприменима, и значение 0, если МТ M несамоприменима. И поскольку МТ M полностью определяется её собственным шифром, этот предикат можно записать так: $S_{app}(\mathbb{C}(M))$.

Теорема 5.9. *Проблема самоприменимости алгоритмически неразрешима (предикат $S_{app}(M)$ невычислим).*

Доказательство. Пусть, от противного, проблема самоприменимости алгоритмически разрешима. Тогда для любой МТ M найдется машина Тьюринга M_0, которая по входному слову $x = \mathbb{C}(M)$ выдаст ответ 1 и остановится в заключительном состоянии q^0_{end}, если машина M самоприменима, и выдаст ответ 0 и остановится в

заключительном состоянии q^0_{end}, если машина M несамоприменима:

$$M_0(\mathbb{C}(M)) = \begin{cases} 1, & \textit{если } M \textit{ самоприменима;} \\ 0, & \textit{если } M \textit{ несамоприменима.} \end{cases}$$

Определим новую МТ M_1 следующим образом.

Алфавиты, начальное состояние и все функции (таблицы), определяющие работу МТ M_0, будут заимствованы для построения МТ M_1. Но состояние q^0_{end}, которое было конечным (для МТ M_0), для новой машины M_1 конечным не будет, поскольку к заимствованным командам МТ M_0 будут добавлены две новые команды и одно новое состояние q^1_{end} – конечное состояние этой новой МТ M_1. Добавляемые две команды К1 и К2 имеют следующий вид:

$$\begin{cases} q^0_{end}\ 1\ q^0_{end}\ 1 & \text{(К1);} \\ q^0_{end}\ 0\ q^1_{end}\ 0 & \text{(К2).} \end{cases}$$

Добавлением к описанию МТ M_0 этих двух команд и одного нового заключительного состояния q^1_{end} построение МТ M_1 заканчивается.

Согласно команде К1, в случае, когда МТ M_0 даёт ответ "1", означающий *самоприменимость* машины Тьюринга M, и останавливается, МТ M_1 *входит в бесконечный цикл и никогда не останавливается*, переходя из состояния q^0_{end} в то же самое состояние q^0_{end} без изменения информации на ленте. Поэтому имеет место следующее

Свойство 1 машины Тьюринга M_1: машина M_1 *неприменима к шифрам самоприменимых машин.*

Согласно команде К2, в случае, когда МТ M_0 дает ответ "0", означающий *несамоприменимость* машины Тьюринга M, и останавливается, МТ M_1 переходит в свое заключительное состояние q^1_{end} и останавливается. Поэтому имеет место

Свойство 2 машины Тьюринга M_1: машина M_1 *применима к шифрам несамоприменимых машин.*

Свойство 3 машины Тьюринга M_1. Машина M_1 существует тогда и только тогда, когда существует машина M_0.

Это свойство следует из того, что если МТ M_0 существует, то всегда можно добавить к ней две точно определённые команды К1 и

К2, получая МТ M_1. И обратно: если МТ M_1 существует, то всегда можно изъять из ее определения две точно определённые команды К1 и К2, получая МТ M_0.

Сама же построенная МТ M_1 может быть либо а) самоприменимой, либо б) несамоприменимой. Эти два случая являются взаимоисключающими, и каждый из них приводит к противоречию.

Действительно, в случае а) МТ M_1 самоприменима, следовательно, *применима* к своему собственному шифру – *к шифру самоприменимой машины.* Но по *свойству 1* машина M_1 *неприменима к шифрам самоприменимых машин.*

В случае б) МТ M_1 несамоприменима, следовательно, *неприменима* к своему собственному шифру – *к шифру несамоприменимой машины.* Но по *свойству 2* машина M_1 *применима к шифрам несамоприменимых машин.*

Указанные противоречия исключают существование МТ M_0 и возможность алгоритмической разрешимости проблемы самоприменимости. □

Определение 5.9. *Проблема Z алгоритмически сводится к проблеме W, если для любой индивидуальной задачи $z \in Z$ существует алгоритм (машина Тьюринга) M, который преобразует эту задачу z в некоторую другую индивидуальную задачу w, принадлежащую проблеме W: $M(z) = w \in W$, причём результат решения задачи w всегда в точности совпадает с результатом решения задачи z.*

Это определение не будет вызывать недоумения, если вспомнить: как решаются задачи в неалгоритмической математике. Например, получив для решения уравнение (входное слово)

$$\sin x + \cos x - \sin x \cos x - 1 = 0,$$

можно разложить его левую часть на множители

$$(\sin x - 1)(1 - \cos x) = 0,$$

сведя его к другому уравнению (*выходному слову*), решения которого в точности совпадают с решениями исходного уравнения.

Остаётся только представить машину Тьюринга, которая преобразует запись на ленте $\sin x + \cos x - \sin x \cos x - 1 = 0$ в запись

$(\sin x - 1)(1 - \cos x) = 0$.

Подчеркнём, что приведенный пример преобразования тригонометрического уравнения предназначен только для того, чтобы показать, что принцип сведения оной задачи к другой – традиционный приём в математике; и этот приём не предполагает здесь попытки дальнейшего алгоритмического решения.

Алгоритмическую сводимость проблемы Z к проблеме W будем обозначать $Z \ltimes W$.

Свойства алгоритмической сводимости.

1^0 $Z \ltimes Z$ (любая задача тривиально сводится к самой себе);

2^0 $(Z1 \ltimes Z2) \wedge (Z2 \ltimes Z3) \Longrightarrow (Z1 \ltimes Z3)$ (транзитивность следует из существовании композиции для любых двух машин Тьюринга).

3^0 Если $Z1 \ltimes Z2$, и $Z1$ – алгоритмически неразрешимая проблема, то и $Z2$ является неразрешимой проблемой.

Действительно, предположив, что $Z2$ – алгоритмически разрешима, любую индивидуальную задачу $z \in Z1$ сначала можно в силу условия $Z1 \ltimes Z2$ преобразовать алгоритмом сведения в некоторую задачу $w \in Z2$, а затем решить эту задачу w некоторым алгоритмом, поскольку по сделанному предположению $Z2$ – алгоритмически разрешима. Тогда любую индивидуальную задачу $z \in Z1$ можно решить при помощи алгоритма, являющегося композицией указанных двух алгоритмов, что противоречит неразрешимости проблемы $Z1$. $\square$

Проблема останова (***применимости***) состоит в следующем. По данной машине Тьюринга M и любому заданному входному слову P установить, применима ли машина к этому слову P, или нет?

Теорема 5.10. *Проблема останова алгоритмически неразрешима.*

Доказательство. Очевидно, легко алгоритмически свести неразрешимую проблему самоприменимости к проблеме останова, взяв в качестве входного слова P шифр заданной машины M, откуда по свойству 3^0 алгоритмической сводимости следует неразрешимость проблемы останова.

Проблема выводимости в заданной системе марковских подстановок состоит в следующем. Даны система марков-

ских подстановок П и начальное слово $\mathscr{B}$. Можно ли для произвольного слова P установить: выводимо это слово P из начального слова $\mathscr{B}$ при помощи марковской системы подстановок П или нет?

Теорема 5.11. *Проблема выводимости в заданной системе марковских подстановок алгоритмически неразрешима.*

Доказательство можно провести путём сведения неразрешимой проблемы самоприменимости к проблеме выводимости в заданной системе марковских подстановок. Для этой цели следует взять произвольную машину Тьюринга M и ее шифр $\mathbb{C}(M)$ в качестве входного (начального) слова $\mathscr{B} = \mathbb{C}(M)$. В качестве системы подстановок П следует взять систему, алгоритмически эквивалентную машине M (такая система подстановок существует и может быть алгоритмически построена по заданной МТ M в силу теоремы 5.6). Если q_{end} – заключительное состояние МТ M, и слово $P = q_{end}1$ выражает остановку машины M с выдачей ответа "1", то выводимость этого слова по входному слову $\mathscr{B} = \mathbb{C}(M)$ и указанной системе подстановок П определяет неразрешимую проблему самоприменимости. □

5.9 Неразрешимые формальные теории. Теоремы Чёрча

Для сколь нибудь сложной математической
теории разрешимость является скорее
исключением из общего правила
Р. Линдон [7, с.57]

Идея доказательств теорем о неразрешимости исчисления предикатов первого порядка и формальной арифметики, представленных ниже в этом параграфе, основана на использовании предиката $T(i, a, t)$ ="машина Тьюринга с шифром i, будучи применена к входному слову a, остановится на шаге t" и на сведении проблемы самоприменимости к проблеме выводимости формул этих двух формальных теорий.

Обозначим M_i машину Тьюринга, имеющую шифр i, φ_i – реализуемое МТ M_i отображение и $\varphi_i(a)$ – *значение, которое вычисляет* МТ M_i, *будучи примененной к слову* a.

Определение 5.10. *Формальная теория называется разрешимой, если существует алгоритм, позволяющий для любой формулы этой теории за конечное число шагов определить, является она теоремой или нет. Если же существует формула теории, для которой такого алгоритма нет, то эта формула и сама теория называются неразрешимыми.*

Теорема 5.12. *Теория L исчисления высказываний разрешима.*

Доказательство основано на следующем свойстве исчисления высказываний: все выводимые (доказуемые) формулы этой теории являются тождественно истинными. Действительно, легко проверить, что все аксиомы теории L – истинны, и что применение единственного правила вывода теории L *modus ponens* $\frac{A, A \supset B}{B}$ к истинной формуле A приводит к истинной формуле B.

Если дана произвольная формула $\mathscr{L}$ теории L, то она конечна и является либо константой, либо зависит от некоторого числа n переменных $(x_1, ..., x_n)$. Тогда методом перебора не более чем 2^n наборов значений этих переменных можно вычислять значения $\mathscr{L}(x_1, ..., x_n)$ в следующем порядке:

$$
\begin{array}{c}
False, False, ..., False, False, False \\
False, False, ..., False, False, \ True \\
False, False, ..., False, \ True, False \\
False, False, ..., False, \ True, \ True \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
True, \ True, \ldots, True, \ True, True
\end{array}
$$

Если на всех 2^n наборах значений переменных $\mathscr{L}(x_1, ..., x_n) = True$, то $\mathscr{L}$ тождественно истинна и поэтому является теоремой в L. Иначе, как только $\mathscr{L}(x_1, ..., x_n) = False$, вычисления прекращаются, и в этом случае формула $\mathscr{L}$ теоремой не является.

Указанная процедура вычисления значений формулы $\mathscr{L}$ и является разрешающим алгоритмом.

Теорема 5.13.

a) Предикат $T(i, a, t)$ разрешим.

b) $\varphi_i(a)$, как частичная функция аргументов i и a, вычислима.

Доказательство. a)Опишем разрешающий алгоритм, позволяющий для любой заданной тройки натуральных чисел (i, a, t) найти значение предиката $T(i, a, t)$. Сначала определим, описыва-

ет ли число i некоторую машину Тьюринга. Если нет, то значением $T(i,a,t)$ будет ложь. Иначе, выполняя вычисления на этой МТ с заданным входным словом a до шага t определим, именно ли на шаге t МТ останавливается. Если это так, то значение $T(i,a,t)$ будет истинным, иначе – ложным.

b) Напомним, что $\varphi_i(a)$ определена как *значение, которое вычисляет* МТ M_i, *будучи примененной к слову* a. При этом $\varphi_i(a)$ является частичной функцией, поскольку она определена только для тех пар значений (i,a), для которых существует шаг t такой, на котором предикат $T(i,a,t)$ становится истинным. Иначе говоря, функция $\varphi_i(a)$ для каждой фиксированной МТ M_i эквивалентна таблице, ставящей в соответствие аргументу a значение функции $\varphi_i(a)$. Для таких a, на которых $\varphi_i(a)$ определена, ее значение находится следующим алгоритмом. По шифру i восстанавливается программа МТ M_i; работа МТ M_i осуществляется с шага 0 над начальным словом a до шага t, на котором предикат $T(i,a,t)$ становится истинным, и с ленты считывается найденное значение $\varphi_i(a)$. Иначе, если для данного a указанная МТ никогда не останавливается, значение $\varphi_i(a)$ не определено, что не противоречит её частичности. $\square$

Из разрешимости предиката $T(i,a,t)$ следует, что представляющая его характеристическая функция

$$F_T(i,a,t) = \begin{cases} 1, & \text{если } T(i,a,t) = True, \\ 0, & \text{если } T(i,a,t) = False, \end{cases}$$

также является вычислимой.

Теорема 5.14. *Функция*

$$\psi(a) \stackrel{\text{def}}{=} \begin{cases} \varphi_a(a) + 1, & \textit{если } \exists T(a,a,t) = True, \\ 0, & \textit{в противном случае,} \end{cases}$$

невычислима.

Доказательство. Предположим, что функция ψ вычислима, т.е. существует вычисляющая её машина Тьюринга M_ψ, имеющая шифр p. Подчеркнём: вычислимая по предположению функция ψ отождествляется с машиной M_ψ, имеющей шифр p. Положим $a = p$. Тогда результатом вычисления ψ от аргумента p будет $\psi(p) = \varphi_p(p) + 1$ или $\psi(p) = 0$.

С другой стороны, если вычислено значение $\psi(p) = \varphi_p(p) + 1$, то МТ с шифром p самоприменима, и тогда существует шаг t, для которого $T(p, p, t) = True$; а если $\psi(p) = 0$, то МТ с шифром p несамоприменима. Следовательно, вычислимость функции $\psi(p)$ влечёт алгоритмическую разрешимость проблемы самоприменимости, о которой известно, что она неразрешима.

Полученное противоречие завершает доказательство теоремы.

Теорема 5.15 (А. Чёрч)**.** *Исчисление предикатов первого порядка неразрешимо.*

Доказательство. Достаточно указать хотя бы одну неразрешимую формулу исчисления предикатов первого порядка. Таковой будет формула $\exists t T(i, i, t)$, недоказуемая вследствие того, что она выражает неразрешимую проблему самоприменимости.

Действительно, если формула $\exists t\ T(i, i, t)$, которую мы обозначим $\mathscr{S}$, разреши́ма, то вычислима представляющая её характеристическая функция

$$C_{\mathscr{S}}(i, i, t) = \begin{cases} 1, & \text{если } \exists t T(i, i, t) = True; \\ 0, & \text{если } \exists t T(i, i, t) = False, \end{cases}$$

и существует вычисляющая функцию $C_{\mathscr{S}}$ машина Тьюринга M. Если результат этого вычисления есть $C_{\mathscr{S}}(i, i, t) = 1$, то $\exists t T(i, i, t) = True$, и МТ M_i с шифром i является самоприменимой. Тогда, применив МТ M_i к слову i, получим $\varphi_i(i)$, и затем, добавив единицу, определим функцию $\psi(i) = \varphi_i(i) + 1$.

Если же $C_{\mathscr{S}}(i, i, t) = 0$, и тогда $\exists t T(i, i, t) = False$, то МТ M_i несамоприменима. В этом случае положим $\psi(i) = 0$.

Таким образом, функция $\psi(i)$ оказывается вычислимой. Но это противоречит установленной теоремой 5.14 невычислимости функции $\psi(i)$. $\square$

Будем полагать, что формальная теория Ar адекватна содержательной арифметике натуральных чисел, что предполагает выполнение следующих условий:

A1) для каждого предложения арифметики натуральных чисел в теории Ar имеется формула, выражающая это предложение;

A2) если предложение арифметики натуральных чисел истинно, то соответствующая ему формула теории Ar доказуема в Ar;

А3) в теории Ar доказуемы только те формулы, которые выражают истинные предложения арифметики натуральных чисел.

В силу возможности арифметизации машин Тьюринга, предложение $\exists t T(i,i,t)$, выражающее проблему самоприменимости, можно сформулировать в терминах содержательной арифметики натуральных чисел. Поэтому в теории Ar в соответствии с условием А1 имеется формула, выражающая это предложение.

Обозначим $\mathscr{F}_i$ формулу, выражающую в теории Ar предложение $\exists t T(i,i,t)$. Формула имеет вид

$$\exists t(C_{\mathscr{S}}(\bar{i},\bar{i},t)=1),$$

где $C_{\mathscr{S}}(i,i,t)$ – характеристическая (арифметическая) функция предиката самоприменимости машины Тьюринга с шифром i. Согласно условию А2, если $\exists t T(i,i,t) = True$, то в теории Ar доказуема (выводима) формула $\mathscr{F}_i$. И обратно: согласно условию А3, если формула $\mathscr{F}_i$ доказуема, то $\exists t T(i,i,t) = True$.

Следовательно, формула $\mathscr{F}_i$ доказуема в теории Ar тогда и только тогда, когда предикат $\exists t T(i,i,t)$ истинен.

Теорема 5.16 (А. Чёрч)**.** *Теория Ar (формальная арифметика) неразрешима.*

Доказательство. Предположим, от противного, что в теории Ar существует разрешающий алгоритм, определяющий доказуемость формул в Ar. Применим этот разрешающий алгоритм к формуле $\mathscr{F}_i$, выражающей проблему самоприменимости. Если окажется, что формула $\mathscr{F}_i$ доказуема, то предикат $\exists t T(i,i,t)$ будет истинен, а если формула $\mathscr{F}_i$ окажется недоказуемой, то предикат $\mathscr{F}_i$ будет ложен. Таким образом, получен разрешающий алгоритм для предиката $\exists t T(i,i,t)$, что противоречит неразрешимости этого предиката, установленной при доказательстве теоремы 5.15.

Теорема 5.17 (Гёделя о неполноте)**.** *Если теория Ar (формальная арифметика) непротиворечива, то она неполна: в ней существует неразрешимое предложение.*

Доказательство. Рассмотрим формулу $\neg(\exists t(C_{\mathscr{S}}(\bar{i},\bar{i},t)=1))$, где $C_{\mathscr{S}}(i,i,t)$ – характеристическая (арифметическая) функция предиката самоприменимости машины Тьюринга с шифром i, и обозна-

чим эту формулу $\neg\mathscr{F}_i$. Предположим, что предложение $\neg\mathscr{F}_i$ разрешимо в Ar. Тогда, в силу непротиворечивости теории Ar, возможны два взаимоисключающих случая. Либо предложение $\neg\mathscr{F}_i$ выводимо в Ar , и тогда МТ с шифром i несамоприменима, либо $\mathscr{F}_i$ выводимо в Ar, и тогда МТ с шифром i самоприменима. Следовательно, предположение о разрешимости предложения $\neg\mathscr{F}_i$ приводит к противоречию: разрешимости проблемы самоприменимости. Поэтому либо предложение $\neg\mathscr{F}_i$ неразрешимо, либо система Ar противоречива.

5.10 Алгоритмическая модель RAM с равнодоступными ячейками памяти

Изучение алгоритмического метода решения задач позволило выяснить, что он применим далеко не всегда. Действительно, существуют алгоритмически неразрешимые проблемы – такие, что алгоритм для решения произвольной индивидуальной задачи, входящей в эту проблему, вообще может не существовать. Поэтому изучение компьютерной (алгоритмической) математики предполагает, что будут рассматриваться только алгоритмически разрешимые проблемы.

Тем не менее, среди алгоритмически разрешимых проблем существуют как простые, так и сложные, когда для решения задач необходимо выполнить огромное число шагов – какая бы при этим ни использовалась алгоритмическая модель или компьютер. Поэтому *важно уметь оценивать временную вычислительную сложность проблем – максимальное число шагов, которое потребуется для решения задач, входящих в проблему.*

Элементарные вычислительные шаги в различных моделях существенно различаются, хотя, как показано выше, такие модели математически являются эквивалентными. *Основная идея состоит в том, чтобы оценивать временную сложность алгоритмов независимо от вида используемой модели – вплоть до современного компьютера.*

Для реализации этой идеи была предложена вычислительная модель, называемая RAM-машиной или просто RAM (*RAM* –

Random Access Memory, – машина с произвольным или равнодоступным обращением к памяти). Под этой равнодоступностью понимается то, что в отличие от модели Тьюринга, ячейки памяти RAM-модели имеют адреса, и к каждой ячейке можно обращаться по собственному адресу ячейки на произвольных шагах вычислений и в произвольном порядке. Используемые в RAM принцип адресного обращения к памяти и программное управление позволяют существенно приблизить эту модель к модели Дж. фон Неймана, на основе которой построены современные компьютеры. Поэтому алгоритмические возможности компьютеров теоретически можно считать равными алгоритмическим возможностям RAM, за исключением того, что RAM имеет неограниченную память, а память компьютеров всегда ограничена. Что же касается отсутствия ограничения на объём используемой памяти в RAM – это необходимо для сохранения эквивалентности RAM модели Тьюринга и распространения на неё тезиса Чёрча.

RAM-модель описывается набором следующих элементов.

Входная лента – аналог потока ввода (рис.5.30) – последовательность ячеек, с которой возможно *только считывание* информации. В каждой клетке может содержаться положительное или отрицательное целое число. Перед началом вычислений на входной ленте записана исходная информация для обработки. Полагается, что считывание осуществляется читающей головкой, которая в начале работы RAM обозревает крайнюю левую ячейку и сдвигается на одну ячейку вправо после каждого считывания.

Выходная лента – аналог потока вывода – позволяет последовательно записывать в ячейки целые числа. Вначале все ячейки входной ленты пусты. После записи в ячейку обязательно происходит сдвиг вправо на следующую ячейку. Записанную на выходную ленту информацию изменить нельзя.

Память RAM состоит из последовательности регистров с адресами $0, 1, 2, \ldots$, каждый из которых может хранить произвольное целое число. Число регистров памяти не ограничено, как и число ячеек ленты в модели Тьюринга.

Регистр с адресом 0 имеет специальное назначение: он является универсальным сумматором – элементом памяти, предназначенным

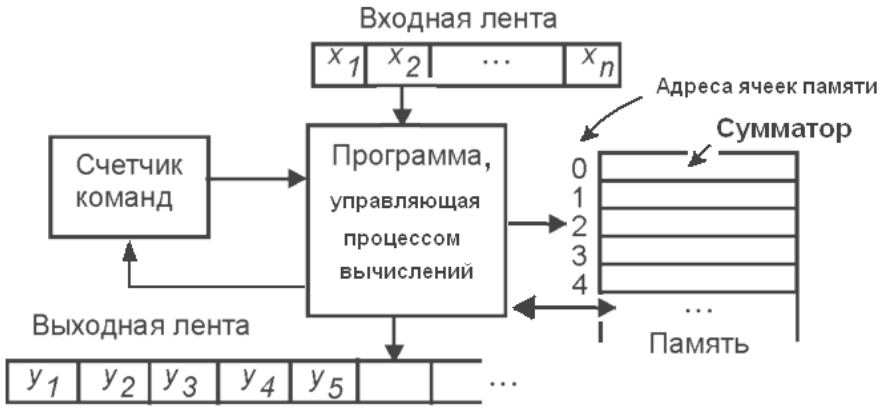


Рис. 5.30: Вычислительная модель RAM

для выполнения операций. Как в одноадресных компьютерах, выполнение бинарных операций происходит по схеме

$$\boxed{\text{<сумматор>} \circ \text{<операнд команды>} \Rightarrow \text{сумматор}},$$

где " $\circ$ " – знак операции (ADD,SUB,MULT,DIV – см. ниже), а угловые скобки обозначают содержимое заключенного в них объекта.

Перед началом вычислений все регистры полагаются очищенными.

Программа RAM не записывается в память (и поэтому не может себя изменять), а *является управляющей последовательностью*, подобно таблицам, управляющим работой машины Тьюринга. Программа состоит из последовательности команд. Команды нумеруются числами $1, 2, 3, \ldots$, которые называют номерами команд.

Один шаг в модели RAM – это выполнение одной команды.

Команда RAM имеет вид

$$\boxed{\text{КОП} \mid a} \ ,$$

где КОП обозначает код операции, a – операнд команды.

Счётчик команд указывает на текущую выполняемую команду (вначале – на первую). После выполнения команды, которая не является командой условного или безусловного перехода, счётчик

команд увеличивается на единицу. Иначе – устанавливается на номер команды, указанный в выполненной команде перехода.

Для их описания команд будем использовать следующие обозначения.

Ргn – регистр с номером n, где $n = 0, 1, 2, ...$.

$C(i)$ – содержимое регистра с номером i.

$C(C(i))$ – содержимое регистра, номер которого в свою очередь является содержимым регистра с номером i (для описания косвенной адресации).

$V(a)$ – значение операнда a в команде, которое обозначает число i, если a имеет вид "$= i$"; число $C(i)$, если a имеет вид "i"; число $C(C(i))$, если a имеет вид "$*i$".

$\alpha \leftarrow \beta$ – обозначение пересылки β в α, где β и α является содержимым произвольной ячейки, регистра или непосредственным операндом.

Пусть i – натуральное число. В модели RAM возможны три вида операндов:

Обозначение операнда в команде	*Что используется в качестве операнда*
$= i$	само число i (непосредственная адресация)
i	содержимое регистра с номером i (прямая адресация)
$*i$	содержимое регистра, номер которого является содержимым регистра с номером i (косвенная адресация)

Ниже приведена простейшая программа суммирования чисел. Предполагается, что на входной ленте информация представлена в следующей порядке: $n, a_1, a_2, ..., a_n$, где n – количество суммируемых элементов; $a_1, a_2, ..., a_n$ – числа, подлежащие суммированию.

```
      LOAD   0    // Очистка сумматора;
      READ   1    // Количество суммируемых чисел в Рг1;
      STORE  2    // Очистка Рг2 для накопления суммы;
4     READ   3    // Читать очередное число с ленты в Рг3;
      LOAD   2    // Взять частичную сумму из Рг2;
```

```
ADD     3    // Добавить к ней очередное число
STORE   2    // и вернуть в Рг2;
LOAD    1    // Сколько раз осталось суммировать – в Рг1 –
SUB    =1    // уменьшаем на единицу
STORE   1    // и сохраняем в Рг1;
JGTZ    4    //  Если не всё просуммировано – на команду 4,
WRITE   2    // иначе – вывод найденной суммы.
```

Список команд RAM.

№	*Команда*	*Описание действия команды*
1	LOAD a	$C(0) \leftarrow V(a)$ Запись значения в сумматор
2	STORE i STORE $*i$	$C(i) \leftarrow C(0)$ Запись содержимого сумматора в Рг i $C(C(i)) \leftarrow C(0)$ Запись содержимого сумматора в регистр, номер которого хранится в Рг i
3	ADD a	$C(0) \leftarrow C(0) + V(a)$ Добавить значение a к сумматору
4	SUB a	$C(0) \leftarrow C(0) - V(a)$ Вычесть значение a из сумматора
5	MULT a	$C(0) \leftarrow C(0) \times V(a)$ Умнож. значение сумматора на a
6	DIV a	$C(0) \leftarrow \lfloor C(0)/V(a) \rfloor$ Разделить значение сумматора на a и взять в сумматор в качестве результата целую часть от результата деления
7	READ i READ $*i$	Считать в регистр с номером i содержимое очередной ячейки входной ленты Считать в регистр с номером,содержащемся в регистре i, содержимое очередной ячейки входной ленты
8	WRITE i	Записать в текущую ячейку выходной ленты значение a и сдвинуть головку записи вправо на одну ячейку
9	JUMP n	Перейти к команде с номером n – установить на счётчике команд число n
10	JGTZ n	Если содержимое сумматора больше нуля, то перейти к команде с номером n; иначе – к следующей по порядку команде
11	JZERO n	Если содержимое сумматора равно нулю, то перейти к команде с номером n; иначе – к следующей по порядку команде
12	HALT n	Завершение работы RAM-машины

Ранее было установлено, что модели Тьюринга, Маркова, частично рекурсивных функций алгоритмически эквивалентны: если вычислительную проблему можно разрешить на одной из этих моделей, то можно разрешить и на другой. И понятие вычислимости является одним и тем же для этих моделей.

Оказывается также, что близкая по своему построению модель РАМ алгоритмически эквивалентна любой из перечисленных выше моделей.

Теорема 5.18. *Модели Тьюринга и РАМ алгоритмически эквивалентны.*

Доказательство теоремы будет конструктивным: нужно показать, что обрабатываемые данные и шаги МТ могут описаны в терминах модели РАМ и, обратно, команды РАМ могут быть описаны в терминах модели МТ. Поскольку полное, исчерпывающее доказательство по такой схеме является громоздким и неудобным для восприятия, будут приведены его основные элементы и опущены несущественные детали, которые при изучения курса могут быть использованы как упражнения.

I^o. Пусть проблема разрешима на МТ. Тогда решение любой индивидуальной задачи этой проблемы потребует выполнения не более некоторого числа T шагов. Вследствие этого МТ будет использовать при выполнении решения не более $2T$ ячеек памяти на ленте, поскольку, даже сдвигаясь только влево или только вправо, МТ не сможет сместиться более чем на T ячеек от начального положения.

Алфавит МТ (все используемые символы, включая символы состояний) может быть представлен множеством двоичных слов – как при кодировании в современном компьютере. Тогда эти символы в то же время являются целыми числами, а входное слово МТ длины n может быть размещено на входной ленте РАМ как последовательность этих чисел[14].

При помощи команды READ перепишем целые числа со входной ленты в регистровую память от РгT до Рг$(T + n - 1)$.

В Рг$(T + n)$ занесём начальное состояние МТ; обозначим $Q = T + n$; в процессе вычислений в этом же регистре с номером Q будет содержаться текущее состояние.

[14]Никакого ограничения на целые числа, размещаемые в регистрах РАМ, нет

Пусть Рг$(T+n+1)$ будет содержать указатель – текущий номер обозреваемого регистра. Обозначим этот зафиксированный номер $L = T + n + 1$.

Далее, начиная с Рг$(T+n+2)$ в линейном представлении будут записываться таблицы изменения состояний, выходов – записи на ленту – и сдвига. Эти таблицы МТ являются функциями двух аргументов, определённых на конечном множестве, например $q = h(i, j)$: после чтения символа i в состоянии j МТ переходит в состояние q. Тогда если $i = \overline{1, k_1}$, $j = \overline{1, k_2}$, а таблица $h(i, j)$ размещается, начиная с регистра с номером r, то при линейном размещении таблицы h элемент $h(i, j)$ будет содержаться в регистре с номером $r + (i - 1) \cdot k_2 + j - 1$.

Обозначим r – номер регистра, начиная с которого размещается таблица переходов МТ, v – номер регистра, начиная с которого размещается таблица выходов, а w – номер регистра, начиная с которого размещается таблица сдвигов МТ. Пусть текущий обозреваемый на ленте МТ символ соответствует содержимому регистра РгL.

Следующий фрагмент RAM программы обеспечивает выбор читаемого символа i и текущего состояния j.

```
LOAD   Q      // Q=T+n; чтение начального или текущего состояния;
STORE  NSt    // Текущее состояние j помещается в РгNSt;
LOAD   *L     // Чтение по адресу из РгL очередного символа;
              //  на первом шаге в РгL содержится значение T;
STORE  NSy    // считанный символ i помещается в РгNSy.
```

Получив i и j, РАМ модель позволяет легко реализовать следующие арифметические вычисления.

Вычисление номера регистра, в котором содержится символ нового состояния, по формуле $r + (i - 1) \cdot k_2 + j - 1$ и помещение его в РгNState.

Вычисление номера регистра для выбора нового записываемого символа по формуле $v+(i-1)\cdot k_2+j-1$ и помещение его в Рг$NSym$.

Вычисление номера регистра для выбора сдвига $\{-1;\ 0,\ +1\}$ по формуле $v + (i - 1) \cdot k_2 + j - 1$ и помещение его в РгNSh.

Затем новое состояние записывается в РгQ следующими командами:

```
    LOAD   *NState   // Новое состояние помещается в сумматор,
    STORE  Q         // пересылается в РгQ и становится текущим.
```

Сдвиг реализуется изменением на единицу адреса регистра, из которого читается информация. Если в РгL хранится адрес, который нужно, например, увеличить на единицу (реализуя сдвиг вправо) то содержимое РгL увеличивается на единицу.

```
    LOAD   L         // Номер обозреваемого регистра – в сумматор;
    ADD    *NSh      // Плюс 1,0 или -1 – реализация сдвига;
    STORE  L         // Номер следующего обозреваемого регистра;
```

Если номером заключительного состояния является значение J, то для моделирования остановки МТ может быть применен следующий фрагмент RAM-программы:

```
    STORE  J       // Номер заключительного состояния - в сумматор;
    SUB    Q       // Сравнение с текущим состоянием;
    JZERO  S       // Если совпадают – переход на команду HALT.
     ...
S   HALT
```

Все элементы команд МТ промоделированы фрагментами РАМ-программы.

II^o. Пусть для произвольной проблемы существует разрешающая её РАМ-программа. Достаточно показать, что любая команда РАМ может быть промоделирована некоторой машиной Тьюринга. Поскольку реализуемой является суперпозиция машин Тьюринга, то РАМ программа может быть промоделирована соответствующей композицией этих машин.

Начальную информацию на входной ленте РАМ можно считать эквивалентной начальной информации на ленте МТ, если выбрать подходящее кодирование. Взяв алфавит МТ $\{\Lambda, -, 1, 0, *, \diamond\}$, можно представлять через пробел положительные (без знака) и отрицательные (со знаком $-$) двоичные числа так, что каждому целому числу предшествует номер регистра в соответствии с размещением информации в регистрах РАМ. Например, запись на ленте МТ

$\Lambda 1 * 1101\Lambda 10*\text{-}101\Lambda 11*\text{-}11\Lambda$

соответствует последовательности значений в регистрах РАМ с номерами 1,2,3: $1 * 13, 2 * -5, 3 * -3$.

Символы "$*$"и "$\diamond$" предназначаются для использования в качестве разделителей для удобства программирования МТ. В частности, для отделения номера регистра от его содержимого и пометки свободных ячеек ленты, которые могут использоваться на текущем шаге как рабочие.

Выполнению операций с отрицательными числами предшествуют подпрограммы МТ, анализирующие знаки.

Вычитание рассматривается как сложение, если операнды имеют одинаковый знак, с сохранением знака минус, если операнды отрицательные. Реализацию вычитания несложно реализовать на МТ; это предлагается в качестве упражнения.

Умножение и деление операндов с одинаковыми знаками эквивалентно умножению и делению положительных операндов. Разные знаки при выполнении этих операций показывают, что к результату умножения или деления необходимо приписать минус.

Теперь вспомним: существование частично рекурсивной (рекурсивной) функции равносильно существованию МТ, вычисляющей эту же функцию. Сложение, умножение и целочисленное деление $\lfloor \frac{x}{y} \rfloor$, $y \neq 0$, реализуемы в классе частично рекурсивных функций. Поэтому команды ADD, SUB, MULT, DIV реализуемы на МТ.

При моделировании РАМ машинами Тьюринга используются суперпозиции МТ. Как было показано при изучении МТ, при их использовании возможно реализовать условный и безусловный переход (условную смену состояний). Поэтому команды РАМ JUMP, JGTZ, JZERO реализуемы на МТ. Команда HALT с подводом к ячейке n реализуется на МТ сдвигом и переходом в заключительное состояние. Команды LOAD, STORE, READ, WRITE реализуются чтением и записью в нужных местах на ленте МТ при помощи необходимого перемещения (сдвига) головки МТ. $\square$

6 Сложность алгоритмов и вычислительных проблем

6.1 Временная и пространственная сложность алгоритмических моделей

Принципиальным ограничением возможностей алгоритмического метода в математике является *алгоритмическая неразрешимость*, препятствующая получению доказательств и прочих решений для многих задач. Но если рассматривать исключительно класс алгоритмически разрешимых проблем, то возникают и другие ограничения, связанные с необходимостью решать задачи, укладываясь в заданное время и используя некоторый ограниченный объём памяти для хранения и обработки информации.

Действительно, в подавляющем числе случаев при использовании компьютеров для решения задач требуется, чтобы решение было получено за время, не превышающее заданного.

Например, для узнавания объекта – выдачи решения "свой" или "чужой" – в различных военных информационных системах может быть выделено строго ограниченное время от нескольких секунд до нескольких минут.

Прогноз для принятия решений при оперативном управлении операциями покупки и продажи ценных бумаг на бирже, выданный после часа расчётов, может оказаться неактуальным.

Время алгоритмического решения зависит как от особенностей решаемой задачи, так и от объёма начальной информации, представленной для её решения. Действительно, например, задача сложения двух матриц размерности $n \times n$ требует меньшего числа команд (шагов вычислительной модели) чем задача умножения этих же матриц. Но число шагов (команд сложения, умножения), выполняемых при решении этих задач, очевидно, растёт с ростом n. Кроме этого, имеет значение особенность исходных данных в каждом конкретном случае. Так, умножение на единичную матрицу тривиально и гораздо проще, чем умножение на матрицу, не содержащую нулей.

Из этих соображений возникает *понятие временной сложности*

вычислительной проблемы для заданной алгоритмической модели или компьютера как требуемое число шагов (команд) для решения самой сложной индивидуальной задачи этой проблемы.

Строгой формулировке определения временной сложности препятствует то, что шаги различных вычислительных моделей и компьютеров существенно различаются. Если сравнивать между собой только компьютеры, то при переходе от одного компьютера к другому временная сложность решения одной и той же проблемы будет изменяться в константу раз (эта константа определяется быстродействием).

Базируясь на тезисе Чёрча-Тьюринга, можно математически строго определить временную сложность проблемы как число шагов машины Тьюринга, которое потребуется для решения самой сложной индивидуальной задачи этой проблемы самой лучшей (по минимуму шагов) тьюринговской программой. Однако один шаг МТ существенно проще одной команды процессора компьютера, например, умножений или деления. Поэтому с теоретической точки зрения представляется важным сравнить между собой не только компьютерные, но и теоретические алгоритмические модели, в первую очередь – машину Тьюринга – с целью сопоставления числа требуемых шагов для выполнения одних и тех же операций. Именно для этого может быть использовано сравнение МТ с РАМ, которая может быть охарактеризована как модель командно-адресного компьютера.

Для сравнения числа шагов (или единиц времени), затрачиваемых на решение одной и той же задачи моделями РАМ и машины Тьюринга используются два подхода, которые часто называют критериями [1].

При использовании *равномерного* весового критерия полагается, что каждая команда РАМ выполняется за одну единице времени, а каждый регистр РАМ использует ровно одну единицу памяти.

При использовании *логарифмического* весового критерия учитывается особенность РАМ, состоящая в том, что размер числа (количество бит, отводимых на представление числа) в произвольном регистре может быть, оставаясь конечным, сколь угодно большим. Это является следствием того, что никакие ограничения на целые

числа, представимые в (абстрактной!) модели PAM не накладываются.

Логарифмический критерий оценивает произвольное целое число i при помощи следующей логарифмической функции (веса числа):

$$l(i) = \begin{cases} \lfloor log|i| \rfloor + 1, & \text{если } i \neq 0; \\ 1, & \text{если } i = 0. \end{cases}$$

Эта функция определяет число бит, достаточных для представления числа i в двоичном регистре.

Логарифмический весовой критерий предполагает, что цена (или вес) команды пропорционален длине её операндов.

Для непосредственного операнда вида $= i$ вес принимается равным $l(i)$.

Для операнда вида i, отсылающего к значению в регистре с номером i, вес принимает значение, равное

$$l(i) + l(c(i)),$$

где $l(i)$ – цена номера регистра, а $l(c(i))$ цена числа $c(i)$, являющегося содержимым регистра с номером i.

Для операнда вида $*i$, отсылающего к значению в регистре, номер которого содержится в регистре с номером i, вес принимает значение, равное

$$l(i) + l(c(i)) + l(c(c(i))),$$

где $l(i)$ – цена номера регистра-указателя, а $l(c(i))$ цена числа $c(i)$, являющегося номером регистра, содержащего числовое значение операнда, а $l(c(c(i)))$ – вес числа, которое будет использовано при выполнении команды.

Произвольный операнд (i, $= i$, или $*i$) будем обозначать символом a, а его вес – $t(a)$.

Принцип нахождения логарифмического веса (времени выполнения) команд PAM поясним на примере команды

$$ADD * i$$

При выполнении этой команды сначала происходит извлечение первого слагаемого из памяти. В соответствии с заданным операндом

сложность этого извлечения оценивается величиной

$$l(i) + l(c(i)) + l(c(c(i))).$$

Второе слагаемое является содержимым сумматора – регистра с номером 0 и обозначается $c(0)$. Его сложность определяется как $l(c(o))$. В итоге логарифмический вес или *время выполнения команды* $ADD * i$ полагается равным

$$l(c(o)) + l(i) + l(c(i)) + l(c(c(i))).$$

Временной сложностью программы p для алгоритмической модели $\mathscr{M}$ называется значение функции $f_{p,\mathscr{M}}(n)$ от длины входной информации n, равное наибольшей (по всем допустимым входным информациям длины n) из сумм времён, затраченных на каждую сработавшую команду.

Логарифмической ёмкостной сложностью PAM-программы называется сумма по всем работавшим регистрам (включая нулевой регистр – сумматор) величин $l(x_i)$, где x_i – наибольшее по модулю целое число, содержавшееся в регистре с номером i за всё время вычислений.

Ёмкостной сложностью машины Тьюринга $\mathcal{M}$ (программы, определённой машиной $\mathcal{M}$), называется длина рабочей зоны ленты, измеряемой числом ячеек, за пределы которой МТ $\mathcal{M}$ "не выходит" за все время выполнения программы.

Говорят, что временная сложность программы p для алгоритмической модели $\mathscr{M}_1$ полиномиально связана с временной сложностью той же программы для модели $\mathscr{M}_2$, если найдутся такие полиномы $\mathcal{P}_1(x)$ и $\mathcal{P}_2(x)$, что при любой допустимой длине входа n

$$f_{p,\mathscr{M}_1}(n) = \mathcal{P}_1(f_{p,\mathscr{M}_2}(n))$$

и

$$f_{p,\mathscr{M}_2}(n) = \mathcal{P}_2(f_{p,\mathscr{M}_1}(n)).$$

Следующие две теоремы служат обоснованием того, что имеет смысл сравнивать различные вычислительные модели, такие как PAM, машина Тьюринга или какой-либо компьютер, заранее считая, что они полиномиально связаны.

Теорема 6.1. *При логарифмическом весе команд для любой РАМ-программы существует эквивалентная машина Тьюринга, временная (емкостная) сложность которой полиномиально связана с временной (емкостной) сложностью РАМ-программы.*

Теорема 6.2. *При равномерном весе команд для любой РАМ-программы, не содержащей команд умножения, существует эквивалентная машина Тьюринга, временная (емкостная) сложность которой полиномиально связана с временной (емкостной) сложностью РАМ-программы.*

Конструктивные доказательства этих теорем, основанные на сравнении команд соответствующих моделей и анализе алгоритмической реализации команд умножения и целочисленного деления, опущены из-за их объёмности. Для разбора и самостоятельного получения этих доказательств можно использовать приёмы, представленные в книге Ахо, Хопкрофта и Ульмана [1].

6.2 Временная и пространственная сложность вычислительных проблем. Класс P

Напомним, что *вычислительной проблемой* называется совокупность однотипных задач, для решения которых используются алгоритмические методы. Задача $z \in Z$ называется *индивидуальной задачей* вычислительной проблемы Z.

Под задачей понимается предписание, согласно которому по данной начальной информации – входному слову x требуется получить правильный ответ – выходное слово y, используя какую-либо алгоритмическую модель. Входные слова x принадлежат заданному множеству *допустимых* слов X. Множество X характеризует проблему, а каждое слово $x \in X$ – индивидуальную задачу.

Длина входного слова (заданная числом символов) называется длиной входа и обозначается $n = len(x)$.

Любая заданная машина Тьюринга M реализует определённую программу (алгоритм) и обеспечивает решение задач для некоторого множества допустимых входных слов X, определяющего вычислительную проблему.

Временно́й сложностью машины Тьюринга M (алгоритма M) $TIME_M(n)$ называется наибольшее число шагов её работы $t(x)$, которое может потребоваться при решении индивидуальных задач, имеющих длину входа, равную n:

$$TIME_M(n) = \max_{\{x \in X : len(x)=n\}} t(x).$$

Пространственной сложностью машины Тьюринга M (алгоритма M) $SPACE_M(n)$ называется наибольшее число ячеек ленты $s(x)$ (наибольшая длина рабочей зоны), которое потребуется при решении индивидуальных задач, имеющих длину входа, равную n:

$$SPACE_M(n) = \max_{\{x \in X : len(x)=n\}} s(x).$$

Предполагая, что при описании алгоритмов и решении задач алгоритмическими методами используются исключительно полиномиально связанные модели и языки, в теории сложности алгоритмических вычислений используют оценки временной и пространственной сложности с точностью до полиномиальной эквивалентности используемых языков и моделей.

В этом смысле *разумным* считается использование машин и языков, отличающихся по времени выполнения команд и размерам ячеек памяти не более чем полиномиально – *класса полиномиально эквивалентных алгоритмических моделей.* Полагая, что используются модели исключительно из этого класса, в определении сложности алгоритма конкретный тип модели M (машины Тьюринга, PAM, компьютера или алгоритмического языка) можно опустить.

Определение 6.1. *Временно́й сложностью проблемы Z называется наименьшее число шагов решения самой "трудной" из задач вычислительной проблемы Z (при заданной длине входа n) самым "быстрым" решающим её алгоритмом:*

$$TIME_Z(n) = \min_{A \in \mathscr{A}} \max_{z(n) \in Z} t_A(z(n)),$$

где $\mathscr{A}$ – множество алгоритмов, которые могут быть применены для решения задач, входящих в проблему Z; A – произвольный

алгоритм из $\mathscr{A}$*;* $t_A(z(n))$ *– число шагов, которые выполнит алгоритм* A *в процессе решения индивидуальной задачи* $z(n)$*, имеющей вход длины* n*.*

Определение 6.2. *Пространственной сложностью проблемы* Z *называется наименьшее число ячеек памяти, требуемое для решения самой "трудной" из задач вычислительной проблемы* Z *(при заданной длине входа* n *) самым "экономным" решающим её алгоритмом:*

$$SPACE_Z(n) = \min_{A \in \mathscr{A}} \max_{z(n) \in Z} s_A(z(n)),$$

где $\mathscr{A}$ *– множество алгоритмов, которые могут быть применены для решения задач, входящих в проблему* Z*;* A *– произвольный алгоритм из* $\mathscr{A}$*;* $s_A(z(n))$ *– число ячеек, которые будут использованы в процессе решения индивидуальной задачи* $z(n)$*, имеющей вход длины* n*, алгоритмом* A*.*

В связи с тем, что измерение сложности происходит для произвольной модели из класса полиномиально эквивалентных, с теоретической точки зрения достаточно использовать верхние оценки вида

$$TIME_Z(n) \leq O(f(n)),$$

где $f(n)$ – вещественная функция натурального аргумента. Например, если $TIME_Z(n) \leq O(n)$, то говорят, что проблема Z имеет линейную временную сложность; при $TIME_Z(n) \leq O(n^2)$ – квадратичную сложность.

Определение 6.3. *Классом* **P** *называется множество вычислительных проблем, имеющих полиномиальную временну́ю оценку сложности:*

$$\mathbf{P} = \{Z : TIME_Z(n) \leq O(\mathscr{P}(n))\},$$

где $\mathscr{P}(n)$ *– полином от длины входа* n*.*

В современной теории сложности вычислений понятие полиномиального алгоритма может рассматриваться как уточнение интуитивного понятия "эффективный алгоритм", а класс **P** в этом смысле является определением "класса эффективно решаемых проблем".

Несложные вычисления позволяют убедиться, что целесообразно считать проблемы, имеющие полиномиальную оценку временно́й

сложности, эффективно решаемыми. Как уже упоминалось, при выполнении реальных вычислений всегда имеется ограничение на время решения задач. Считая, что условная единица времени равняется одному шагу выполнения алгоритма, обозначим предельное время и, соответственно, равное ему наибольшее допустимое число шагов вычислений на имеющемся в распоряжении пользователя компьютере через T.

Итак, время T является предельно допустимым. Если временна́я сложность проблемы есть $f(n)$, то должно выполняться неравенство $f(n) \leq T$, и уравнение

$$f(n) = T$$

позволяет найти предельную размерность задачи n^*, которую превысить на имеющемся компьютере нельзя.

Что же произойдёт, если заменить имеющийся компьютер на компьютер, работающий в 100 раз быстрее? Такая замена эквивалентна тому, что на имевшемся компьютере можно было бы в 100 раз увеличить допустимое время решения задач и получить возможность решать задачи большей размерности, равной n^{**}. Тогда новая предельная размерность определялась бы из уравнения

$$f(n) = 100T.$$

Поскольку $T = f(n^*)$, то предельная размерность n^{**} находится из уравнения

$$f(n) = 100f(n^*).$$

Вычисления по этой формуле, в частности, приводят к таким результатам.

а) Линейная временная сложность:

$$f(n) = n; \quad f(n) = 100n^*; \quad n = 100n^*; \quad n^{**} = 100n^*.$$

б) Квадратичная временная сложность:

$$f(n) = n^2; \; f(n) = 100n^{*2}; \; n^2 = 100n^{*2}; \; n^{**} = \sqrt{100} \times n^*; \; n^{**} = 10n^*.$$

в) Экспоненциальная временная сложность:

$$f(n) = 2^n; \; 2^n = 100 \times 2^{n^*}; \; n^{**} = n^* + \log_2 100; \; n^{**} \approx n^* + 6,644.$$

Приведенные вычисления показывают, что максимальная допустимая размерность решаемых задач при стократном увеличении быстродействия компьютера увеличивается в 100 раз только для проблем, имеющих линейную сложность. В случае квадратичной сложности – уже только в 10 раз, а *в случае экспоненциальной сложности* 2^n – *только* **на** 6 ! Так, справляясь с решением целочисленных уравнений не более чем с 1000 переменными при экспоненциальной сложности проблемы и получив с целью увеличения предельного числа переменных новый, в 100 раз более быстрый компьютер, можно добиться увеличения допустимого числа переменных только до 1006. Легко проверить: если заменить при тех же условиях имеющийся компьютер на другой, в $1000000 = 10^6$ раз более быстрый, то допустимое число переменных увеличится лишь до 1019. Сведенные в таблицу 6 расчёты убедительно показывают,

Таблица 6: Временная сложность и предельная размерность [5]

Временна́я сложность	Наибольший возможный размер входа на имеющемся компьютере	Наибольший возможный размер входа на в 100 раз более быстром компьютере
n	N_1	$100N_1$
n^2	N_2	$10N_2$
n^3	N_3	$4,64N_3$
n^5	N_4	$2,5N_4$
2^n	N_5	$N_5 + 6,64$
3^n	N_6	$N_6 + 4,19$

что линейный многократный рост быстродействия компьютеров не является радикальным средством, позволяющим существенно увеличить (более чем на небольшую аддитивную добавку) предельную размерность решаемых задач экспоненциальной временной сложности. В то же время, задачи, имеющие полиномиальную сложность, позволяют получить увеличение предельной размерности решаемых экспоненциальных задач в разы́. Особенно это проявляется при степени полинома, меньшей или равной 3.

Именно в силу указанного обстоятельства, и прежде всего – с теоретической точки зрения, вычислительные проблемы разделя-

ются на полиномиально (эффективно) разрешимые, составляющие класс **P**, *и все остальные, не принадлежащие этому классу.*

6.3 Проблемы вычисления свойств, недетерминированные вычисления, классы NP и NPC

Определение 6.4. *Проблемой распознавания свойств называется такая совокупность индивидуальных задач, результат алгоритмического решения которых может быть только бинарным (одним из двух фиксированных значений, например, "да" или "нет").*

Рис. 6.31: Левин Л. А. (1948) – советский и американский математик, ученик А.Н. Колмогорова

Можно сказать, что проблемы распознавания свойств предполагают вычисление функций вида $\varphi : \mathscr{D} \to \{0,1\}$, где $\mathscr{D}$ – множество допустимых описаний начальных данных – условий решаемых задач. Допустимыми описаниями начальных данных могут быть, например, такие "Обладает ли объект x свойством y?" или "$\exists y \in Y : P(x,y) = 1$", где $P : X \times Y \to \{0,1\}$ – предикат. В такой интерпретации X – это множество описаний индивидуальных задач, входящих в проблему, а Y – множество допустимых и подлежащих проверке значений y. Именно Y определяет область поиска при решении задачи распознавания свойства.

В теории сложности вычислений, основы которой были заложены Л. А. Левиным, С. А. Куком и Р. М. Карпом, рассматривается именно класс проблем распознавания свойств – сужение множества вычислительных проблем до совокупностей задач с бинарным ответом. В действительности класс проблем распознавания свойств достаточно содержателен: в нём, в частности, содержатся проблемы выяснения существования корней уравнений, циклов в графах, вы-

яснения факта доказуемости формул в формальных системах, выполнимости логических формул, полноты систем автоматов и многие другие.

Основные усилия математиков – создателей теории сложности вычислений Кука-Карпа-Левина были направлены на изучение природы сложности и свойств переборных задач (экспоненциальных по своей сущности, поскольку число 2^n характеризует полный перебор всех подмножеств n-элементного множества).

Идея, основанная на элиминации (исключении) перебора при исследовании проблем распознавания свойств привела к определению недетерминированных моделей вычислений.

Рассмотрим проблему поиска – такую проблему распознавания: существует ли элемент y в заданном множестве Y такой, что выполняется некоторый предикат $P(y)$:

$$\exists y \in Y \ (P(y) = 1)?$$

Рис. 6.32: Стивен Артур Кук (1939) – американский математик, специалист в области теории вычислений, лауреат премии Тьюринга.

В этой проблеме можно выделить две следующие подзадачи: нахождение элемента y и вычисления предиката $P(y)$ для найденного элемента y.

Так, если говорить о проблеме существования корня какого-нибудь уравнения, можно представить такие две подзадачи: 1) выбора (нахождения каким-либо образом) предполагаемого корня и 2) выяснения того, что найденное значение действительно является корнем, например, путём его подстановки в уравнение.

Но найти предполагаемый корень, как правило, гораздо труднее, чем при помощи подстановки (что является вычислением свойства) убедиться, что действительно найден корень.

Если считать, что выбор элемента y происходит *недетерминированным (некоторым никак не определяемым) образом*, что является глубочайшей математической абстракцией, и только потребовать, чтобы вычисление предиката (свойства) $P(x, y)$ было полиномиальным по временно́й сложности ($P \in \mathbf{P}$), то можно определить

класс вычислительных проблем распознавания с полиномиальной проверкой свойств – *класс* **NP** (**N***ondeterministic* **P***olynomial*).

Полагается, что недетерминированный выбор осуществляется соответствующим *недетерминированным алгоритмом за один абстрактный шаг.* При такой интерпретации любую строго определённую (детерминированную) вычислительную модель (язык) можно преобразовать в недетерминированную (**N**−язык), дополнив абстрактным оператором (одной командой) недетерминированного выбора, например

$$y := CHOOSE(Y),$$

и полагать по определению, что недетерминированный оператор $CHOOSE$ выбирает из конечного множества Y такой его элемент, что при последующей проверке свойства $P(x, y)$ в случае, если во множестве Y существует элемент y^*, для которого $P(x, y^*) = 1$, то оператор $CHOOSE$ выдаст именно этот y^*. В противном случае, когда в Y не существует такой элемент y^*, что $P(x, y^*) = 1$, оператор $CHOOSE$ выдаст произвольный $y \in Y$.

Опять же, пользуясь исключительно абстракцией, можно сказать, что оператор $CHOOSE$ *"угадывает" решение задачи.*

Будем полагать, что слова x и y состоят из букв конечных алфавитов.

Определение 6.5. *Классом проблем распознавания свойств с полиномиальной проверкой называется множество всех таких проблем распознавания свойств, которые состоят из задач, дающих на входном слове x длины n ответ "ДА" (1) тогда и только тогда, когда является истинным некоторый вычислимый (проверяемый) с полиномиальной по n временной сложностью предикат (свойство) $P(x, y)$, где y – некоторое слово, длина которого ограничена полиномом от n. В противном случае – ответ "НЕТ" (0) эквивалентен тому, что значение $P(x, y)$ является ложным или подходящее слово y не существует.*

Слово y часто называют сертификатом для слова x, а полиномиальный алгоритм, вычисляющий значение предиката $P(x, y)$, – алгоритмом проверки сертификата.

Так, если x описывает (задаёт) некоторое уравнение, и требу-

ется распознать: имеет ли это уравнение корень, то сертификатом будет любое значение y, предположительно являющееся описанием корня, а алгоритм подстановки этого описания (корня) в описание уравнения (в уравнение)– алгоритмом проверки сертификата.

Рис. 6.33: Ричард Мэннинг Карп (1935) – американский математик, специалист в области теории вычислений, лауреат премии А. Тьюринга.

Другой пример: x описывает ориентированный конечный граф, и требуется определить: существует ли в этом графе такой цикл, проходящий через все вершины, в котором каждая вершина встречается только один раз. Сертификатом y будет любой список (последовательность) вершин графа. Алгоритмом проверки сертификата – линейная по числу вершин проверка того, что данный список вершин образует цикл: каждая вершина графа входит в последовательность ровно один раз, и первая по списку вершина совпадает с последней. Заметим, что в произвольном графе возможно как существование некоторого множества таких циклов, так и отсутствие даже одного такого цикла.

В определении класса с полиномиальной проверкой свойства способ и сложность выбора сертификата (или, в частности, кандидата на решение) y не оговаривается: важно лишь то, что алгоритм проверки сертификата, вычисляющий предикат $P(x, y)$ должен иметь полиномиальную временную сложность.

В качестве $\mathbf{N}-$языка будем использовать недетерминированную машину Тьюринга (НДМТ), а обычную МТ будем называть детерминированной (ДМТ).

НДМТ является абстрактной моделью вычислений и имеет две головки: первую – как у обычной одноленточной, детерминированной МТ (ДМТ), а вторая головка является "угадывающей" и записывает на ленту слово y, которое является сертификатом-"догадкой" (ничего кроме этого угадывающая головка не делает). В этом смысле НДМТ реализует оператор недетерминированного выбора $CHOOSE$ и за полиномиальное по длине входного слова

x число шагов записывает его на ленту рядом со словом x. После этого НДМТ переходит в состояние q_0 и далее уже выполняет детерминированную программу (работая как ДМТ) проверки свойства – вычисления предиката $P(x, y)$. Если НДМТ "доходит" до конечного состояния $q_\sharp$, то она записывает на ленту единицу (ДА) или ноль (НЕТ). Если НДМТ никогда не останавливается, что может произойти, например, в силу несуществования подходящего y, то свойство $P(x, y)$ не выполняется.

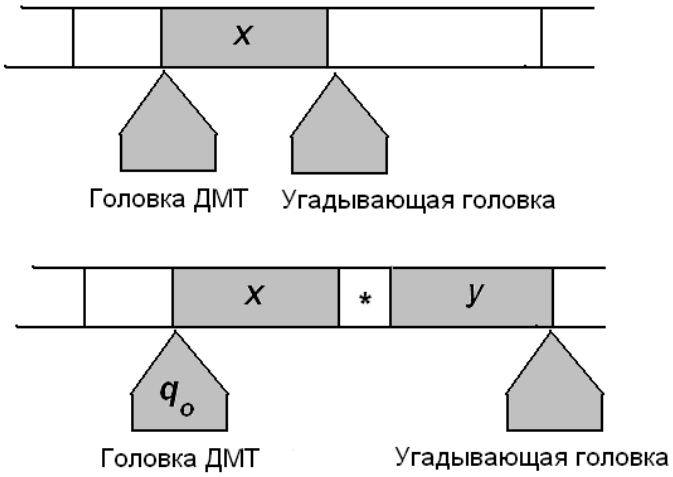


Рис. 6.34: Угадывание сертификата НДМТ

Определение 6.6. *Классом* **NP** *называется множество всех таких проблем распознавания свойств, для которых найдется недетерминированная машина Тьюринга, позволяющая решить каждую индивидуальную задачу проблемы за полиномиальное от длины входа задачи число шагов.*

В определении 6.6 вместо языка НДМТ может быть взят любой полиномиально эквивалентный тьюринговскому **N**−язык.

Очевидно, что следующий алгоритм на **N**−языке будет принадлежать классу **NP**, если только выполнение шага 3 – вычисление предиката $P(x, y)$ – реализуется с полиномиальной сложностью.

```
1  Ввод слова x; // За O(n) шагов - линейно по длине входа
2  y:=CHOOSE(Y); // Сертификат (y) угадывается за один шаг
3  Если P(x,y)=1 то вывод "ДА" иначе вывод "НЕТ";
4  end;
```

Таким образом, для того, чтобы установить принадлежность вычислительной проблемы распознавания свойства классу **NP**, *достаточно показать, что алгоритм проверки сертификата (вычисления $P(x, y)$), представленный на любом из полиномиально эквивалентных языков, имеет полиномиальную сложность.*

Поэтому определения 6.5 и 6.6 определяют один и тот же класс проблем распознавания свойств – класс **NP**.

Для конечных дискретных задач, разрешимых алгоритмически, зачастую можно не только указать множество $Y = Y(x)$ допустимых значений сертификата [15] – таких, что $y \in Y$, – но и оценить его мощность При этом число элементов в Y может расти с ростом размера входа n экспоненциально и даже ещё быстрее.

Если слово-сертификат y состоит из букв r-значного алфавита и имеет длину $len(y)$, оцениваемую полиномом $\mathscr{P}(n)$, где n – размер входа x, то множество Y будет содержать не менее $r^{\mathscr{P}(n)}$ элементов. А в случае, например, когда $len(y) = n$, и вычисляется некоторое свойство на всевозможных перестановках n-элементного множества, Y будет содержать $n! > \sqrt{2\pi n}(\frac{n}{e})^n$ вариантов.

Рис. 6.35 поясняет, что, согласно определению 6.6, решение задачи распознавания свойства **N**−алгоритмом можно представить как последовательное выполнение вычисления сертификата и его полиномиальной проверки. Зафиксированная в определении 6.6 полиномиальность проверки сертификата не снижает временну́ю вычислительную сложность исходной задачи, поскольку в задачах, входящих в **NP**, сложность нахождения сертификата элиминируется "догадкой".

Очевидно что $\mathbf{P} \subset \mathbf{NP}$, поскольку в случае, когда проблема принадлежит классу **P**, все её индивидуальные задачи разрешимы за полиномиальное время, а если задача распознавания свойства вычислима полиномиально, то входящая в неё или ею собственно являющаяся проверка свойства вычисляется не более чем за полином шагов. Иначе говоря, не исключается, что временная сложность нахождения сертификата может оказаться полиномиальной, и тогда решение исходной задачи (см. 6.35) может быть сведено к последо-

[15] Множество допустимых значений сертификата Y определяется по смыслу каждой конкретной задачи распознавания свойства, поэтому применена запись $Y(x)$

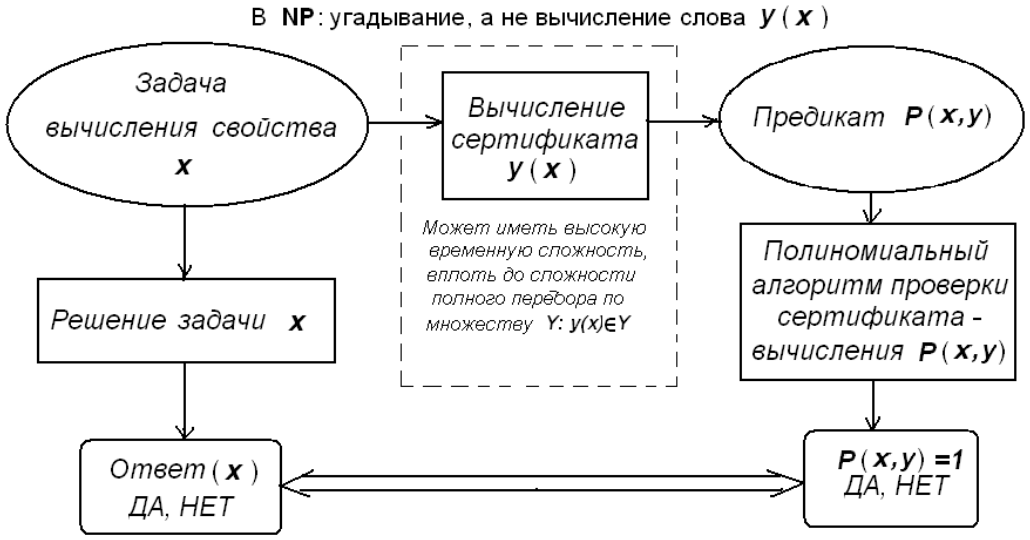


Рис. 6.35: Схема, поясняющая определение класса **NP**

вательному выполнению двух полиномиальных алгоритмов, что в итоге даёт полиномиальный алгоритм решения.

Вопрос о равенстве классов – действительно ли $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ – является одной из центральных открытых [16] проблем математики уже более трёх десятилетий. Эта проблема является одной из семи проблем тысячелетия, за решение которой Математический институт Клэя [17]назначил премию в миллион долларов США. Большинство математиков склонны считать, что $\mathbf{P} \neq \mathbf{NP}$, поскольку в противном случае можно было бы решить любую задачу распознавания свойств за полиномиальное время, но этому противоречит тот факт, что за всю историю математики для многих переборных задач полиномиальные алгоритмы их решения так и не были найдены.

[16]на год издания данной книги

[17]Математический институт Клэя — частная некоммерческая организация, расположенная в Кембридже, штат Массачусетс. Институт Клэя получил широкую известность после объявления 24 мая 2000 года списка Проблем тысячелетия (Millennium Prize Problems). Семь проблем этого списка были определены как "важные классические задачи, решение которых не найдено в течение многих лет". За решение каждой из проблем предложен приз в 1 миллион долларов. Одна из семи проблем тысячелетия (гипотеза Пуанкаре) решена российским математиком Г.Перельманом.

Определение 6.7. *Говорят, что проблема распознавания свойств W полиномиально сводится к проблеме распознавания свойств Z (что обозначается $W \propto Z$), если существует алгоритм (машина Тьюринга) A, которая, получив на вход любую задачу (слово) $w \in W$ преобразует это слово в слово (задачу) $A(w) = z$, $z \in Z$, причём алгоритм A имеет полиномиальную временную сложность, и задача z даёт ответ "ДА" тогда и только тогда, когда даёт ответ "ДА" задача w.*

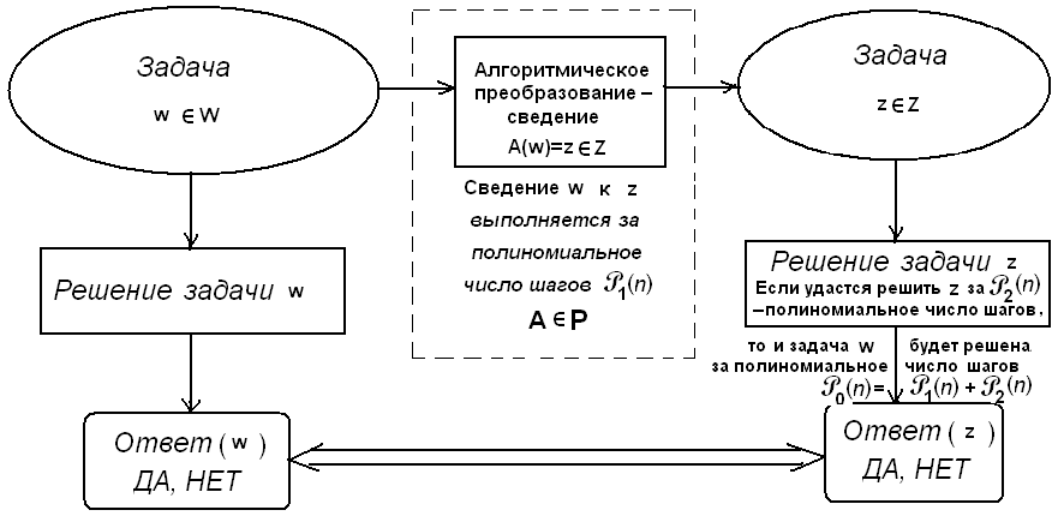


Рис. 6.36: Схема, поясняющая определение полиномиальной сводимости

Понятие полиномиальной сводимости можно объяснить следующим образом (рис. 6.36). Получить алгоритмическое решение произвольной задачи распознавания свойства w можно двумя способами. Первый – решать её напрямую, а второй – преобразовать её в другую алгоритмическую задачу $z = A(w)$, которая даёт ответ ДА тогда и только тогда, когда и задача w даёт ответ ДА. Именно алгоритмическое преобразование A и является алгоритмом полиномиальной сводимости – т.е. имеющим полиномиальную временную сложность ($A \in \mathbf{P}$). Требование $A \in \mathbf{P}$ снимает необходимость указывать какую-либо конкретную алгоритмическую модель, реализующую A, поскольку любая из полиномиально эквивалентных моделей алгоритмов, заменяющая машину Тьюринга в определении сводимости, сохранит полиномиальность A.

Определением полиномиальной сводимости на множестве проблем распознавания свойств вводится бинарное отношение "$\propto$" , которое, очевидно, является рефлексивным и транзитивным:

$$W \propto W;\ \ (W \propto V)\ \wedge\ (V \propto Z)\ \Longrightarrow W \propto Z.$$

Теорема 6.3. $(W \propto Z) \wedge (Z \in \mathbf{P}) \Longrightarrow W \in \mathbf{P}$.

Доказательство. Поскольку имеет место полиномиальная сводимость $W \propto Z$, любую индивидуальную задачу $w \in W$ можно преобразовать за полиномиальное число шагов $\mathscr{P}_1(n)$ в задачу $z \in Z$, и решив затем задачу z, получить решение, эквивалентное решению исходной задачи w. Поскольку $Z \in \mathbf{P}$, задача z может быть решена за полиномиальное число шагов $\mathscr{P}_2(n)$ (см. рис. 6.36). Тогда решение задачи w может выполнено в два этапа с полиномиальной суммарной сложностью $\mathscr{P}_0(n) = \mathscr{P}_1(n) + \mathscr{P}_2(n)$.

Следствие. Для того, чтобы доказать, что исследуемая проблема вычисления свойства полиномиально разрешима, достаточно показать, что она полиномиально сводится к какой-нибудь проблеме из класса $\mathbf{P}$.

Теорема 6.4. *Любая проблема распознавания свойства из класса* $\mathbf{NP}$ *может быть решена детерминированным алгоритмом с временной сложностью, оцениваемой как* $O(2^{\mathscr{P}(n)})$, *где* $\mathscr{P}(n)$ – *некоторый полином от размера* n *входа.*

Доказательство. Пусть Z – произвольная проблема распознавания свойства из класса $\mathbf{NP}$. Тогда недетерминированное угадывание сертификата y, – слова над r-символьным конечным алфавитом, имеющего полиномиальную длину $\mathscr{P}_y(n)$, можно заменить полным детерминированным перебором $r^{\mathscr{P}_y(n)}$ всех возможных вариантов сертификата. И для каждого варианта сертификата последовательно осуществлять полиномиальную проверку свойства за $\mathscr{P}_{\text{св}}(n)$ шагов. Тогда такой детерминированный переборный алгоритм обеспечит решение любой индивидуальной задачи проблемы Z за время (число шагов)

$$const \cdot \mathscr{P}_{\text{св}}(n) \cdot r^{\mathscr{P}_y(n)} = const \cdot 2^{log_2 \mathscr{P}_{\text{св}}(n)} \cdot 2^{log_2 r \cdot \mathscr{P}_y(n)} = O(2^{\mathscr{P}(n)}).$$

Определение 6.8. *Классом* **NPC** *называется множество всех таких проблем Z распознавания свойств, которые удовлетворяют следующим двум условиям:*

I. *Z входит в класс* **NP***;*

II. *любая проблема из* **NP** *полиномиально сводится к Z.*

Проблемы, принадлежащие классу **NPC**, называют **NP**- *полными или универсальными переборными проблемами.*

Теорема 6.5.

$$(W \in \mathbf{NP}) \wedge (Z \propto W) \wedge (Z \in \mathbf{NPC}) \Longrightarrow W \in \mathbf{NPC}.$$

Доказательство. Действительно, для того, чтобы показать, что некоторая проблема W является **NP**-полной, согласно определению 6.8 достаточно убедиться, что $W \in \mathbf{NP}$ и что любая проблема из **NP** полиномиально сводится к W. Требование $W \in \mathbf{NP}$ уже содержится в условии теоремы. Поскольку $Z \in \mathbf{NPC}$, имеем

$$\forall S \in \mathbf{NP}\ (S \propto Z).$$

По условию теоремы также имеем сводимость $Z \propto W$. И по транзитивности отношения сводимости получаем

$$\forall S \in \mathbf{NP}(S \propto Z) \wedge (Z \propto W) \Longrightarrow \forall S \in \mathbf{NP}(S \propto W) \Leftrightarrow W \in \mathbf{NPC}.$$

Следствие. Для того, чтобы доказать, что исследуемая проблема вычисления свойства является **NP**-полной, достаточно показать, что она содержится в классе **NP** и к ней полиномиально сводится какая-нибудь заведомо **NP**-полная проблема.

6.4 Установление NP-полноты вычислительных проблем распознавания свойств

Честь открытия первой **NP**-полной проблемы – *выполнимости конъюнктивной нормальной формы* (КНФ) принадлежит С. Куку.

Эта проблема, которая будет обозначаться $Z_{\text{ВЫП}}$ или SAT (*Boolean Satisfiability Problem*), состоит в следующем.

Дана КНФ

$$K(x_1,...,x_n) = (x_{11}^{\sigma_{11}} \vee \cdots \vee x_{1k_1}^{\sigma_{1k_1}})\& \ldots \&(x_{m1}^{\sigma_{m1}} \vee \cdots \vee x_{mk_m}^{\sigma_{mk_m}})$$

$$= \bigwedge_{j=1}^{m} (\bigvee_{i=1}^{k_j} x_{ji}^{\sigma_{ji}}).$$

Требуется распознать свойство: существует ли набор из n значений логических переменных $x_1, ..., x_n$, на котором КНФ $K(x_1, ..., x_n)$ обращается в единицу.

КНФ является логическим произведением m скобок с номерами $j = 1, ..., m$, в каждой из которых содержится дизъюнкция k_j литералов. Поэтому номера переменных, входящих в скобки, помечены парами символов ji, которые следует понимать так: x_{ji} – это некоторая переменная из множества $x_1, ..., x_n$, находящаяся в $j-$й скобке и записанная $i-$й по порядку слева направо. Литерал y^τ, как принято, означает y, если $\tau = 1$ или $\bar{y}$, если $\tau = 0$.

Теорема 6.6. $Z_{ВЫП} \in \mathbf{NPC}$.

Доказательство. Достаточно убедиться в том, что $Z_{\text{ВЫП}} \in \mathbf{NP}$ и любая проблема из класса **NP** полиномиально сводится к $Z_{\text{ВЫП}}$.

1. Длина входа любой задачи выполнимости есть $O(mn)$ (в каждой из m скобок не более n литералов). Сертификатом является булевый вектор длины n. Чтобы проверить, обращает ли вектор в единицу КНФ, нужно подставить его в формулу КНФ и убедиться, что все входящие в неё скобки обращаются в единицу. Для каждой скобки будет выполнено не более n подстановок, и такие подстановки будут выполнены не более чем в m скобках (первая же нулевая скобка влечёт отрицательный ответ). Таким образом, проверка сертификата потребует $O(mn)$ шагов и, следовательно, является полиномиальной, что означает: $Z_{\text{ВЫП}} \in \mathbf{NP}$.

2. Для любой задачи из **NP** *полиномиальность алгоритма распознавания свойства в точности означает существование детерминированной машины Тьюринга, которая осуществляет проверку сертификата за полиномиальное число шагов*, т.е. существование

МТ, которая вычисляет предикат $P(x, y)$, когда сертификат y уже получен (см. рис. 6.34). Идея доказательства Кука состоит в построении за полиномиальное число шагов такой КНФ, которая будет выполнима тогда и только тогда, когда указанная машина Тьюринга, проверяющая сертификат, даёт ответ 1 ("ДА").[18]

Определим логические переменные для построения КНФ, эквивалентной машине Тьюринга, проверяющей сертификат. Для этого возьмём произвольную проверяющую МТ с k-элементным ленточным алфавитом $A = a_0, ..., a_i, ..., a_{k-1}$ и r-элементным алфавитом состояний $Q = q_0, ..., q_j, ..., q_{r-1}$. Поскольку МТ выполняет проверку за полиномиальное число шагов, обозначим это число шагов T, а текущий номер шага обозначим $t \in \{1, 2, ..., T\}$.

Если МТ при выполнении вычислений совершает T шагов, то, двигаясь только в одну сторону, она не сможет пройти более T ячеек. Поэтому рабочую зону на ленте можно ограничить номерами от $-T$ до T и полагать, что МТ начинает работать, обозревая ячейку с номером $s = 1$; переменная s, таким образом, будет обозначать текущий номер обозреваемой ячейки ленты и содержаться в целочисленном отрезке от $-T$ до T.

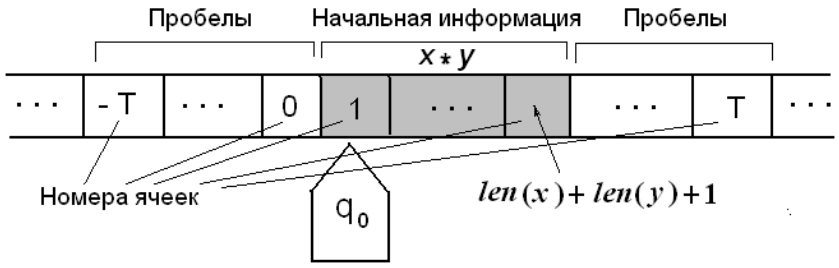


Рис. 6.37: Начальное состояние МТ и нумерация ячеек ленты

Будем использовать следующие *вводимые по определению* логические переменные:

$v_t^j = 1 \Leftrightarrow$ на шаге t МТ находится в состоянии q_j; иначе $v_t^j = 0$;

$w_{s,t} = 1 \Leftrightarrow$ на шаге t МТ обозревает ячейку ленты с номером s;

$u_{s,t}^i = 1 \Leftrightarrow$ на шаге t в ячейке s содержится символ a_i.

[18]На первый взгляд, сравнение МТ с КНФ может показаться неожиданным, поскольку МТ – динамический логический объект, а КНФ – статический. "Динамика" в вычисление КНФ будет введена путём применения переменных с индексами – номерами шагов МТ, номерами ячеек ленты, номерами символов входного алфавита и алфавита состояний

Учитывая, что на ленте могут быть записаны как символы алфавита A (из символов этого алфавита состоят слова x), так и символы алфавита $\mathscr{Y}$ (из символов этого алфавита состоят слова y) и пробелы, будем считать, что индекс i может принимать $\sigma = k+d+1$ значений, где d – число символов в алфавите $\mathscr{Y}$.

Таким образом, всего введено

$$\mu = T \times k + (2T+1) \times T + (2T+1) \times T \times \sigma$$

логических переменных, причём μ – полином от длины входа x, поскольку r, k, d – константы, а $T = \mathscr{P}(n)$.

Теперь запишем в форме логических условий соотношения, точно определяющие правильную работу МТ по проверке сертификата. В силу функциональной полноты представления логических функций в виде КНФ, это является возможным.

Логическое условие $C = 1 \Leftrightarrow$ *в каждой ячейке s на каждом шаге t записан ровно один символ.* Запишем вспомогательное выражение

$$C_{s,t} = \underbrace{\left(\bigvee_{0 \le i \le k-1} u^i_{s,t} \right)}_{\text{какой-нибудь символ}} \wedge \underbrace{\left(\bigwedge_{0 \le i < \tau \le k-1} (\bar{u}^i_{s,t} \vee \bar{u}^\tau_{s,t}) \right)}_{\text{причём только один – никакие два вместе}}.$$

Поскольку $C_{s,t} = 1 \Leftrightarrow$ в произвольной ячейке s на произвольном шаге t записан ровно один символ, получаем

$$C = \bigwedge_{-T \le s \le T} \bigwedge_{1 \le t \le T} C_{s,t}.$$

Далее, поскольку промежутки изменения индексов s, t, i, j уже определены, иногда будем пользоваться упрощённой записью:

$$C = \bigwedge_s \bigwedge_t C_{s,t}.$$

Логическое условие D будет равно единице тогда и только тогда, когда МТ *на каждом шаге находится ровно в одном состоянии*:

$$D = \bigwedge_t D_t;$$

$$D_t = \underbrace{\left(\bigvee_j v_t^j \right)}_{\text{в каком-нибудь состоянии}} \wedge \underbrace{\left(\bigwedge_{0 \le j < m \le r-1} (\bar{v}_t^j \vee \bar{v}_t^m) \right)}_{\text{причём ровно в одном}}.$$

Логическое условие E будет равно единице тогда и только тогда, когда МТ *на каждом шаге обозревает ровно одну ячейку ленты*:

$$E = \bigwedge_t E_t;$$

$$E_t = \underbrace{\left(\bigvee_s w_{s,t} \right)}_{\text{какую-нибудь ячейку}} \wedge \underbrace{\left(\bigwedge_{-T \le s < q \le T} (\bar{w}_{s,t} \vee \bar{w}_{q,t}) \right)}_{\text{причём только одну}}.$$

Логическое условие F будет равно единице тогда и только тогда, когда *начальная конфигурация МТ имеет вид* $q_0 x * y$, что означает: находясь в начальном состоянии q_0 МТ на шаге 0 обозревает первый символ слова x, за которым через разделитель $*$ записан сертификат y, причём слово $x * y$ слева и справа в рабочей зоне окаймлено пробелами. Слово x, представляющее произвольную индивидуальную задачу, задаётся вместе со своей длиной $len(x)$.

Будем обозначать $\mathscr{Y}$ конечный алфавит, из символов которого состоит любое слово-сертификат y, с добавление символа пробела.

$$F = v_0^0 w_{1,0} \wedge \underbrace{\left(\bigvee_{-T \le s \le 0} u_{s,0}^{\Lambda} \right)}_{\text{пустые ячейки слева}} \wedge \underbrace{\left(\bigvee_{1 \le m \le len(x)} u_{m,0}^{x(m)} \right)}_{\text{затем слово x}} \wedge \underbrace{u_{len(x)+1,0}^{*}}_{\text{звёздочка}} \wedge$$

$$\wedge \underbrace{\bar{u}_{len(x)+2}^{\Lambda} \left(\bigwedge_{len(x)+2 \le s \le T} \left(\bigvee_{\alpha \in \mathscr{Y}} u_{s,0}^{\alpha} \right) \right)}_{\text{затем непустое слово у некоторой длины}} \wedge$$

$$\wedge \underbrace{\left(\bigwedge_{len(x)+2 \le s \le T-1} (u_{s,0}^{\Lambda} \to u_{s+1,0}^{\Lambda}) \right)}_{\text{и после первой пустой ячейки справа идут пустые ячейки}},$$

где Λ – *номер символа* "пусто", $*$ – номер символа "звёздочка". Заметим, что длина слова-сертификата y, оставаясь полиномиальной от длины входа x, в отличие от неё заранее неизвестна. Именно этот факт отражается второй и третьей строчкой формулы F.

Следующее условие G будет равно единице тогда и только тогда, когда МТ точно выполняет свою программу. Такая программа существует в силу существования этой проверяющей машины (полиномиального алгоритма проверки свойства) и состоит из команд

$$q_j a_i \to q_{\varphi(j,i)} a_{\psi(j,i)} d_{j,i},$$

где φ – функция, определяемая таблицей переходов, а ψ – функция, определяемая таблицей выходов, $d_{j,i}$ – сдвиг головки МТ при переходе из состояния q_j по входу a_i. Функцию сдвига переопределим следующим образом:

$$d_{j,i} = \begin{cases} -1, & \text{если сдвиг влево;} \\ 0, & \text{если сдвиг отсутствует;} \\ 1, & \text{если сдвиг вправо.} \end{cases}$$

Выполнение одной команды на шаге t описывается логической формулой – импликацией

$$K_{s,t}^{i,j} = \underbrace{(v_t^j u_{s,t}^i w_{s,t} \to v_{t+1}^{\varphi(j,i)} u_{s,t+1}^{\psi(j,i)} w_{s+d_{j,i},\ t+1})}_{\text{если на шаге t МТ обозревает ячейку } s} \wedge \underbrace{(u_{s,t}^i \bar{w}_{s,t} \to u_{s,t+1}^i)}_{\text{иначе ячейка не изменяется}} .$$

$$G = \bigwedge_s \bigwedge_t \bigwedge_i \bigwedge_j K_{s,t}^{i,j}.$$

Последнее логическое условие H истинно тогда и только тогда, когда на последнем шаге T МТ остановится в конфигурации $q_\sharp 1$, где $\sharp$ – номер заключительного состояния.

$$H = v_T^\sharp \wedge \underbrace{\left(\bigwedge_{-T \le s \le T} (u_{s,T}^\Lambda \vee u_{s,T}^1) \right)}_{\text{только символы пусто или 1}} \wedge \underbrace{\left(\bigwedge_{-T \le s \le q \le T} (\bar{u}_{q,T}^1 \vee \bar{u}_{s,T}^1) \right)}_{\text{причём только одна единица}} \wedge$$

$$\wedge \quad \underbrace{\left(\bigwedge_{-T \le s \le T} (w_{s,T} \longrightarrow u_{s,T}^1) \right)}_{\text{остановившись на ячейке s, видит в ней 1}} .$$

Логическое условие

$$CORR = C \wedge D \wedge E \wedge F \wedge G \wedge H$$

корректно описывает работу МТ, проверяющую сертификат для любой задачи из **NP**. Если в формуле $CORR$ произвести эквивалентные логические преобразования вида

$$\alpha \to \beta = \bar{\alpha} \vee \beta,$$

$$\overline{\alpha\beta\gamma} = \bar{\alpha} \vee \bar{\beta} \vee \bar{\gamma},$$

$$\alpha \to \beta\gamma\tau = (\bar{\alpha} \vee \beta) \wedge (\bar{\alpha} \vee \gamma) \wedge (\bar{\alpha} \vee \tau),$$

то можно представить $CORR$ в конъюнктивной нормальной форме – $CORR_{\text{КНФ}}$ от переменных $v_t^j, u_{s,t}^i, w_{s,t}; t = 1, ..., T;\ j = 0, 1, ..., r-1;$ $i = 0, 1, ..., k-1$. Поскольку T – полином от длины входа, то число перечисленных переменных, которые входят в формулу $CORR_{\text{КНФ}}$, также оценивается полиномом от длины входа.

Очевидно, что $CORR_{\text{КНФ}}$ построена за полиномиальное от длины входа число шагов и равна 1 тогда и только тогда, когда НДМТ, угадав сертификат, начнёт работу в конфигурации $q_0 x * y$, и совершив t шагов остановится в конфигурации $q_\sharp\ 1$.

Действительно, если $CORR_{\text{КНФ}} = 1$, то взяв выполняющие её значения переменных из множества

$$\{u_{s,0}^\alpha,\ len(x) + 2 \le s \le T,\ \ \alpha \in \mathscr{Y}^*\},$$

где $\mathscr{Y}^*$ – алфавит символов сертификата, не содержащий пробела, можно получить слово y такое, что МТ переводит $q_0 x * y$ в $q\sharp 1$, и тогда $P(x, y) = 1$.

Обратно, если $P(x, y) = 1$, то назначив значения переменным $v_t^j, u_{s,t}^i, w_{s,t}$ по результату работы МТ над начальной конфигурацией $q_0 x * y$, получим набор значений, на котором $CORR_{\text{КНФ}} = 1$.

Доказательство того, что любая проблема из класса **NP** полиномиально сводится к $Z_{\text{ВЫП}}$, завершено. $\square$

Имея **NP**−полную проблему $Z_{\text{ВЫП}}$, можно установить **NP**−полноту другой, исследуемой проблемы $Z \in \mathbf{NP}$, если удастся

доказать, что $Z_{\text{ВЫП}}$ полиномиально сводится к Z ($Z_{\text{ВЫП}} \propto Z$). Рассмотрим, как это осуществляется в случае исследования проблемы о m-полном подграфе.

Проблема $m-$полный подграф, обозначаемая далее $Z_{mПП}$, состоит в следующем. Задан конечный неориентированный граф Γ с M вершинами. Требуется установить: имеется ли в графе Γ полный подграф с m вершинами – такой подграф, в котором каждая пара вершин соединена ребром.

Теорема 6.7. $Z_{mПП} \in \mathbf{NPC}$.

Доказательство.

1. $Z_{k\text{ПП}} \in \mathbf{NP}$. Принадлежность классу **NP** следует из следующего проверяющего алгоритма.

Множество U рёбер графа может быть задано как множество пар вершин (двумерный массив) такой, что $u[i,j] = 1$, если в графе Γ существует ребро (i,j), иначе $u[i,j] = 0$; $i = 1, \ldots M; j = 1, \ldots M$. Номера указанных для проверки m вершин можно считать заданными одномерным массивом L, содержащим список m вершин проверяемого подграфа: $\{i_1, i_2 \ldots, i_m\}$. Выполнив алгоритм

```
1   ОТВЕТ =1;
2   Для q от 1 до m-1      //  Ответ будет равен нулю, если
3   Для r от q + 1 до m    // хотя бы одна пара из m вершин
4   Если u[L[q],L[r]]= 0 то ОТВЕТ =0; // не связана ребром,
5   end;                   // иначе ответ равен 1
```

можно проверить свойство m-полноты подграфа за $O(m(m-1)/2)$ шагов, что не превысит полинома от размера входа задачи, равного $const \times M^2$ при задании графа матрицей связности U с учётом того, что $m < M$.

2. $Z_{\text{ВЫП}} \propto Z_{k\text{ПП}}$. Доказательство этой полиномиальной сводимости состоит в построении по заданной КНФ

$$K(x_1, ..., x_n) = (x_{11}^{\sigma_{11}} \vee \cdots \vee x_{1k_1}^{\sigma_{1k_1}}) \& \ldots \& (x_{m1}^{\sigma_{m1}} \vee \cdots \vee x_{mk_m}^{\sigma_{mk_m}})$$

графа Γ_K, который содержит $m-$полный подграф тогда и только тогда, когда КНФ $K = K(x_1, ..., x_n)$ выполнима.

Если КНФ K выполнима, то в каждой из её m скобок имеется положительный литерал. На этом основывается следующее построение графа Γ_K.

Положим, что $m(k_1 + ... + k_m)$ вершин графа будут соответствовать всем литералам КНФ K, причём каждому литералу припишем номер содержащей его скобки. Список вершин графа Γ_K будет иметь вид

$$Nodes = \{(x_{11}^{\sigma_{11}}, 1), (x_{12}^{\sigma_{12}}, 1), \ldots, (x_{mk_m}^{\sigma_{mk_m}}, m)\}.$$

Таким образом, каждая вершина получает две пометки:

(*литерал, номер содержащей его скобки*).

Множество рёбер графа Γ_K построим по списку его вершин так, что пара вершин из списка $Nodes$ соединяется ребром тогда и только тогда, когда

а) литералы, входящие соединяемые вершины, находятся в разных скобках КНФ K (вторые пометки не совпадают)

б) и при этом один из этих литералов не является инверсией другого.

Сложность построения рёбер графа Γ_K, очевидно, оценивается как

$$O((m(k_1 + ... + k_m))^2),$$

полиномиально от длины КНФ K. Граф Γ_K построен.

Пусть КНФ K выполнима. Тогда в каждой её скобке имеется литерал, принявший значение 1. Выберем ровно m таких литералов – по одному из каждой скобки. Поскольку они все приняли значение единица, то ни один из них не является инверсией другого. И при этом все они находятся в разных скобках. Следовательно, по построению графа Γ_K, каждая пара вершин, соответствующих выбранным литералам в списке $Nodes$, будет соединена ребром. Поэтому граф Γ_K будет содержать $m-$полный подграф на этих m выбранных вершинах.

Если же граф Γ_K содержит $m-$полный подграф, то взяв вершины этого подграфа, получим список m литералов – по одному из каждой скобки – таких, что ни один из них не является инверсией другого. Тогда всем им можно назначить единичное значение: если литерал имеет вид $x_{ji}^{\sigma_{ji}}$, то принять $x_{ji} = \sigma_{ji}$; таким путём будет получен выполняющий набор значений переменных для КНФ K. □

Для доказательства **NP**−полноты методом полиномиальной сводимости часто бывает удобно использовать проблему $Z_{3-\text{ВЫП}}$, состоящую в следующем.

Проблема $Z_{3-\text{ВЫП}}$. Дана КНФ, в каждой скобке которой содержится дизъюнкция ровно трёх литералов (3-КНФ) . Вычислить свойство: выполняется ли такая КНФ (существует ли набор значений входящих в КНФ переменных, обращающий её в единицу)?

Теорема 6.8. $Z_{3-ВЫП} \in \mathbf{NPC}$.

Доказательство. Полиномиальность проверки того, что на заданном наборе значений переменных 3-КНФ обращается в единицу, очевидна. Сложность этой проверки не превысит длины данной 3-КНФ, умноженной на константу. Поэтому $Z_{3-\text{ВЫП}} \in \mathbf{NP}$.

Покажем, что к проблеме $Z_{3-\text{ВЫП}}$ полиномиально сводится проблема $Z_{\text{ВЫП}}$, о которой уже известно, что она **NP**-полна. Для этого преобразуем исходную КНФ следующим образом.

Если в скобке КНФ содержатся два литерала $(\alpha \vee \beta)$, то, введя новую логическую переменную u заменим эту скобку на

$$(\alpha \vee \beta \vee u) \wedge (\alpha \vee \beta \vee \bar{u}) \equiv (\alpha \vee \beta).$$

Если в скобке содержится единственный литерал (α), то его замена на выражение

$$(\alpha \vee \gamma) \wedge (\alpha \vee \bar{\gamma}) \equiv \alpha$$

с новой переменной γ позволит далее применить представленное выше преобразование для скобки с двумя литералами.

Если же в скобке содержится $k > 3$ литералов, и тогда она имеет вид

$$(x_1^{\sigma_1} \vee x_2^{\sigma_2} \vee \cdots \vee x_k^{\sigma_k}), \tag{45}$$

то путём введения $(k-3)$-х новых переменных $z_1, z_2, ..., z_{k-3}$ дизъюнкт (45) заменим на конъюнкцию (46), состоящую из $k-2$ скобок, в каждой из которых будет содержаться ровно 3 литерала:

$$(x_1^{\sigma_1} \vee x_2^{\sigma_2} \vee z_1) \wedge (x_3^{\sigma_3} \vee \bar{z_1} \vee z_2) \wedge (x_4^{\sigma_4} \vee \bar{z_2} \vee z_3) \wedge \ldots$$

$$\cdots \wedge (x_{k-2}^{\sigma_{k-2}} \vee \overline{z_{k-4}} \vee z_{k-3}) \wedge (x_{k-1}^{\sigma_{k-1}} \vee x_k^{\sigma_k} \vee \overline{z_{k-3}}). \tag{46}$$

Покажем, что если существует набор значений литералов $x_1^{\sigma_1}, ..., x_k^{\sigma_k}$, выполняющий дизъюнкт (45), то найдётся такой набор значений $\delta_1, ..., \delta_{k-3}$ дополнительных введенных переменных

$z_1, \ldots, z_{k-3}$, что каждый из $k-2$ дизъюнктов в формуле (46) будет также выполнен, и тогда будет выполнено выражение (46).

Действительно, если имеется набор значений литералов , выполняющий дизъюнкт $(x_1^{\sigma_1} \vee ... \vee x_k^{\sigma_k})$, то в таком дизъюнкте найдётся хотя бы один литерал $x_q^{\sigma_q}$, $1 \leq q \leq k$, обращающийся в единицу. Он входит в выражении (46) в скобку с номером $q-1$ при $q \geq 3$ (или в первую скобку при $q = 1$ или 2, тогда первая скобка заведомо выполнима). Тогда при $q \geq 3$, положив $z_i = 1$ для всех $i \leq q-2$ получим, что в каждой i-той из $q-2$ первых скобок выражения (46) будет содержаться единица. В $(q-1)$-й скобке уже имеется единичный литерал $x_q^{\sigma_q}$. Положив далее $z_i = 0$ для всех скобок с номерами $i > q-1$, получим $\bar{z}_i = 1$ во всех остальных скобках от q-й до k-той. В итоге будем иметь выполнимость всех скобок выражения (46).

Теперь предположим, что выражение (46) выполнимо, и покажем, что тогда выполним хотя бы один литерал $x_i^{\sigma_i}$ в дизъюнкте (45). От противного, пусть все литералы в (45) отрицательны. Тогда первые $k-3$ дизъюнкта удовлетворены только тогда, когда $z_i = 1$, $i = 1, 2, ..., k-3$. Но при $z_{k-3} = 1$ не выполняется последний дизъюнкт $(x_{k-1}^{\sigma_{k-1}} \vee x_k^{\sigma_k} \vee \overline{z_{k-3}})$.

Непосредственным подсчётом можно убедиться в полиномиальности эквивалентного преобразования дизъюнкта (45) в произведение дизъюнктов с тремя литералами в каждом (46) и далее – в полиномиальности преобразования любой КНФ в 3-КНФ. □

Проблема $Z_{qВПГ}$ о q-вершинном покрытии графа состоит в следующем. Дан граф $\Gamma = < X, U >$, имеющий $n = |X|$ вершин и $m = |Y|$ рёбер. Существует ли в графе Γ подмножество $W_q \subset X$ из q вершин такое, что для любого ребра графа Γ во множестве W_q найдётся вершина, инцидентная этому ребру (являющаяся одной из вершин, входящих в это ребро)?

Теорема 6.9. $Z_{qВПГ} \in$ **NPC**.

Доказательство. Сведём **NP**-полную проблему $Z_{k\text{ПП}}$ о k-полном подграфе в графе с n вершинами к проблеме $Z_{\text{qВПГ}}$, где $q = n-k$. Для этого построим граф Γ', являющийся дополнением графа $\Gamma = < X, U >$, $|X| = n$, $|Y| = m$. Чтобы построить граф Γ', берутся все вершины X, попарно соединяются рёбрами, и из полученного полного графа удаляются рёбра, входящие во множество U.

Пусть $V \subset X$ произвольное подмножество из k вершин. Покажем, что k-полный подграф на этих вершинах является подграфом графа Γ тогда и только тогда, когда дополнение $V^{'} = X \setminus V$ множества вершин V образует k вершинное покрытие графа $\Gamma^{'}$.

Действительно, предположим, что k-полный подграф с вершинами множества V содержится в графе Γ. Рассмотрим произвольное ребро графа-дополнения $\Gamma^{'}$. Если бы обе вершины этого ребра содержались во множестве V, то это ребро принадлежало бы графу Γ, а не графу $\Gamma^{'}$. Поэтому ни одно ребро полного k-подграфа не содержится в графе-дополнении $\Gamma^{'}$, и, следовательно, все рёбра этого графа $\Gamma^{'}$ инцидентны хотя бы одной вершине из $V^{'}$.

Обратно, если дополнение $V^{'} = X \setminus V$ образует вершинное покрытие графа $\Gamma^{'}$, то *любое* ребро, имеющее концы во множестве V, не может принадлежать графу $\Gamma^{'}$, а принадлежит графу Γ, что равносильно существованию в Γ k-полного подграфа.

Сведение проблемы $Z_{k\text{ПП}}$ к проблеме $Z_{\text{qВПГ}}$, таким образом, состоит в построении по исходному n-вершинному графу его дополнения с полиномиальной временно́й сложностью $O(n^2)$. $\square$

Проблема $Z_{СЦЛНБ}$ совместности системы целочисленных линейных неравенств с булевыми переменными состоит в следующем.

Дана система неравенств

$$\begin{cases} a_{11}y_1 + a_{12}y_2 + ... + a_{1n}y_n \geq b_1; \\ a_{21}y_1 + a_{22}y_2 + ... + a_{2n}y_n \geq b_2; \\ ..... \\ a_{m1}y_1 + a_{m2}y_2 + ... + a_{mn}y_n \geq b_m; \end{cases} \qquad (*)$$

с булевыми переменными $y_1, ..., y_n$ и целочисленными a_{ji}, b_j, $j = \overline{1,m}$, $i = \overline{1,n}$.

Существует ли решение этой системы?

Теорема 6.10. $Z_{СЦЛНБ} \in \mathbf{NPC}$.

Доказательство. Полиномиальность проверки того, удовлетворяет ли булевый вектор длины n системе неравенств $(*)$, очевидна.

Покажем, что к проблеме $Z_{\text{СЦЛНБ}}$ полиномиально сводится **NP**-

полная проблема выполнимости КНФ. По произвольной КНФ

$$K(x_1, ..., x_n) = (x_{11}^{\sigma_{11}} \vee \cdots \vee x_{1k_1}^{\sigma_{1k_1}}) \& \ldots \& (x_{m1}^{\sigma_{m1}} \vee \cdots \vee x_{mk_m}^{\sigma_{mk_m}})$$

построим систему вида (*). Переменные x_{ji} в скобках КНФ $K = K(x_1, ..., x_n)$ помечаются двумя символами ji: j – номер дизъюнкта, а i – порядковый номер переменной в дизъюнкте, но все равно обозначение x_{ji} соответствует только одной какой-либо переменной из множества n переменных $x_1, .., x_n$, которые могут быть использованы для записи КНФ K. Поэтому, просматривая символы ji в литералах дизъюнкта, можно указать номера переменных $i_1, ..., i_{k_j}$, входящих в этот дизъюнкт.

Булевы переменные $y_1, ..., y_i, ..., y_n$ конструируемой системы введём следующим образом:

$$y_i = \begin{cases} x_i, & \text{для положительных литералов } (\sigma = 1); \\ 1 - x_i, & \text{для отрицательных литералов } (\sigma = 0). \end{cases}$$

Тогда y_i будет обращаться в единицу тогда и только тогда, когда будет обращаться в единицу литерал $x_i^{\sigma_i}$.

Выберем по записи j-го дизъюнкта КНФ K k_j положительных целых чисел: $a_{j1}, a_{j2}, ..., a_{jk_j} > 0$. Обозначим

$$b_j = \min_{j1 \le s \le jk_j} a_{js}.$$

Применим запись номеров переменных $y_1, ..., y_i, ..., y_n$ соответственно номерам литералов в КНФ K. Очевидно, что

$$(x_{j1}^{\sigma_{j1}} \vee \cdots \vee x_{jk_j}^{\sigma_{jk_1}}) = 1 \Leftrightarrow a_{j1}y_{i_1} + a_{j2}y_{i_2} + ... + a_{ji_{k_j}}y_{i_{k_j}} \ge \min_{j1 \le s \le jk_j} a_{js},$$

Действительно, если в дизъюнкте в левой части эквивалентности не выполнен ни один литерал, то все переменные в сумме $a_{j1}y_{i_1} + a_{j2}y_{i_2} + ... + a_{ji_{k_j}}y_{i_{k_j}}$ равны нулю; сумма равна нулю, и неравенство нарушается: $0 < \min_{j1 \le s \le jk_j} a_{js}$. Если же в дизъюнкте хотя бы один литерал равен единице, то неравенство в правой части эквивалентности выполняется.

Легко проверяется обратное: из выполнения неравенства следует, что дизъюнкт обращается в единицу, иначе – в ноль.

Например, (см. таблицу на рис. 6.38) $(x_1 \vee \bar{x_3} \vee x_5) = 1$ тогда и только тогда, когда $5y_1 + 4(1 - y_3) + 8y_5 \geq 4 \Leftrightarrow 5y_1 - 4y_3 + 8y_5 \geq 0$.

x_1 x_3 x_5 y_1 y_3 y_5	$x_1 \vee \overline{x}_3 \vee x_5$	$5y_1 + 4(1 - y_3) + 8y_5$
0 0 0	1	4
0 0 1	1	12
0 1 0	0	0
0 1 1	1	8
1 0 0	1	9
1 0 1	1	17
1 1 0	1	5
1 1 1	1	13

Рис. 6.38

Построив эквивалентные неравенства для всех дизъюнктов КНФ K, получим систему m линейных целочисленных неравенств. Очевидно, что эта система будет совместна тогда и только тогда, когда КНФ выполнима. Сведение КНФ к системе (*), очевидно, полиномиально по длине этой КНФ, поэтому окончательно получаем: $Z_{\text{ВЫП}} \propto Z_{\text{СЦЛНБ}}$.

6.5 NP-трудные проблемы

Вернёмся к проблеме $Z_{\text{СЦЛНБ}}$ совместности системы целочисленных линейных неравенств с булевыми переменными и преобразуем её в проблему $Z_{\text{ЦЛПБ}}$ – целочисленного линейного программирования с булевыми переменными, дополнив линейной целевой функцией, которую требуется минимизировать:

$$\begin{cases} c_1y_1 + c_2y_2 + ... + c_ny_n \ \to \min; \\ a_{11}y_1 + a_{12}y_2 + ... + a_{1n}y_n \geq b_1; \\ a_{21}y_1 + a_{22}y_2 + ... + a_{2n}y_n \geq b_2; \\ ..... \\ a_{m1}y_1 + a_{m2}y_2 + ... + a_{mn}y_n \geq b_m; \end{cases} \qquad (**)$$

здесь c_i, $i = \overline{1,n}$, также как и остальные коэффициенты и свободные члены, являются целыми числами.

Недетерминированный оператор может угадать допустимое решение оптимизационной задачи $(**)$, но не очевидно, что проверка свойства оптимальности угаданного решения будет полиномиальной.

В то же время проблема совместности

$$\begin{cases} c_1y_1 + c_2y_2 + ... + c_ny_n \ \geq \gamma; \\ a_{11}y_1 + a_{12}y_2 + ... + a_{1n}y_n \geq b_1; \\ a_{21}y_1 + a_{22}y_2 + ... + a_{2n}y_n \geq b_2; \\ ..... \\ a_{m1}y_1 + a_{m2}y_2 + ... + a_{mn}y_n \geq b_m; \end{cases} \qquad (***)$$

где γ-целое число, как уже доказано, является **NP**-полной.

Обозначим l сумму целых отрицательных коэффициентов из множества $c_1, ..., c_n$, а L – сумму целых положительных коэффициентов этого множества. Тогда

$$l \leq c_1y_1 + c_2y_2 + ... + c_ny_n \leq L.$$

Учтём, что $|l| + L + 1 = \mathscr{M}$ – константа, не зависящая от n.

Решая систему $(***)$ последовательно для значений

$$\gamma = -l, (-l+1), ..., -1, 0, 1, 2, ..., (L-1), L \qquad (47)$$

не более чем $|l| + L + 1 = \mathscr{M}$ раз, можно получить решение оптимизационной проблемы $(**)$ за $1 \leq \mu \leq \mathscr{M}$ шагов, если до шага $\mu - 1$ включительно ответ задачи о совместности системы $(***)$ будет "НЕТ" (отрицательным полагаем также ответ на нулевом шаге при $\mu - 1 = 1$), а затем станет "ДА". И тогда значение $\hat{\gamma}$, являющееся μ-ым элементом последовательности значений (47), будет являться минимумом для задачи $(**)$ *и в то же время минимальным значением* γ*, при котором совместна система* $(***)$.

Если для оптимизационной проблемы

$$\begin{cases} c_1y_1 + c_2y_2 + ... + c_ny_n \ \to \min; \\ a_{11}y_1 + a_{12}y_2 + ... + a_{1n}y_n \geq b_1; \\ a_{21}y_1 + a_{22}y_2 + ... + a_{2n}y_n \geq b_2; \\ ..... \\ a_{m1}y_1 + a_{m2}y_2 + ... + a_{mn}y_n \geq b_m; \end{cases}$$

ввести свойство: "*Минимальное значение целевой функции равно* $\hat{\gamma}$, то ответ "ДА" влечёт выполнение системы

$$\begin{cases} c_1y_1 + c_2y_2 + ... + c_ny_n \ \geq \hat{\gamma}; \\ a_{11}y_1 + a_{12}y_2 + ... + a_{1n}y_n \geq b_1; \\ a_{21}y_1 + a_{22}y_2 + ... + a_{2n}y_n \geq b_2; \\ ..... \\ a_{m1}y_1 + a_{m2}y_2 + ... + a_{mn}y_n \geq b_m; \end{cases}$$

Обратно, если эта система выполнима, то минимум соответствующей оптимизационной задачи равен $\hat{\gamma}$.

Таким образом, к оптимизационной проблеме $Z_{\text{ЦЛПБ}}$ сводится **NP**-полная проблема $Z_{\text{СЦЛНБ}}$ совместности линейных целочисленных неравенств с булевыми переменными (и поэтому сводится любая проблема из **NP**). Но в то же время проблема $Z_{\text{ЦЛПБ}}$, возможно, не входит в класс **NP**, так как не удаётся показать, что проверка экстремальности решения является полиномиальной.

Рассмотрение проблемы $Z_{\text{ЦЛПБ}}$ поясняет введение следующего понятия.

Определение 6.9. *Классом* **NPH** *недетерминированных полиномиально трудных проблем* (**N***ondeterministic* **P***olynomial* **H***ard*) *называется совокупность всех таких проблем распознавания свойств, к каждой из которых полиномиально сводится любая проблема из* **NP**.

Проблемы класса **NPH** называются **NP**-трудными, и каждая из них является не менее сложной, чем любая **NP**-полная проблема.

Из определения 6.9 следует, что проблема является **NP**−полной, если она **NP**−трудная и входит в класс **NP**:

$$Z \in \mathbf{NPH} \ \wedge \ Z \in \mathbf{NP} \ \Rightarrow Z \in \mathbf{NPC}.$$

Любая **NP**-полная проблема является **NP**-трудной. Обратное в общем случае неверно, поскольку **NP**-трудная проблема может иметь распознаваемое свойство, не вычислимое за полиномиальное число шагов (см. рис. 6.39).

Серьёзным основанием полагать, что $\mathbf{P} \neq \mathbf{NP}$ *является то, что ни для одной из открытых математиками* **NP***-трудных про-*

блем до сих пор не найден полиномиальный алгоритм решения.

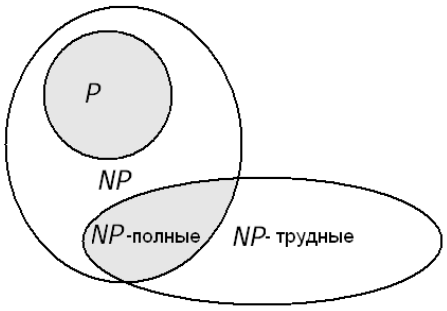


Рис. 6.39: Соотношение между классами P, NP, NPC и NPH

Если бы хотя бы для одной **NP**−*трудной проблемы удалось найти полиномиальный алгоритм её решения, то это стало бы значительным математическим открытием и позволило бы "расправиться" за полиномиальное число шагов с огромным количеством труднорешаемых задач, имеющих многочисленные применения в информатике, экономике, технике.*

Приведём ещё ряд полезных для понимания природы вычислительной сложности и дальнейших исследований примеров **NP**−полных проблем (список таких проблем постоянно растёт по мере изучения новых задач)[19]. Многие из этих проблем в экстремальной постановке *не будут обладать возможностью полиномиальной проверки свойства*, и тогда их следует называть **NP**−трудными, чтобы подчеркнут отмеченный факт.

Раскрашиваемость графа *(хроматическое число).*

Даны: Конечный граф $\Gamma =< X, U >$ и натуральное число $k \leq |X|$.

Свойство: является ли граф Γ $k-$раскрашиваемым, т.е. существует ли такая функция (раскраски в k цветов) $f : X \to \{1, 2, ..., k\}$, что если пара вершин x и y является концами одного ребра $(x, y) \in U$, то $f(x) \neq f(y)$.

Разбиение на лесá.

Даны: Конечный граф $\Gamma =< X, U >$ и натуральное число $k \leq |X|$.

Свойство: Можно ли разбить вершины графа Γ на K таких непересекающихся подмножеств $X_1, ..., X_i, ..., X_k$, что для всех $i = 1, 2, ..., k$ подграф Γ_i графа Γ, состоящий из вершин множества X_i и всех инцидентных этим вершинам рёбер из множества рёбер U графа Γ, не содержит циклов.

[19] В монографии [5] приведен обширный список труднорешаемых задач.

Независимое множество графа.

Даны: Конечный граф $\Gamma =< X, U >$ и натуральное $k \leq |X|$.

Свойство: Содержит ли граф Γ независимое множество мощности не менее k, т. е. такое подмножество из не менее k вершин, что в нём никакие две вершины не соединены ребром.

Связный подграф ограниченной степени.

Даны: Конечный граф $\Gamma =< X, U >$, натуральное число $d \leq |X|$ и натуральное $k \leq |U|$.

Свойство: Существует ли такое подмножество рёбер $U^{'} \in U$, что подграф $\Gamma =< X, U^{'} >$ связен и не имеет вершин степени более d.

Планарный подграф.

Даны: Конечный граф $\Gamma =< X, U >$ и натуральное число $k \leq |X|$.

Свойство: Существует ли подмножество рёбер $U^{'} \in U$ такое, что $|U^{'}| \geq k$ и граф $\Gamma =< X, U^{'} >$ планарен [20].

Гамильтонов цикл.

Дан: Конечный граф $\Gamma =< X, U >$.

Свойство: Существует ли в графе Γ гамильтонов цикл – последовательность ребер, образующая цикл, проходящий ровно один раз через каждую вершину этого графа.

Коммивояжер.

Даны: Конечный граф $\Gamma =< X, U >$, функция $f : U \to \mathbb{Z}^{+}$, определяющая цену ребра, и натуральное число $K \leq |X|$.

Свойство: Существует ли цикл, проходящий через каждую вершину множества X ровно один раз (гамильтонов), такой, что суммарная цена входящих в него рёбер не превышает K.

Изоморфизм подграфу.

Даны: Два графа $\Gamma_1 =< X_1, U_1 >$ и $\Gamma_2 =< X_2, U_2 >$.

Свойство: Содержит ли граф Γ_1 подграф, изоморфный графу Γ_2.

Минимальное покрытие.

Даны: Набор подмножеств $M_1, ..., M_s$ конечного множества $\mathfrak{M}$, $M_1 \cup M_2 \cup ... M_s = \mathfrak{M}$ и натуральное $k \leq s$.

Свойство: Существует ли покрытие множества $\mathfrak{M}$ не более чем

[20]Планарным называют граф, для которого можно указать изоморфный граф на плоскости такой, что линии, соответствующие различным рёбрам не пересекаются

k подмножествами.

Разбиение.

Дано: множество $A = a_1, a_2, ...a_m$ объектов, для каждого из которых задан целый положительный вес $s(a_i),\ i = 1, 2, ..., m$.

Свойство: Существует ли разбиение множества $A = A_1 \cup A_2$, $A_1 \cap A_2 = \emptyset$, на два подмножества такое, что

$$\sum_{i \in A_1} s(a_i) = \sum_{i \in A_2} s(a_i).$$

3-разбиение.

Даны: Множество A, состоящее из $3m$ элементов, положительное целое B, положительные целые "размеры" $s(a)$ всех элементов $a \in A$, причём $B/4 < s(a) < B/2$ и $\sum_{a \in A} s(a) = mB$.

Свойство: Существует ли разбиение множества A на m непересекающихся подмножеств $A_1, A_2, ..., A_m$ такое, что $\sum_{a \in A_i} s(a) = B$ для всех $i = 1, 2, ..., m$.

Подматрица со свойством связности.

Даны: Матрица $m \times n$ с элементами 0 и 1 и натуральное число $k < n$.

Свойство: Существует ли в матрице A подматрица B размера $m \times k$, обладающая свойством связности единиц, т.е. такая подматрица B, столбцы которой можно переставить так, что в каждой строке все единицы будут идти подряд.

Редактирование слова.

Даны: Конечный алфавит $\mathscr{A}$, два слова x и y, состоящие из букв этого алфавита, и натуральное число k.

Свойство: Можно ли получить из слова x слово y, применив не более k операций вычёркивания символа и перестановки соседних символов.

Многопроцессорное расписание.

Даны: Конечное множество заданий $\mathscr{T}$, число процессоров m, длительность выполнения задания $l(\tau) \in \mathbb{Z}^+$ для каждого $\tau \in \mathscr{T}$ и общий директивный срок выполнения заданий $D \in \mathbb{Z}^+$.

Свойство: Существует ли $m-$процессорное расписание для заданий $\mathscr{T}$, которое удовлетворяет директивному сроку D (существует ли возможность назначения каждой работе процессора и времени её начала $t(\tau)$ так, чтобы выполнялись условия:

а) для всех заданий $t(\tau) + l(\tau) \leq D$ и

б) в каждый момент времени $t \in 1, 2, ..., D$ число загруженных процессоров не превышало m).

Целочисленное программирование.

Существует ли целочисленное решение задачи $AX \leq B$ такое, что $\sum_{i=1}^{n} c_i x_i \geq \gamma$, где A – целочисленная матрица $m \times n$, B – целочисленный вектор столбец, c_i – целочисленные коэффициенты, x_i неизвестные целочисленные компоненты вектор-столбца X, $i = 1, ..., n$.

Рюкзак.

Даны: Конечное множество (предметов) U, вес $w(u)$, стоимость $c(U)$ каждого $u \in U$, положительные целые числа W и C.

Свойство: Существует ли такое подмножество предметов $U' \in U$, что

$$\sum_{u \in U'} w(u) \leq W \text{ и } \sum_{u \in U'} \geq C.$$

Частный случай диофантовых уравнений.

Существуют ли положительные целые корни уравнения $ax^2 + by + c$, где a, b, c – положительные целые числа.

Алгебраические уравнения в двухэлементном поле GF_2.

Существует ли решение системы уравнений

$$\mathscr{P}_i(x_1, ..., x_n) = 0, \quad i = 1, ..., m,$$

где $\mathscr{P}_i(\cdot)$ – полиномы над полем GF_2 (т.е. такие, что в них любой моном является либо единицей, либо произведением переменных).

Минимальная ДНФ частично заданной булевой функции.

Даны: m $n-$координатных булевых наборов $\tilde{\alpha}_1, ..., \tilde{\alpha}_j, ..., \tilde{\alpha}_m$ и натуральное k.

Свойство: существует ли ДНФ над множеством булевых переменных $x_1, ..., x_n$, содержащая не более k конъюнкций, истинная на наборах $\tilde{\alpha}_j$, $j = 1, ..., m$.

Булевы формулы с кванторами.

Даны: Булевы переменные $x_1, ..., x_n$ и правильно построенная

булева формула $\mathscr{F}$ с кванторами

$$\mathscr{F} = (Q_1 x_1)(Q_2 x_2)...(Q_n x_n)\mathscr{E},$$

где $\mathscr{E}$ – булева формула без кванторов, а Q есть либо $\forall$, либо $\exists$.

Свойство: Является ли формула $\mathscr{F}$ истинной.

Минимальный разделяющий конечный автомат.

Даны: Конечный алфавит, два конечных множества слов W_1 и W_2 над этим алфавитом и натуральное число K.

Свойство: Существует ли детерминированный конечный автомат с k состояниями такой, что на входных словах множества W_1 он выдаёт единицу, а на входных словах множества W_2 – ноль.

Минимальное дерево решений.

Даны: Конечное множество объектов X, набор бинарных тестов $\mathscr{T} = \{T_1, ..., T_i, ..., T_m\}$, $T_i : X \to \{0, 1\}$, и натуральное k.

Свойство: Существует ли дерево решений, в котором каждая концевая вершина помечена одним из объектов множества X (классифицирующее дерево), а в каждой внутренней вершине проверяется один из тестов множества $\mathscr{T}$, такое, что сумма числа рёбер от листа то корневой вершины дерева, взятая по всем листьям (длина внешнего пути), не превосходит числа K [21].

6.6 Полиномиальная сводимость и класс co-NP в терминах языков

Языком $\mathscr{L}$ над алфавитом $\mathscr{A}$ называется любое подмножество слов $\mathscr{L} \subset S(\mathscr{A})$. Обычно рассматривают языки, имеющие некоторый содержательный смысл.

Задачей распознавания для языка $\mathscr{L}$ называется выяснение того, принадлежит ли произвольное заданное слово $x \in S(\mathscr{A})$ языку $\mathscr{L}$. Таким образом, задача распознавания для языка $\mathscr{L}$ состоит в вычислении свойства "$x \in \mathscr{L}$" для любого слова $x \in S(\mathscr{A})$. Решающий такую задачу распознавания алгоритм называется распознающим.

Рассмотрим некоторую проблему Z распознавания свойства и соотнесём каждую индивидуальную задачу z этой проблемы с пред-

[21] Или число листьев дерева μ не превосходит некоторого заданного числа M

ставляющим её описанием – входным словом x_z из множества всех допустимых слов $S(\mathscr{X})$ над алфавитом $\mathscr{X}$. Тогда множество всех слов x_z, соответствующих индивидуальным задачам, для которых свойство выполняется, можно определить как язык $\mathscr{L}_Z \subset S(\mathscr{X})$. Таким образом, распознавание свойства для задач проблемы Z эквивалентно распознаванию принадлежности описаний задач языку $\mathscr{L}_Z$. Поэтому следующие определения класса **P**, полиномиальной сводимости, **NP** и некоторые другие будут эквивалентными определениям этих понятий, данным ранее.

Определение 6.10. *Классом* **P** *называется семейство всех языков, для каждого из которых существует распознающий алгоритм, имеющий полиномиальную временную сложность.*

Определение 6.11. *Язык $\mathscr{L}_1$ над алфавитом $\mathscr{A}_1$ называется полиномиально сводимым к языку $\mathscr{L}_2$ над алфавитом $\mathscr{A}_2$, если существует вычислимая с полиномиальной временной сложностью функция $f : S(\mathscr{A}_1) \to S(\mathscr{A}_2)$ такая, что любое слово x принадлежит языку $\mathscr{L}_1$ тогда и только тогда, когда $f(x) \in \mathscr{L}_2$.*

Данную в определении сводимость называют сводимостью по Карпу и обозначают $\mathscr{L}_1 \propto \mathscr{L}_2$ (или $\mathscr{L}_1 \leq_p \mathscr{L}_2$).

Определение 6.12. *Классом* **NP** *называется семейство всех таких языков $\mathscr{L} \subset S(\mathscr{A})$, для которых существует дополнительный алфавит $\mathscr{B}$ и предикат (свойство) $Q : S(\mathscr{A}) \times S(\mathscr{B}) \to \{0, 1\}$, вычислимый с полиномиальной временной сложностью, такой, что слово x принадлежит языку $\mathscr{L}$ тогда и только тогда, когда найдется слово $y \in S(\mathscr{B})$, длина которого ограничена полиномом от длины слова $\mathscr{L}$, и $x \in \mathscr{L} \Leftrightarrow Q(x, y) = 1$.*

Согласно этому определению, распознать принадлежность слова языку из класса **NP** можно путём вычисления некоторого предиката $Q = Q(x, y)$ с полиномиальной сложностью, но только если будет предоставлено некоторое (неизвестное) слово $y \in S(\mathscr{B})$, имеющее полиномиальную длину. Такое слово y называют сертификатом для слова x, а предикат $Q(x, y)$ – свойством для языка $\mathscr{L}$. Поэтому класс **NP** называют классом языков с полиномиальным распознаванием или вычислением свойства.

Любой язык $\mathscr{L}$ по определению является зафиксированным подмножеством слов над некоторым алфавитом $\mathscr{A}$: $\mathscr{L} \subset S(\mathscr{A})$. Все остальные слова составляют *дополнение языка*, которое обозначается $\overline{\mathscr{L}} = S(\mathscr{A}) \setminus \mathscr{L}$.

Определение 6.13. *Пусть $\mathfrak{L}$ - произвольный класс языков. Будем называть сопряжённым классом для класса $\mathfrak{L}$ класс* **co-**$\mathfrak{L} = \{L : \overline{L} \in \mathfrak{L}\}$.

Теорема 6.11. $\mathbf{P} = \mathbf{co\text{-}P}$.

Доказательство. Для любого языка $L \in \mathbf{P}$ существует полиномиальный алгоритм, вычисляющий свойство "$x \in L$". Если $x \in L$, этот алгоритм даёт ответ 1 ("ДА"), а если $x \notin L$, то этот алгоритм даёт ответ 0 ("НЕТ"). Но $x \notin L \Leftrightarrow x \in \bar{L}$. Поэтому ответ 0 алгоритма распознавания определяет принадлежность слова x языку $\bar{L} \in \mathbf{co\text{-}P}$. Преобразуем алгоритм распознавания так, чтобы его ответ был инверсным: 1 – когда $x \notin L$ и 0 – когда $x \in L$. При этом алгоритм останется полиномиальным и будет распознавать свойство $x \notin L$. Поэтому $\mathbf{P} = \mathbf{co\text{-}P}$, и говорят, что класс $\mathbf{P}$ замкнут относительно своего дополнения. □

Согласно определению 6.12, если $L \in \mathbf{NP}$, то в задаче распознавания свойства "$x \in L$" ответ "ДА" имеет место только при существовании слова $y \in S(\mathscr{B})$ такого, что $Q(x, y) = 1$.

Если $\bar{L} \in \mathbf{co\text{-}NP}$, то в задаче распознавания свойства "$x \in \bar{L}$" ответ "ДА" имеет место, когда $\overline{\exists y(Q(x, y) = 1)}$, или, эквивалентно, когда для всех y выполняется $Q(x, y) = 0$.

Заметим, что сложность проверки свойства по угаданному слову-сертификату y равенства $Q(x, y) = 1$ и сложность установления факта, что для любого слова y имеет место $Q(x, y) = 0$, вообще говоря, различна. Тем более, что, выбор слова y вообще недетерминирован. Поэтому рассуждения, представленные при доказательстве равенства $\mathbf{P} = \mathbf{co\text{-}P}$, не подходят для выяснения вопроса о совпадении классов $\mathbf{NP}$ и $\mathbf{co\text{-}NP} = \{L : \bar{L} \in \mathbf{NP}\}$.

Очевидно, что $\mathbf{co\text{-}P} \subset \mathbf{co\text{-}NP}$ также как $\mathbf{P} \subset \mathbf{NP}$, и пересечение $\mathbf{NP} \cap \mathbf{co\text{-}NP}$ непусто, поскольку содержит класс $\mathbf{P}$. Вопрос о равенстве этого пересечения классу $\mathbf{P}$ остаётся открытым:

$$\mathbf{P} \stackrel{?}{=} \mathbf{NP} \cap \mathbf{co\text{-}NP}.$$

Равенство **NP** = **co-NP** скорее всего не имеет места, поскольку неизвестно: можно ли эффективно доказать отсутствие подходящего сертификата y при проверке свойства. Ведь в случае, если бы это равенство имело место, любая проблема из **co-NP** должна была бы иметь полиномиальную проверку свойства.

Определение 6.14. *Классом* **co-NPC** *полных в* **co-NP** *языков называется множество таких языков* L*, что*

1) $L \in$ **co-NP** *и*

2) $\forall \mathcal{L} \in$ **co-NP** $(\mathcal{L} \propto L)$.

Можно предположить такое соотношение между классами **P**, **NP**, **NPC**, **co-NP** и **co-NP**, как показано на рис 6.40.

Теорема 6.12. *Если допустить* **P** = **NP**, *то* **NP** = **co-NP**.

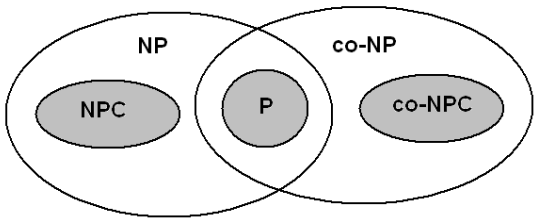


Рис. 6.40: Предполагаемое соотношение между классами P, NP, NPC, co-NP и co-NPC

Доказательство. Возьмём любой язык $L \in$ **co-NP**. Тогда $\bar{L} \in$ **NP**. Согласно предположению **P** = **NP**, имеем $\bar{L} \in$ **P**. Поскольку класс **P** замкнут относительно дополнения, то и $L \in$ **P**. Поскольку в качестве L взят любой язык из **co-NP**, заключаем, что любой язык L из **co-NP** входит в **P**: **co-NP** $\subset$ **P**. Учитывая, что и **P** $\subset$ **co-NP**, получаем **P** = **co-NP**. Наконец, учитывая, что по предположению в условии теоремы **P** = **NP**, получаем **NP** = **co-NP**. $\square$

Таким образом, если предположить, что имеет место маловероятное равенство **P** = **NP**, то класс **NP** окажется замкнутым относительно дополнения.

Равносильным утверждению теоремы 6.12 является следующее утверждение: если **NP** $\neq$ **co-NP**, то **P** $\neq$ **NP**. Поэтому предположение о неравенстве классов **NP** и **co-NP** представляется более правдоподобным, чем их равенство. Если же исследуемая вычислительная проблема содержится как в классе **NP**, так и в классе **co-NP**, то есть основания считать возможным, что эта проблема разрешима полиномиально – принадлежит классу **P** (см. рис. 6.40), и попытаться найти полиномиальный алгоритм её решения.

6.7 Псевдополиномиальная сводимость и сильная NP-полнота

Ранее, при изучении вычислительной сложности проблем, ограничения на величины чисел, входящих в индивидуальные задачи, в явном виде не накладывались[22]. Отправной точкой являлась длина входа задач. В этом параграфе речь пойдет о проблемах распознавания свойств с условиями на максимальные значения чисел в индивидуальных задачах.

По-прежнему, будем обозначать Z – вычислительную проблему распознавания свойства, $z \in Z$ – произвольную индивидуальную задачу этой проблемы, $x = x_z$ – входное слово-описание задачи, $len(x) = n$ – размер входа (длина, число символов при при некотором разумном кодировании задач проблемы Z).

Выделенную часть Z_* проблемы Z, $Z_* \subset Z$, будем называть *подпроблемой* проблемы Z.

Поскольку размер входа, вообще говоря, может быть разным для различных индивидуальных задач, будем обозначать $L(z) = len(x_z)$ функцию, ставящую в соответствие индивидуальной задаче длину её входа: $L : Z \to \mathbb{Z}^+$.

Введём ещё одну функцию $M : Z \to \mathbb{Z}^+$, *ставящую в соответствие любой индивидуальной задаче $z \in Z$ максимальное число, фигурирующее в её задании.* Тогда $M(z)$ есть самое большое число, которое имеется в формулировке индивидуальной задачи z.

Например, в задаче коммивояжера значением $M(z)$ может быть максимальный вес из всех заданных весов рёбер.

Будем говорить, что *две пары функций (L, M) и (L', M') полиномиально эквивалентны* для проблемы Z, если существуют полиномы $\mathscr{P}_1(\cdot), \mathscr{P}_2(\cdot), \mathscr{P}_3(.,.), \mathscr{P}_4(.,.)$ такие, что для всех индивидуальных задач $z \in Z$

$$L(z) \le \mathscr{P}_1(L'(z)),$$
$$L'(z) \le \mathscr{P}_2(L(z),$$
$$M(z) \le \mathscr{P}_3(M'(z), L'(z)),$$

[22]Полагалось, что при "разумном" кодировании целое положительное число M, входящее в описание x_z – вход индивидуальной задачи z, занимает $\lceil \log M \rceil$ символов в слове x

$$M^{'}(z) \leq \mathscr{P}_4(M(z), L(z)).$$

Рассматривая далее функции M и L с точностью до введенной полиномиальной эквивалентности, будем только требовать, чтобы они были вычислимыми, причём за полиномиальное время.

Определение 6.15. *Алгоритм решения проблемы Z называется псевдополиномиальным по времени (или просто псевдополиномиальным), если его временна́я сложность при решении любой задачи $z \in Z$ ограничена сверху полиномом от двух аргументов: $L(z)$ и $M(z)$.*

Из определения 6.15 следует, что любой полиномиальный алгоритм является псевдополиномиальным, но обратное, вообще говоря, не верно. Чтобы в этом убедиться, достаточно указать хотя-бы одну проблему, разрешимую псевдополиномиально, но неразрешимую полиномиально.

Рассмотрим $\mathbf{NP}-$полную проблему РАЗБИЕНИЕ – $Z_{\text{разб}}$, состоящую в следующем.

Дано множество $A = a_1, a_2, ...a_m$ объектов, для каждого из которых задан целый положительный вес $s(a_i),\ i = 1, 2, ..., m$.

Свойство: существует ли разбиение множества $A = A_1 \cup A_2,\ A_1 \cap A_2 = \emptyset$, на два подмножества такое, что

$$\sum_{i \in A_1} s(a_i) = \sum_{i \in A_2} s(a_i).$$

Если предположить, что $\mathbf{P} \neq \mathbf{NP}$, то полиномиального алгоритма решения $Z_{\text{разб}}$ существовать не может, поскольку $Z_{\text{разб}}$ является $\mathbf{NP}-$полной.

Покажем, что для $Z_{\text{разб}}$ существует псевдополиномиальный алгоритм решения.

Обозначим $B = \sum_{i \in A} s(a_i)$. Тогда для того, чтобы свойство равенства сумм выполнялось, B должно быть чётным и тогда для искомого разбиения

$$\sum_{i \in A_1} s(a_i) = \sum_{i \in A_2} s(a_i) = B/2.$$

Пусть $T[i, j]$, где $i = 1, 2, ..., m,\ j = 0, 1, ..., B/2$, будет обо-

значать значение истинности утверждения: "существует такое *подмножество* множества $\{a_1, a_2, ...a_i\} \subseteq A$, что $\sum_{k=1}^{i} s(a_k) = j$ ".

Определённые таким образом значения $T[i, j]$ можно представить в виде таблицы $T_{m\times(B/2+1)}$ вида

$i \setminus j$	0	1	...	j	j+1	...	B/2
1	1	σ	σ	σ	σ	σ	σ
2	1	σ	σ	σ	σ	σ	σ
...	1	σ	σ	σ	σ	σ	σ
i	1	σ	σ	σ	σ	σ	σ
i+1	1	σ	σ	σ	σ	σ	σ
...	1	σ	σ	σ	σ	σ	σ
m	1	σ	σ	σ	σ	σ	σ

Значение "ИСТИНА" в таблице $T_{n\times(B/2+1)}$ будем обозначать единицей, а "ЛОЖЬ"– нулём, а σ обозначает одно из двух возможных значений – 0 или 1.

В нулевом столбце заведомо будут все единицы, поскольку пустое множество будет иметь суммарный вес 0 и входить в качестве подмножества в любое множество.

В первой строке $T[1, j] = 1$ только тогда, когда $j = 0$ или $j = s(a_1)$. А каждая последующая строчка таблицы заполняется с использованием уже полученных значений, расположенных в предыдущей строчке. А именно: для $i:\ 1 < i \leq m$ и $j : 1 \leq j \leq B/2$ равенство $T[i, j] = 1$ будет иметь место только в двух случаях:

1) когда $T[i - 1, j] = 1$ или

2) $\Big(s(a_i) \leq j\Big) \bigwedge \Big(T[i - 1, j - s(a_i)] = 1\Big)$.

Рассмотрим алгоритм решения $Z_{\text{разб}}$, состоящий в последовательном построчном слева-направо заполнении таблицы $T_{m\times(B/2+1)}$. Заметим, что как только будет получено $T[i, B/2] = 1$, задача будет решена с ответом "ДА", поскольку это будет означать существование в A некоторого подмножества A_1 веса $B/2$, и его дополнение в A будет иметь тот же вес. Иначе, если $T[i, B/2] = 0$ для всех строк $i = 1, 2, ..., m$, – ответ будет "НЕТ".

Время решения таким алгоритмом, очевидно, ограничено полиномом от числа клеток таблицы $T_{m\times(B/2+1)}$ и может быть оценено как $O(m \cdot B)$.

Положив функции $M(z) = B$ и $L(z) = n$, видим, что рассматриваемый алгоритм является псевдополиномиальным.

Но полиномиальным для $Z_{\text{разб}}$ этот табличный алгоритм, конечно, не является, поскольку кодирование начальной информации о задаче z в слово $x = x_z$ предполагается "разумным": для каждого числа его вес $s(a_i)$ представляется $O(\log s(a_i))$-разрядным кодом. Поэтому длина входа задачи $z \in Z_{\text{разб}}$ составит $len(x) = O(m \log B)$, и тогда mB не ограничено никаким полиномом от этой длины.

Наличие указанного табличного псевдополиномиального алгоритма для проблемы РАЗБИЕНИЕ показывает, что сложность её решения в значительной степени зависит от величин чисел, входящих в описание индивидуальных задач. Если наложить ограничения полиномиальности на эти числа, – чтобы они не превышали полинома от размера входа задачи, то рассмотренный псевдополиномиальный алгоритм стал бы полиномиальным.

Мы убедились в том, что **NP**−полнота проблемы не обязательно исключает возможность её решения псевдополиномиальным алгоритмом.

Определение 6.16. *Проблема Z называется проблемой с числовыми параметрами, если не существует полинома $\mathscr{P}(\cdot)$, оценивающего для всех индивидуальных задач $z \in Z$ сверху функцию $M(z)$ неравенством*

$$M(z) \leq \mathscr{P}(L(z)). \tag{48}$$

Неравенство (48) соответствует условию, когда наибольший числовой параметр задачи ограничен некоторым полиномом от размера её входа, но для задач с параметрами такого полинома не должно существовать.

Можно сказать, что проблемы с числовыми параметрами имеют числовые значения, оказывающие существенное влияние на временну́ю сложность этих проблем. А их дополнение в классе проблем распознавания свойств (проблемы без числовых параметров) не содержат существенно влияющих на временну́ю сложность параметров.

Легко убедиться в том, что *если какая-нибудь* **NP**−*полная проблема не является проблемой с числовыми параметрами, то она не может быть решена никаким псевдополиномиальным алгорит-*

мом[23].

Поэтому, если предположить, что $\mathbf{P} \neq \mathbf{NP}$, то потенциальная возможность решения псевдополиномиальным алгоритмом имеется только для тех $\mathbf{NP}-$полных проблем, которые являются проблемами с числовыми параметрами.

Определение 6.17. *Проблема распознавания свойства Z называется $\mathbf{NP}-$полной в сильном смысле, если одновременно выполняются следующие условия:*

1) $Z \in \mathbf{NP}$;

2) Z содержит в себе подпроблему $Z_{\mathscr{P}} \subseteq Z$ такую[24]*, что для всех индивидуальных задач $z \in Z_{\mathscr{P}}$ существует полином $\mathscr{P}$ с целыми неотрицательными коэффициентами и*

$$M(z) \leq \mathscr{P}(L(z));$$

3) $Z_{\mathscr{P}} \in \mathbf{NPC}$.

Согласно определению 6.17, любая $\mathbf{NP}-$полная в сильном смысле проблема лежит в классе $\mathbf{NP}$ и содержит в себе $\mathbf{NP}-$полную подпроблему, не являющуюся проблемой с числовыми параметрами.

Класс всех $\mathbf{NP}-$полных в сильном смысле проблем обозначают $\mathbf{NPCS}$.

Заметим, что в классе $\mathbf{NPC}$ существуют как проблемы с числовыми параметрами (например, РАЗБИЕНИЕ), так и проблемы не являющиеся проблемами с числовыми параметрами (например, ВЫПОЛНИМОСТЬ КНФ). Любая $\mathbf{NP}-$полная проблема Z, совпадающая со своим полиномиальным сужением $Z_{\mathscr{P}}$, очевидно, является $\mathbf{NP}-$полной в сильном смысле. Поэтому в класс $\mathbf{NPCS}$, кроме ВЫПОЛНИМОСТИ КНФ, входят проблемы $k-$ПОЛНЫЙ подграф, ГАМИЛЬТОНОВ ЦИКЛ, ВЕРШИННОЕ ПОКРЫТИЕ ГРАФА и др.

Теорема 6.13. *Если $\mathbf{P} \neq \mathbf{NP}$ и проблема Z является $\mathbf{NP}-$полной в сильном смысле, то она не может быть решена никаким псевдополиномиальным алгоритмом.*

---

[23]Ведь в таком случае псевдополиномиальный алгоритм будет являться просто полиномиальным

[24]Подпроблему подпроблему $Z_{\mathscr{P}}$ часто называют полиномиальным сужением проблемы Z

Доказательство. Действительно, если найдётся псевдополиномиальный алгоритм $\mathfrak{A}$ решения проблемы Z, то он будет приме́нйм ко всем её индивидуальным задачам, включая подмножество $Z_{\mathscr{P}}$, не являющееся проблемой с числовыми параметрами. Для $Z_{\mathscr{P}}$ алгоритм $\mathfrak{A}$ будет просто полиномиальным. Но $Z_{\mathscr{P}} \in \mathbf{NPC}$, и тогда оказывается, что проблема $Z_{\mathscr{P}}$ разрешима полиномиальным алгоритмом $\mathfrak{A}$, откуда вытекает, что хотя бы одна $\mathbf{NP}-$полная проблема решена полиномиальным алгоритмом, что влечёт равенство $\mathbf{P} = \mathbf{NP}$, а это приводит к противоречию с условием $\mathbf{P} \neq \mathbf{NP}$ в формулировке теоремы. $\square$

Основываясь на теореме 6.13, можно сделать вывод, что, например, проблема РАЗБИЕНИЕ, если $\mathbf{P} \neq \mathbf{NP}$, не является $\mathbf{NP}-$полной в сильном смысле, т.к. её можно решить псевдополиномиальным алгоритмом.

Сделаем вывод о том, что проблема РАЗБИЕНИЕ не является $\mathbf{NP}-$полной в сильном смысле, ещё более очевидным.

Согласно определению 6.17, если

$$(Z \notin \mathbf{NP}) \vee (\bar{\exists} Z_{\mathscr{P}} : \forall z \in Z_{\mathscr{P}}\ (M(z) \leq \mathscr{P}(L(z)))) \vee (Z \notin \mathbf{NPC}),$$

то $Z \notin \mathbf{NPCS}$.

Пусть $M(z) = \mu$ – максимальный из весов $s(a_i)$. Проблема РАЗБИЕНИЕ заведомо содержится в классе $\mathbf{NP}$.

a) Предположим, что существует полином $\mathscr{P}(L(z))$ такой, что $\forall z \in Z_{\mathscr{P}}\ (M(z) \leq \mathscr{P}(L(z)))$. Но полусумма весов составляет $\frac{1}{2}\sum_i s(a_i) = B/2 \leq m\mu$. Проблему $Z_{\mathscr{P}}$ можно решить за $mB = m \times m\mu$ шагов описанным выше полиномиальным алгоритмом решения проблемы РАЗБИЕНИЕ. Но $m^2\mu = (m \log \mu)^k = \mathscr{P}(L(z))$ при значении

$$k > 2 + \log \sum_{i=1}^{m} s(a_i) > \frac{2 \log m + \log \mu}{\log m + \log \log \mu},$$

поскольку $m \log \mu = L(z)$ – длина входа. Тогда $Z_{\mathscr{P}}$ разрешима полиномиально; следовательно, $Z_{\mathscr{P}} \notin \mathbf{NPC}$ и $Z \notin \mathbf{NPCS}$.

b) Если же не существует полинома $\mathscr{P}(L(z))$ такого, что $\forall z \in Z_{\mathscr{P}}\ (M(z) \leq \mathscr{P}(L(z)))$, то снова $Z \notin \mathbf{NPCS}$. $\square$

Чтобы доказать сильную **NP**$-$*полноту некоторой проблемы* Z *с числовыми параметрами*, достаточно убедиться в том, что для некоторого конкретного полинома $\mathscr{P}$ найдётся полиномиальное сужение, являющееся **NP**$-$полным, т.е подпроблема

$$Z_{\mathscr{P}}: \ (Z_{\mathscr{P}} \subset Z) \ \wedge (Z_{\mathscr{P}} \subset \mathbf{NPC}).$$

Рассмотрим проблему КОММИВОЯЖЕР $Z_{\text{комм}}$.

Очевидно, что, являясь **NP**$-$полной, $Z_{\text{комм}} \in \mathbf{NP}$. Но в формулировке $Z_{\text{комм}}$ полиномиальных ограничений на числовые параметры – расстояния между городами и границы K для длины пути гамильтонова цикла – нет.

Укажем полиномиальное сужение $Z_{\text{комм}}$ – такую её подпроблему $Z_{\mathscr{P}}^{\text{комм}} \subset Z_{\text{комм}}$, что для любой индивидуальной задачи $z \in Z_{\mathscr{P}}^{\text{комм}}$ её максимальный числовой параметр $M(z)$ будет ограничен длиной входа $L(z)$. Для этого выделим только такие индивидуальные задачи, в которых расстояния между городами равны 1 или 2, а граница K равна числу городов m. Тогда вход z можно закодировать словом, содержащим код числа $K = m$ и $m(m-1)/2$ чисел 1 или 2 (расстояний между городами), которое будет иметь длину $L(z) = \lceil log_2 m \rceil + m(m-1)/2 + c$, где c – положительная целая константа, учитывающая разделители между блоками информации.

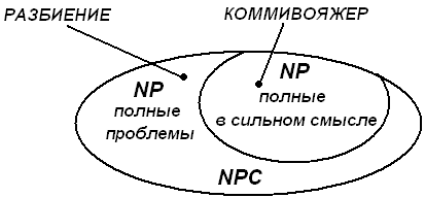


Рис. 6.41: В классе NP-полных проблем содержатся ещё более сложные – NP-полные в сильном смысле

Если взять в качестве $M(z)$ максимум из пары: граница $K = m$ и наибольшее расстояние между городами (в рассматриваемом случае равное двум), то

$$M(z) = m \leq \lceil log_2 m \rceil + \frac{m^2}{2} - \frac{m}{2} < L(z).$$

Все условия принадлежности проблемы $Z_{\text{комм}}$ классу **NP**$-$ полных в сильном смысле проблем выполнены. Таким образом, доказана

Теорема 6.14. *Проблема КОММИВОЯЖЕР является* **NP**$-$*полной в сильном смысле.*

Поскольку выяснилось, что в классе **NPC** содержится проблема РАЗБИЕНИЕ, не являющаяся **NP**−полной в сильном смысле, и проблема КОММИВОЯЖЕР, являющаяся **NP**−полной в сильном смысле, то класс NPC можно представить как показано на рис. 6.41.

Таким образом, установлено, что среди универсальных переборных проблем, т.е. в классе **NPC**, содержатся в некотором смысле самые трудные – **NP**−полные в сильном смысле проблемы, которые нельзя решить не только полиномиальным, но даже псевдополиномиальным алгоритмом.

Определение 6.18. *Пусть Z и W – две произвольные проблемы распознавания свойств. Говорят, что проблема Z псевдополиномиально сводится к проблеме W (что обозначается $Z \propto_{nn} W$), если существует вычислимая функция $f : Z \to W$ такая, что*

1) для любой индивидуальной задачи z, в случае, когда она даёт ответ "ДА" задача $f(z) = w \in W$ также будет давать ответ "ДА";

2) функция f должна быть вычислима за время, ограниченное полиномом от двух переменных $M(z)$ и $L(z)$ (псевдополиномиальным алгоритмом);

3) существует такой полином $\mathscr{P}_1(\cdot)$, что для всех $z \in Z$ имеет место неравенство

$$L(z) \leq \mathscr{P}_1(L^{'}(f(z));$$

4) существует такой полином $\mathscr{P}_2(\cdot,\cdot)$, что для всех $z \in Z$ имеет место неравенство

$$M^{'}(f(z)) \leq \mathscr{P}_2(L(z), M(z)),$$

где $(L, M), (L^{'}, M^{'})$ – полиномиально эквивалентная пара функций, определяющих длину входа и максимальное число в определении задач $z \in Z$ и $w \in W$ *(см. стр. 163)**.*

Ответ на вопрос: как доказывать сильную **NP**−полноту, даёт следующая

Теорема 6.15.

$$(Z \in \mathbf{NPCS}) \wedge (W \in \mathbf{NP}) \wedge (Z \propto_{nn} W) \Rightarrow W \in \mathbf{NPCS}.$$

Приведём только *идею доказательства* теоремы, оставляя его детали для самостоятельной проработки с целью углублённого "погружения" в предмет.

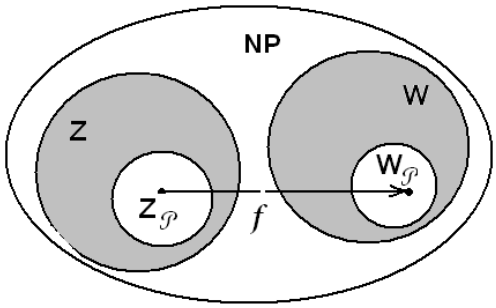


Рис. 6.42: К доказательству теоремы 6.15

Следует учесть, что псевдополиномиально вычислимая функция f, фигурирующая в определении 6.18 псевдополиномиальной сводимости, отображает **NP**− полную подпроблему $Z_{\mathscr{P}} \in Z$, не являющуюся проблемой с числовыми параметрами, в **NP**− полную подпроблему $W_{\mathscr{P}} \in W$, также не являющуюся проблемой с числовыми параметрами. Вместе с имеющимся условием $W \in \mathbf{NP}$ это даёт сильную **NP**−полноту проблемы W. □

Примерами **NP**−полных в сильном смысле проблем являются КОММИВОЯЖЕР, УПОРЯДОЧЕНИЕ ВНУТРИ ИНТЕРВАЛОВ, 3-РАЗБИЕНИЕ, 4-РАЗБИЕНИЕ [5].

6.8 Другие классы сложности

Определение 6.19. *Классом* **PSPACE** *называется множество вычислительных проблем, имеющих полиномиальную пространственную оценку сложности* (*используемой памяти*)*:*

$$\mathbf{PSPACE} = \{Z : SPACE_Z(n) \leq O(\mathscr{P}(n))\},$$

где $\mathscr{P}(n)$ *– полином от длины входа* n*.*

Для решения любой индивидуальной задачи каждой из проблем класса **PSPACE** найдется машина Тьюринга (алгоритм), которая при решении этой задачи использует размер рабочей зоны на ленте (число ячеек памяти), не превосходящее полинома от длины входа задачи.

Часто используют такие обозначения:

DTIME$(f(n))$ – класс вычислительных проблем, разрешимых за время, не превышающее $f(n)$, на детерминированной машине Тьюринга.

NDTIME$(f(n))$ – класс вычислительных проблем, разрешимых за время, не превышающее $f(n)$, на недетерминированной машине Тьюринга (или на другой N-модели или N-языке).

DSPACE$(f(n))$ – класс вычислительных проблем, для решения которых на детерминированной машине Тьюринга потребуется объём памяти, не превышающий $f(n)$.

NSPACE$(f(n))$ – класс вычислительных проблем, для решения которых на недетерминированной машине Тьюринга (или на другой N-модели или N-языке) потребуется объём памяти, не превышающий $f(n)$.

В теории сложности рассматривают следующие классы проблем.
DLOG = **DSPACE**$(O(\log n))$.
NLOG = **NSPACE**$(O(\log n))$.
PLOG = **DSPACE**$(\log^{O(1)} n)$.
P = **PTIME** = $\bigcup_{k>0}$ **DTIME**(n^k) = **DTIME**$(n^{O(1)})$.
NP = **NPTIME** = $\bigcup_{k>0}$ **NTIME**(n^k) = **NTIME**$(n^{O(1)})$.
PSPACE = **PS** = **DSPACE**$(n^{O(1)})$.
NPSPACE = **NPS** = **NSPACE**$(n^{O(1)})$.
DEXPT = **DTIME**$(O(1)^n)$.
NEXPT = **NTIME**$(O(1)^n)$.
EXPS = **DSPACE**$(O(1)^n)$.
DEXPTIME = **EXPTIME** = **EXP** = **DTIME**$(2^{n^{O(1)}})$.
NEXPTIME = **NTIME**$(2^{n^{O(1)}})$.
EXPSPACE = **DSPACE**$(2^{n^{O(1)}})$.

Очевидно, что **P** $\subseteq$ **PSPACE**. Более того, имеет место

Теорема 6.16. **NP** $\subseteq$ **PSPACE**.

Доказательство. Пусть проблема S распознавания свойства входит в класс **NP**. Тогда проверка свойства равносильна проверке сертификата y путём детерминированного полиномиального вычисления некоторого существующего проверочного предиката $Q(x, y)$, где x – входное слово – описание задачи, имеющее длину n. По определению класса **NP**, длина сертификата ограничена полиномом $len(y) \leq \mathscr{P}_y(n)$. Детерминированный алгоритм решения проблемы S может быть таким. Если алфавит, из которого состоят слова y, состоит из r символов, то путём полного перебора в одном и том

же сегменте памяти длины $\mathscr{P}_y(n)$ записываем по очереди все варианты сертификата y не более $r^{\mathscr{P}_y(n)}$ раз . Кроме этого, используем n элементов памяти для хранения входного слова. Как только $Q(x,y)$ оказывается равным единице, выдаём ответ "ДА". Если всегда $Q(x,y)=0$, ответ "НЕТ". Очевидно, занимаемый объём памяти, необходимый для выполнения этого детерминированного алгоритма составит $n+\mathscr{P}_y(n)$ и является полиномиальным. □

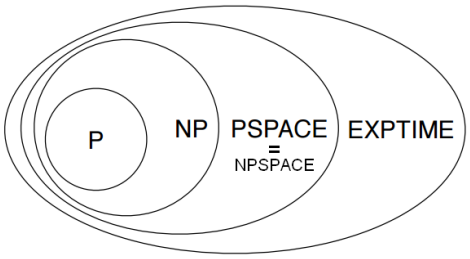


Рис. 6.43: Вложенность классов сложности

Рассмотрим другие соотношения между классами сложности.

PSPACE $\subseteq$ **EXPTIME**.

Действительно, если для решения проблемы вычисления свойств Z использует не более чем полином $\mathscr{P}(n)$ памяти ($Z \in$ **PSPACE**), то в этом полиномиальном сегменте памяти содержится вся информация, включая проверочное слово – сертификат y. Поэтому сложность решения не превысит полного перебора всех вариантов записи полинома, что будет ограничено функцией

$$r^{\mathscr{P}(n)} = 2^{\log_2 \mathscr{P}(n)} < 2^{n^{O(1)}},$$

где r – число символов в объединённом алфавите представления слов в памяти. □

PSPACE = **NPSPACE**. Доказательство этого равенства основано на теореме Сэвича, которая устанавливает справедливость включения **NSPACE**$(f(n)) \subseteq$ **DSPACE**$(f^2(n))$, где $f(n) \geq \log n$ для всех $n \geq 1$.

Определение 6.20. *Проблема распознавания свойства Z называется* **PSPACE**$-$*полной, если*

1) $Z \in$ **PSPACE**, *и*

2) любая проблема из **PSPACE** *полиномиально сводится к Z.*

Проблема КБФ $Z_{\text{КБФ}}$ *о квантифицированных булевых форму-*

лах состоит в следующем.

Дано: Формула вида

$$(Q_1x_1)(Q_2x_2)\dots(Q_mx_m)F(x_1,...,x_m), \tag{49}$$

где $x_1,...,x_m$ – булевы переменные, F – логическая формула, построенная из этих переменных с использованием операций $\{\wedge,\vee,^-\}$, а Q_i обозначает один из кванторов $\forall$ или $\exists$, $i=1,...,n$.

Свойство: является ли истинной формула F.

Теорема 6.17. *Проблема $Z_{КБФ}$ является* **PSPACE**$-$*полной.*

Доказательство.

I) Покажем, что $Z_{\text{КБФ}} \in$ **PSPACE**. Если входом задачи является формула (49) длины n, то часть этой формулы – выражение $F(x_1,...,x_m)$ имеет длину, не превышающую n. Поэтому для любого заданного набора булевых значений $\alpha_1\dots\alpha_n$ значение $F(\alpha_1\dots\alpha_n)$ можно вычислить, используя полиномиальный объём памяти $\mathscr{P}_1(n)$.

Будем использовать следующий алгоритм, повторяя вычисления для $k=1,...,m$.

1^o. Предположим, что формула (49) истинна: $Flag := 1$; $k=1$.

Вычислить $(Q_2x_2)\dots(Q_mx_m)F(0,x_2,...,x_m)$ и записать результат в дополнительный бит памяти β_1.

Вычислить значение β_2 формулы

$$(Q_2x_2)\dots(Q_mx_m)F(1,x_2,...,x_m),$$

и записать результат в дополнительный бит памяти β_2.

Если $Q_1=\forall$, и при этом β_1 или $\beta_2=0$,

то $\{Flag := 0$. Конец алгоритма $\}$.

Если $Q_1=\exists$, и при этом $\beta_1=\beta_2=0$,

то $\{Flag := 0$. Конец алгоритма $\}$.

Если $\{Flag := 1$, то имеется выполняющий набор $\sigma_1,...,\sigma_m$ такой что $(Q_2x_2)\dots(Q_mx_m)F(\sigma_1,...,\sigma_m)=1$. Присвоим $\alpha_1:=\sigma_1$. Получена формула

$$(Q_2x_2)\dots(Q_mx_m)F(\alpha_1,x_2,...,x_m).$$

k^o. Продолжаем вычисления по такой же схеме, как на шаге $k=1$, для $k=2,3,...,m$, решая подзадачи, состоящие в вычислении

истинности формулы

$$(Q_{k+1}x_{k+1})\dots(Q_m x_m)F(\alpha_1,...,\alpha_k,x_{k+1},...,x_m). \qquad (50)$$

Конец алгоритма. Если $Flag := 1$, то формула (49) истинна.

Вычисления формулы (50) могут быть произведены полным перебором – на каждом наборе переменных, связанных кванторами всеобщности, должен найтись набор значений переменных, связанным кванторами существования, обращающий формулу (50) в единицу. При этом объём занимаемой памяти не будет превышать полинома $\mathscr{P}_1(n)+O(n)+O(1) = \mathscr{P}_0(n)$, где $\mathscr{P}_1(n)$ оценивает поле записи формулы, $O(n)$ оценивает память для хранения вектора $\alpha_1,...,\alpha_k$, а $O(1)$ оценивает память, требуемую для дополнительных переменных – $Flag,\ k$.

Поэтому $Z_{\text{КБФ}} \in \mathbf{PSPACE}$.

II) Покажем, что любая проблема Z из **PSPACE** полиномиально сводится к $Z_{\text{КБФ}}$.

Построим за полиномиальное время по входу $x = x_z$ длины n произвольной индивидуальной задачи $z \in Z$ квантифицированную формулу $F = F_x$ такую, которая будет истинной тогда и только тогда, когда ответом задачи z будет "ДА".

Любая задача $z \in Z$ при её решении использует некоторый полиномиальный объём $\mathscr{P}_1(n)$ памяти и поэтому выполняется за время, не превышающее $2^{\mathscr{P}_1(n)}$, что соответствует полному перебору всех возможных записей на ленте МТ, включая конечную конфигурацию. При этом не исключается, что время решения может быть экспоненциальным. А если время решения может быть экспоненциальным, то полиномиальное сведение к формуле без кванторов, как при доказательстве только полиномиальности проверки свойства в теореме о **NP**−полноте проблемы ВЫПОЛНИМОСТЬ, оказывается невозможным.

Ответом задачи z будет "ДА" в том и только в том случае, когда существует алгоритм – машина Тьюринга $\mathscr{M}$, которая по входному слову $x = x_z$ корректно производит вычисления и возвращает ответ "ДА", причём $\mathscr{M}$ использует полиномиальный объём памяти $\mathscr{P}_1(n) = m$.

Выражением этой эквивалентности будет являться квантифицированная формула F, к построению которой мы приступаем.

Для описания любого *текущего слова* на ленте МТ $\mathscr{M}$, которое в процессе данного доказательства будем называть *конфигурацией*, потребуется объём памяти не более m. Поэтому всего может существовать не более 2^m различных конфигураций. МТ $\mathscr{M}$ переходит из одной конфигурации в другую, и если она не зацикливается, то путь от начальной конфигурации, обозначаемой переменной $Start$, в заключительную конфигурацию, обозначаемую переменной End и соответствующую ответу "ДА", будет иметь длину не более 2^m.

Определим последовательность квантифицированных формул $\Phi_i,\ \ i = 0, 1, ..., m$, где $\Phi_i(A, B)$ истинна только тогда, когда конфигурация, обозначаемая переменной B, достижима из конфигурации, обозначаемой переменной A, *не более чем* за 2^i шагов. Потребуем, чтобы при $i = m$ формула имела вид $\Phi_m(Start, End)$.

Заметим, что переменные $Start$, End, A, B и все прочие, обозначающие конфигурации, соответствуют каждая группе из m переменных, определяющих слова-конфигурации на ленте МТ. Будем называть эти переменные также конфигурациями, полагая что это не вызывает недоразумений.

При $i = 0$ квантифицированная формула $\Phi_0(X, Y)$ истинна, если $X = Y$ или Y достижимо из X за один шаг, поэтому она может быть представлена в виде ДНФ полиномиальной длины от m, логически перечисляющей (при помощи связки $\vee$) все варианты одношагового перехода из конфигурации в конфигурацию.

Далее, имея $\Phi_i(X, Y)$, получим

$$\Phi_{i+1}(X, Y) = \exists W[\{\Phi_i(X, W) \wedge \Phi_i(W, Y)\} \vee \Phi_i(X, Y)] \qquad (51)$$

$$= \exists W[\overline{\{\Phi_i(X, W) \wedge \Phi_i(W, Y)\}} \to \Phi_i(X, Y)]$$

$$= \exists W[\overline{\{\Phi_i(X, W)} \vee \overline{\Phi_i(W, Y)\}} \to \Phi_i(X, Y)]$$

Действительно, переход из X в Y не более чем за 2^{i+1} шагов машины Тьюринга ($\Phi_{i+1}(X, Y) = 1$) возможен если и только если $\Phi_i(X, Y) = 1$, когда переход из X в Y заведомо возможен не более чем за $2^i < 2^{i+1}$ шагов, или найдётся такая конфигурация W, что имеется возможность перехода из X в W не более чем за 2^i шагов

и одновременно с этим – из W в Y, также не более чем за 2^i шагов. Это условие формализуется выражением $\Phi_i(X,W) \wedge \Phi_i(W,Y)$, при выполнении которого переход из X в Y осуществим не более чем за $2^i + 2^i = 2^{i+1}$ шагов.

Однако рекурсивное построение формулы $\Phi_m(Start, End)$ на основе задания $\Phi_0(X,Y)$ и применения соотношения (51) приведет к экспоненциальному росту длины формулы, поскольку на каждом шаге рекурсии длина получаемой формулы будет утраиваться. Поэтому применим другую рекурсивную формулу, которая обеспечит не более чем полиномиальный рост длины выражений, получаемых на каждом шаге рекурсии.

Формула в квадратных скобках в правой части равенства (51) равносильна формуле

$$\forall G \forall H[(G = X)\&(H = W) \vee (G = W)\&(H = Y) \rightarrow \Phi_i(G,H)], \quad (52)$$

которую можно объяснить следующим образом. Транзитивная схе-

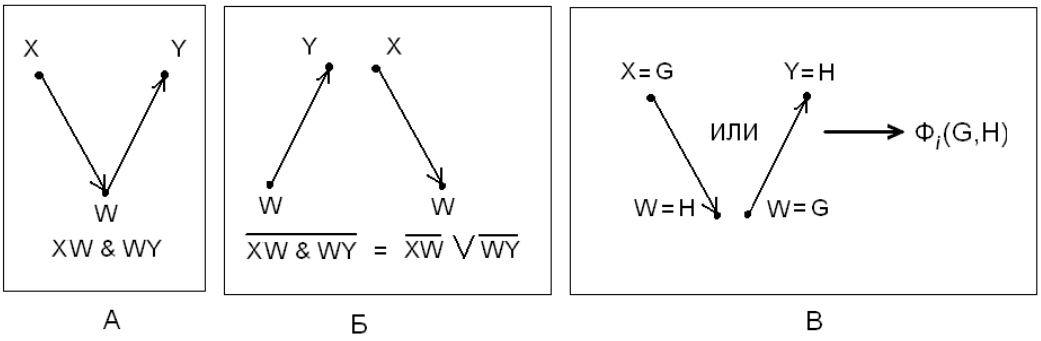


Рис. 6.44: К объяснению формулы (52)

ма перехода от конфигурации X к конфигурации Y (см. рис. 6.44 А) инвертируется (см. рис. 6.44 Б), и тогда остаётся её правая или левая ветвь. Схема 6.44 В поясняет формулу (52).

Какими бы мы ни взяли G и H в формуле (52), она будет иметь следующие значения:

если $G = X$ и $H = Y$, то импликация обратится в единицу, если $\Phi_i(G,H) = \Phi_i(X,Y) = 1$, т.е. существует переход из X в Y не более чем за 2^i шагов;

если $G = X$, $H \neq Y$, $H = W$, и при этом $G \neq W$ и $H \neq Y$, то импликация обратится в единицу, если $\Phi_i(X,W) \wedge \Phi_i(W,Y) = 1$,

т.е. при наличии перехода из X в Y за не более чем 2^{i+1} шагов (см. рис. 6.44 В);

если $G \neq X$, $G = W$, $H = Y$, и при этом и $H \neq W$, то импликация обратится в единицу, если $\Phi_i(X, W) \wedge \Phi_i(W, Y) = 1$, т.е. при наличии перехода из X в Y за не более чем 2^{i+1} шагов;

в тривиальном случае, если $G = H$ формула обращается в единицу, поскольку $\Phi_i(G, G) = 1$;

во всех остальных случаях формула (52) равна нулю.

Формула 51 принимает вид

$$\Phi_{i+1}(X, Y) = \tag{53}$$

$$\exists W \forall G \forall H \underbrace{[(G = X)\&(H = W) \vee (G = W)\&(H = Y) \to \Phi_i(G, H)]}_{\mathscr{F}(X,Y,W,G,H)},$$

откуда следует, что $\Phi_m(Start, End(1))$ можно вычислить при помощи соотношения (53) рекурсивно за m шагов, причём на первом шаге рекурсии длина получаемой формулы будет оцениваться некоторым полиномом $\mathscr{P}_2(n)$ и далее будет возрастать на некоторый полином $\mathscr{P}_3(n)$; в итоге длина выражения $\Phi_m(Start, End)$ не превысит полинома $\mathscr{P}_2(n) + m\mathscr{P}_3(n)$.

В итоге квантифицированная формула $F = F_x$, которая будет истинной тогда и только тогда, когда ответом задачи z на вход $x = x_z$ длины n произвольной индивидуальной задачи $z \in Z$ будет "ДА имеет вид

$$F = \exists Start\ \exists End\ [B(Start)\ \&D(End)\ \&\ \Phi_m(Start, End)], \tag{54}$$

где формулы $B(Start)$ и $D(End)$ выражают правильность соответственно начальной и заключительной конфигурации и имеют полиномиально ограниченную длину.

Все кванторы, входящие в правую часть формулы (54), можно вынести в её начало, и тогда получим

$$F = \exists Start\ \exists End\ \exists W\ \forall G\ \forall H\ \left(\mathscr{F}_{\{\wedge,\vee,\bar{}\}}(Start, End, W, G, H)\right), \tag{55}$$

где булева формула

$$\mathscr{F}_{\{\wedge,\vee,\bar{}\}}(Start, End, W, G, H) = B(Start)\ \&\ D(End)\ \&\ \mathscr{F}(Start, End, W, G, H)$$

представляет собой выражение, записанное в базисе $\wedge, \vee, ^-$ в результате использования тождеств [25]

$$a \to b = \bar{a} \vee b \quad \text{и} \quad (a \leftrightarrow b) = a\&b \vee \bar{a}\&\bar{b}.$$

6.9 Матроиды, жадные алгоритмы и сложность алгоритмического решения задач

Задача $\mathscr{M}1$. Дана матрица с целыми неотрицательными коэффициентами $A = \|a_{i,j}\|_{n\times n}$. Требуется выбрать такое подмножество её элементов, что

а) *в каждом столбце матрицы содержится не более одного элемента* и одновременно с этим

б) *сумма выбранных элементов максимальна.*

Эквивалентная формулировка задачи является частным случаем проблемы линейного целочисленного программирования с $\{0,1\}$-переменными:

$$\begin{cases} \sum_{i=1}^{n}\sum_{j=1}^{n} a_{i,j}x_{i,j} \to \max; \\ \sum_{i=1}^{n} x_{i,j} \le 1, \quad j = \overline{1,n}; \\ x_{i,j} \in \{0,1\}, \quad i = \overline{1,n}; j = \overline{1,n}. \end{cases} \tag{56}$$

Будем решать задачу $\mathscr{M}1$, отбирая последовательно элементы из матрицы $\|a_{i,j}\|_{n\times n}$ так, чтобы каждый раз *выбирать наибольший элемент* из оставшихся невыбранными, *проверяя при этом выполнение ограничений* $\sum_{i=1}^{n} x_{i,j} \le 1, \quad j = \overline{1,n}$. Алгоритмы такого типа называют *жадными*, *пожирающими*, а в англоязычной литературе – *GREEDY*. Жадные алгоритмы напоминают градиентные алгоритмы в численных методах непрерывной математики.

Легко убедиться, что жадный алгоритм правильно решит задачу $\mathscr{M}1$ с временно́й сложностью $O(n^2)$. Для примера рассмотрим задачу с матрицей

$$\begin{pmatrix} (10) & (8) & 4 \\ 6 & 7 & (6) \\ 5 & 6 & 4 \end{pmatrix}.$$

[25] с учётом того, что $a = b$ эквивалентно $a \leftrightarrow b$

В скобках отмечены элементы, которые выберет жадный алгоритм, каждый раз добавляя наибольшее число в столбце и получая в итоге сумму, равную 24.

Теперь усложним задачу.

Задача М 2. Дана матрица с целыми неотрицательными коэффициентами $A = \|a_{i,j}\|_{n\times n}$. Требуется выбрать такое подмножество её элементов, что

а) в каждом столбце *и в каждой строке* матрицы содержится не более одного элемента и одновременно с этим

б) *сумма выбранных элементов максимальна.*

Теперь в задаче содержится уже две группы ограничений – по столбцам и по строкам:

$$\begin{cases} \sum_{i=1}^{n}\sum_{i=j}^{n} a_{i,j}x_{i,j} \to \max; \\ \sum_{i=1}^{n} x_{i,j} \le 1, \quad j = \overline{1,n}; \\ \sum_{j=1}^{n} x_{i,j} \le 1, \quad i = \overline{1,n}; \\ x_{i,j} \in \{0,1\}, \quad i = \overline{1,n}; j = \overline{1,n}. \end{cases} \tag{57}$$

Выбрав самый большой элемент – число $a_{1,1} = 10$ в первой строке и первом столбце матрицы, жадный алгоритм исключит для себя возможность в дальнейшем выбрать какой-либо ещё элемент из первого столбца и первой строки в силу двойных ограничений задачи. Поэтому на оставшихся двух шагах он выберет $a_{2,2} = 7$ и $a_{3,3} = 4$, получая в итоге сумму, равную 21.

$$\begin{pmatrix} (10) & 8 & 4 \\ 6 & (7) & 6 \\ 5 & 6 & (4) \end{pmatrix}$$

Однако полученное решение не является оптимальным. Если взять в каждой строке и в каждом столбце по одному элементу как показано ниже, то полученная сумма выбранных элементов будет больше и составит 22.

$$\begin{pmatrix} (10) & 8 & 4 \\ 6 & 7 & (6) \\ 5 & (6) & 4 \end{pmatrix}$$

Дадим следующую достаточно общую постановку дискретной оптимизационной проблемы.

Задача $\mathscr{M}3$. Даны конечное множество $E = \{e_1, \ldots, e_n\}$ произвольных элементов, некоторое семейство его подмножеств $J \subseteq 2^E$, где 2^E – булеан, и функция $w : E \to \mathbb{Q}^+$.

Требуется найти подмножество $S \in J$ с наибольшей суммой $\sum_{e \in S} w(e)$:

$$\begin{cases} \max_{S \in J} \sum_{e \in S} w(e); \\ J \subseteq 2^E; \\ E = \{e_1, \ldots, e_n\}. \end{cases} \tag{58}$$

Значение $w(e)$ будем называть *весом* элемента $e \in E$. Задача $\mathscr{M}3$ может быть переформулирована так: найти допустимое подмножество $S \in J$ с наибольшим суммарным весом.

Для удобства иногда будем обозначать суммарный вес произвольного множества $V \subseteq E$ как $w(V) = \sum_{e \in V} w(e)$.

Определение 6.21. *Жадным* ($GREEDY$) *называется следующий алгоритм решения задачи М3:*

1. `Упорядочить веса элементов`
   $w(e[1]) \geq w(e[2]) \geq ... \geq w(e[n])$;
2. $S := \emptyset$;
3. `Для` i `от` 1 `до` n `выполнить`
   `Если` $(S \cup \{e[i]\}) \in J$ `то` $S := S \cup \{e[i]\}$.

Замечание. Для упрощения в определении 6.21 алгоритма $GREEDY$ полагается, что после упорядочения элементов множества E по весу они получили имена (и номера) $e[1], e[2], \ldots, e[n]$.

Типичным примером применения алгоритма $GREEDY$ является стратегия "иди в ближайший" для задачи о кратчайшем пути в графе.

Сложность алгоритма $GREEDY$ определяется сложностью задачи сортировки $n-$элементного массива, которая оценивается как $O(n \log n)$, плюс сложность $n-$кратной проверки свойства "$(S \cup \{e[i]\}) \in J$". Если эта проверка реализуема с полиномиальной сложностью, то алгоритм $GREEDY$ в целом будет иметь полиномиальную сложность.

Простота алгоритма $GREEDY$ делает его привлекательным и определяет важность постановки следующего вопроса: *каким усло-*

виям должно удовлетворять семейство J в задаче $\mathscr{M}3$, чтобы алгоритм $GREEDY$ правильно решал эту задачу для произвольной функции $w : E \to \mathbb{Q}^+$?

Оказывается, что для правильного решения задачи $\mathscr{M}3$ алгоритмом $GREEDY$ достаточно, чтобы пара $< E, J >$ образовывала *матроид.*

Определение 6.22. *Матроидом называется пара $< E, J >$, где E – конечное множество, $J \subseteq 2^E$ – семейство его подмножеств, удовлетворяющее следующим двум условиям, которые называют аксиомами матроида:*

M1) $\emptyset \in J$, и если $(A \in J) \wedge (B \subseteq A)$, то $B \in J$;

M2) для любых множеств A и B таких, что $A \in J$, $B \in J$ и $|B| = |A| + 1$, найдётся элемент $e \in B \setminus A$ такой, что $A \cup \{e\} \in J$.

Множества семейства J называют *независимыми*, а остальные множества из 2^E – *зависимыми.*

Определение 6.23. *Максимальным независимым подмножеством множества $C \subseteq E$ называется такое независимое подмножество $A \subseteq C$, что не существует независимого множества B, удовлетворяющего условию $A \subset B \subseteq C$.*

Теорема 6.18. *Пусть E – конечное множество, $J \subseteq 2^E$ и J удовлетворяет условию M1 из определения 6.22. Тогда пара $< E, J >$ является матроидом в том и только в том случае, если выполняется условие*

M3) для любого подмножества $C \subseteq E$ каждые два максимальных независимых подмножества этого множества C имеют одинаковую мощность.

Доказательство. Теорема устанавливает равносильность для определения матроида аксиом M2 и M3, поэтому следует доказать, что при условии, когда выполнена аксиома $M1$, имеет место $M2 \Rightarrow M3$ и $M3 \Rightarrow M2$.

а) Пусть аксиома матроида M3 не выполняется. Тогда найдётся множество $C \subseteq E$ такое, в котором два максимальных независимых множества $A \subset C$ и $B \subset C$, $A \in J$, $B \in J$, имеют разную мощность: $|B| > |A|$. Пусть $B^* \subset B$ такое сужение множества B, что $|B^*| = |A| + 1$. Если бы выполнялась аксиома M2, то взяв элемент $e \in$

$B^* \setminus A$ можно было бы расширить независимое множество A, получая $A^* = A \cup \{e\}$, $A^* \in J$. Но это противоречило бы максимальности независимого множества A в C. Поэтому $\neg M3 \Rightarrow \neg M2$.

б) Пусть теперь не выполняется аксиома матроида М2. Тогда найдутся два множества A и B такие, что $A \in J$, $B \in J$, $|B| = |A|+1$, но $A \cup \{e\} \notin J$ для любого $e \in B \setminus A$. Пусть $C = A \cup B$. Тогда A и B – два независимых неравномощных множества в C, причём множество A расширить в C, сохранив независимость, невозможно. Из этого следует, что $A \not\subset C$, потому что противное противоречило бы аксиоме М1. Таким образом, A – максимальное независимое множество в C, а B или, может быть, его расширение – также максимальное независимое множество в C, но имеющее большую мощность, чем множество A. Откуда следует $\neg M3$.

Определение 6.24. *Мощность максимального независимого подмножества произвольного множества $C \subseteq E$ называется рангом этого множества и обозначается $r(C)$.*

Максимальные независимые подмножества множества E называются базами матроида $< E, J >$.

Ранг множества E называется рангом матроида $r(E)$.

Следствие из теоремы 6.18. Любые две базы матроида равномощны.

Теорема 6.19 (Радо-Эдмондса)**.** *Если пара $< E, J >$ является матроидом, то множество S, найденное алгоритмом $GREEDY$ из определения 6.21 при решении задачи $\mathscr{M}3$, действительно является допустимым подмножеством с наибольшим суммарным весом.*

Если же пара $< E, J >$ не является матроидом, то найдётся такая функция $w : E \to \mathbb{Q}^+$, что множество S, найденное алгоритмом $GREEDY$, не будет являться множеством с наибольшим суммарным весом.

Доказательство.

1) Пусть $< E, J >$ – матроид, $S = \{s_1, ..., s_k\}$ – независимое множество, построенное алгоритмом $GREEDY$. Тогда $w(s_1) \geq w(s_2) \geq \cdots \geq w(s_k)$.

Множество S – база матроида, поскольку никакой элемент $e \in E$, не включенный в S при выполнении алгоритма $GREEDY$, не

может пополнить множество S. Действительно, если на некотором шаге t выполнения алгоритма было сформировано независимое множество $S_t \subset S$ и затем элемент e_i был отвергнут как нарушающий условие $(S_t \cup \{e_i\}) \in J$, то для любого независимого множества A такого, что $S_t \subset A \subset S$ выполнение условия $(A \cup \{e_i\}) \in J$ означало бы что $(S_t \cup \{e_i\}) \subset (A \cup \{e_i\})$. Но $(S_t \cup \{e_i\}) \notin J$, что противоречит аксиоме М1 матроида. Поэтому расширяя множество S_t на следующих шагах алгоритма, добавить к нему отвергнутые ранее элементы из E невозможно.

Рассмотрим *любое независимое множество элементов* $V = \{v_1, ..., v_m\}$, *отличное от* $S = \{s_1, ..., s_k\}$, полагая, что элементы множества V упорядочены по весу: $w(v_1) \geq w(v_2) \geq \cdots \geq w(v_m)$. Покажем, что $k \geq m$. Действительно, если предположить, что $k < m$, то $|V| > |S|$, и тогда в соответствии с аксиомой матроида М2 в сужении $V^* \subset V$ таком, что $|V| = |S| + 1$, обязан найтись элемент $e \in V \setminus S$ такой, что $S \cup \{e\} \in J$. Но это противоречит тому, что S – база матроида.

Покажем теперь, что $w(S) \geq w(V)$, т. е. что *алгоритм* $GREEDY$ *выбирает множество* S, *суммарный вес которого не меньше, чем суммарный вес любого другого независимого множества* V. Достаточно показать, что $w(s_j) \geq w(v_j)$ для всех $j \in \overline{1, m}$ (напомним, что и множество S, и множество V упорядочено по неубыванию, и $k \geq m$).

Базис индукции. $w(s_1) \geq w(v_1)$, поскольку по определению алгоритма $GREEDY$ элемент $w(s_1)$ является максимальным в E.

Индуктивный переход. Пусть $w(s_{j-1}) \geq w(v_{j-1})$. Предположим $w(s_j) < w(v_j)$ и рассмотрим два независимых множества $C = \{s_1, ..., s_{j-1}\}$ и $D = \{v_1, ..., v_{j-1}, v_j\}$. По аксиоме М2 множество $F = \{s_1, ..., s_{j-1}, v_j\}$ будет независимым, поскольку $v_j \in D \setminus C$. Тогда алгоритм $GREEDY$ должен был выбрать не элемент s_j, а элемент v_j, что приводит к противоречию.

2) Пусть пара $< E, J >$ не является матроидом. Тогда нарушается одна из аксиом матроида – М1 или М2.

Пусть не выполняется аксиома М1. Тогда найдутся два множества A и B, $A \neq B$, такие, что $A \subset B$, $B \in J$, но $A \notin J$. Определим

функцию w так:

$$w(e) = \begin{cases} 1, & \text{если } e \in A; \\ 0, & \text{если } e \in E \setminus A. \end{cases}$$

Алгоритм $GREEDY$ начнёт выбирать элементы множества A с максимальным весом, равным 1, но не сможет выбрать все элементы множества A, поскольку оно не принадлежит системе независимости J. Поэтому алгоритм $GREEDY$ выберет множество, имеющее суммарный вес $w(S) < |A|$. Но $A \subset B$, поэтому вес независимого множества B равен $w(B) = |A| > w(S)$. Следовательно, алгоритм $GREEDY$ выберет множество S, не являющееся независимым множеством с максимальным суммарным весом.

Если же аксиома M1 выполняется, но *не выполняется аксиома* M2, то найдутся множества $A \in J$ и $B \in J$ такие, что $|A| = k$, $|B| = k + 1$, но для любого $e \in B \setminus A$ будет иметь место $(A \cup \{e\}) \notin J$.

Обозначим $p = |A \cap B|$; тогда $p < k$, поскольку в противном случае, если $p = k$, то $A \subset B$, и тогда $(A \cup \{e\}) \in J$ согласно аксиоме M1, что приведёт к противоречию. Построим такую функцию w:

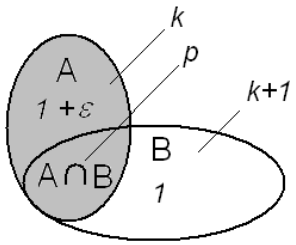


Рис. 6.45: К доказательству теоремы 6.19: построение функции w при невыполнении аксиомы M2

$$w(e) = \begin{cases} 1+\varepsilon, & \text{если } e \in A; \\ 1, & \text{если } e \in B \setminus A; \\ 0 \ - & \text{в остальных случаях.} \end{cases}$$

При таком задании функции w алгоритм $GREEDY$ сначала выберет все элементы множества с весом $1 + \varepsilon$ (здесь $\varepsilon > 0$), но не сможет добавить к выбранному множеству S ни одного элемента из B. Поэтому вес отобранного множества будет $w(S) = k(1 + \varepsilon)$. Но вес независимого множества B составит (см. рис. 6.45) $w(B) = p(1 + \varepsilon) + (k + 1 - p) \cdot 1$, и может оказаться, что $w(B)$ будет больше $w(S)$. Действительно

$$w(B) = p + p\varepsilon + k + 1 - p > k + k\varepsilon = w(S)$$

при $p\varepsilon + 1 > k\varepsilon$, что эквивалентно, при $\varepsilon < \frac{1}{k-p}$. $\square$

При изучении сложности решения задач алгоритмическим методом рассматриваются *проблемы – семейства однотипных индивидуальных задач.* Такой подход применяется потому, что отдельные совокупности задач, входящие в труднорешаемые проблемы, иногда могут быть решены эффективно (с полиномиальной трудоемкостью).

Например, в задачах целочисленного линейного программирования с 0-1 переменными, составляющих в целом **NP**−трудную проблему $Z_{\text{ЦЛП01}}$, *имеется полиномиально разрешимый подкласс*, что следует из теоремы 6.19 Радо-Эдмондса.

Действительно, запишем произвольную задачу из $Z_{\text{ЦЛП01}}$ в виде

$$\begin{cases} \max(c_1x_1 + c_2x_2 + \cdots + c_nx_n); \\ (x_1, x_2, \ldots, x_n) = \tilde{x} \in J; \\ x_i \in \{0, 1\}, \quad i = 1, 2, ..., n, \end{cases}$$

где $J = \{\tilde{x} : A\tilde{x} = B\}$ – множество удовлетворяющих линейным ограничениям булевых векторов $\tilde{x}$. Положим, что выбор элемента e_i из множества $E = \{e_1, ..., e_i, ..., e_n\}$ происходит тогда и только тогда, когда $x_i = 1$, а веса элементов множества E определены как $w(e_i) = c_i \geq 0, \quad i = 1, 2, ..., n$. Булевы векторы соответствуют подмножествам элементов из E:

$0...000 \quad \sim \quad \{\emptyset\},$
$0...001 \quad \sim \quad \{e_1\},$
$0...010 \quad \sim \quad \{e_2\},$
$0...011 \quad \sim \quad \{e_1, e_2\},$
....

$1...111 \quad \sim \quad \{e_1, e_2, \ldots, e_n\}.$

Указанное представление проблемы $Z_{\text{ЦЛП01}}$ делает очевидным

Следствие. *Если в задаче целочисленного линейного программирования с* 0-1 *переменными множество допустимых решений* J *таково, что пара* $< E, J >$ *является матроидом, то эта задача разрешима за полиномиальное время*[26].

[26]алгоритмом *GREEDY*

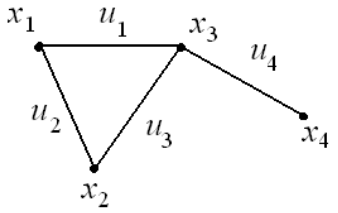


Рис. 6.46: Пример к построению графового матроида

Графовый матроид. Пусть дан произвольный конечный неориентированный граф $\Gamma =< X, U >$, а $J \subseteq 2^U$ – такая совокупность подмножеств рёбер графа Γ, что для любого множества $W \in J$ граф $< X, W >$ *не содержит циклов*. Тогда пару $< U, J >= M_\Gamma$ называют *графовым матроидом*.

На рис. 6.46 приведен пример графа с четырьмя вершинами $X = \{x_1, ..., x_4\}$ и четырьмя рёбрами $U = \{u_1, ..., u_4\}$. Для этого графа семейство J состоит из множеств $\{\emptyset\}$, $\{u_1\}$, $\{u_2\}$, $\{u_3\}$, $\{u_4\}$, $\{u_1, u_2\}$, $\{u_1, u_3\}$, $\{u_1, u_4\}$, $\{u_2, u_3\}$, $\{u_2, u_4\}$, $\{u_3, u_4\}$, $\{u_1, u_2, u_4\}$, $\{u_1, u_3, u_4\}$, $\{u_2, u_3, u_4\}$.

Каждый графовый матроид $M_\Gamma =< U, J >$ порождается некоторым заданным графом $\Gamma =< X, U >$ и строится на основе множества рёбер этого графа Γ.

Покажем, что пара $M_\Gamma =< U, J >$, названная графовым матроидом, действительно является матроидом – удовлетворяет аксиомам М1 и М3.

Действительно, граф, не содержащий циклов, является *деревом* (если он связный) или, в противном случае, *лесом*, состоящим из конечного числа деревьев – компонент связности. Используя метод математической индукции, легко показать, что любое дерево, содержащее r вершин, имеет ровно $r - 1$ рёбер. Если лес содержит s компонент связности с числом вершин в каждой $k_1, ..., k_s$, то такой лес имеет ровно $\sum_i^s (k_i - 1) = n - s$ рёбер, где n – суммарное число вершин леса.

Рассмотрим *любое максимальное подмножество* $W \subseteq C$ в произвольном множестве $C \subseteq U$ такое, что граф $< X, W >$ не содержит циклов. Если граф $< X, C >$ связен, то $< X, W >$ – дерево, и мощность любого максимального независимого множества определяется соотношением $|W| = |X| - 1$. Если граф $< X, C >$ не связен и имеет s компонент связности, то $< X, W >$ – лес, и тогда мощность любого максимального независимого подмножества есть $|W| = |X| - s$. Таким образом, мощности любых максимальных независимых под-

множеств произвольного множества $C \subseteq U$ равны, что обеспечивает для пары $< U, J >$ выполнение аксиомы матроида М3.

Поскольку пустое подмножество рёбер определяет граф с изолированными вершинами не содержащий циклов, имеем $\emptyset \in J$. Если произвольный граф $< X, B >$ не содержит циклов, то и любой его подграф $< X, A >$ такой, что $A \subset B$ не будет содержать циклов, откуда следует: $(A \subset B) \wedge (B \in J) \Rightarrow A \in J$. Поэтому аксиома М1 также выполняется для пары $< U, J >$. $\square$

Практическая полезность понятия графового матроида становится очевидной при рассмотрении алгоритма Краскáла [27] решения задачи о кратчайшем стягивающем лесе, которая состоит в следующем.

Задача SF (о кратчайшем стягивающем лесе).

Дан конечный граф $\Gamma =< X, U >$ с заданной функцией цены рёбер $w : U \to \mathbb{Q}^+$.

Найти в графе Γ подграф $\Gamma^* =< X, W >,\ W \subseteq U,$ такой, чтобы

а) он не содержал циклов,

б) был связным в каждой компоненте связности исходного графа Γ (чтобы каждая пара вершин компоненты связности оставалась связанной ребром) и

в) суммарная цена его рёбер $w(W) = \sum_{u \in W} w(u)$ была минимальной.

Замечание. Если исходный граф Γ связен, то требуется отыскать в нём подграф-дерево, связывающее все вершины и имеющее минимальную суммарную цену рёбер. А если исходный граф Γ не является связным, то требуется отыскать в нём такой подграф-лес с минимальной суммарной ценой всех рёбер, каждая компонента которого будет деревом, связывающим все вершины соответствующей компоненты связности исходного графа Γ, с минимальной суммарной ценой всех рёбер.

Иногда задачу ***SF*** называют задачей о кратчайшей связывающей сети. Она может возникнуть, например, при построении самой недорогой сети дорог, связывающей населённые пункты.

[27] Иногда пишут: алгоритм Крускала, не совсем правильно основываясь на английском написании фамилии американского математика: Joseph Bernard Kruskal (1928 — 2010)

Очевидно, что семейство $J \in 2^U$ всех допустимых множеств рёбер, удовлетворяющих условиям задачи, вместе с исходным множеством рёбер U образуют графовый матроид $< U, J >$. Поэтому в силу теоремы Радо-Эдмондса задача ***SF*** может быть точно решена алгоритмом $GREEDY$. Таковым является следующий

Алгоритм Краскала.

1. Упорядочить веса рёбер исходного графа
   $\Gamma =< X, U >,\ |X| = n,\ |U| = m,$
   $w(u_1) \le w(u_2) \le ... \le w(u_m)$;
2. $S := \emptyset$;
3. $PART := \{A_1, ..., A_n\}$, где $A_i = \{x_i\}$ – одноэлементные множества: сначала в каждом – по одной вершине;
4. Для j от 1 до m выполнить
   если $u_j = (a, b)$ – такое ребро с концами-вершинами a, b,
   что $a \in A,\ b \in B,\ A \in PART,\ B \in PART,\ A \ne B,$
   то $\left\{ S := S \cup u_j;\ PART := ((PART \setminus A) \setminus B) \cup \{A \cup B\} \right\}$.

Алгоритм Краскала, являющийся алгоритмом типа $GREEDY$, построен так, что сначала разбивает все вершины графа Γ в одноэлементные множества, которые входят в подлежащий дальнейшей обработке набор множеств вершин $PART$. Затем в цикле соединяет последовательно пары вершин, образующих ребро с наименьшей ценой, проверяет отсутствие цикла в выстраеваемом графе и добавляет к этому графу очередное ребро в случае, если это не противоречит условию задачи.

На рис. 6.47 приведен ***пример пошагового выполнения алгоритма Краскала***. Шаг 0 соответствует исходному графу и начальному разбиению $PART = \{a\} \cup \{b\} \cup \{c\} \cup \{d\} \cup \{e\} \cup \{f\} \cup \{g\} \cup \{h\} \cup \{i\}$ вершин графа на одноэлементные множества. На шаге 1 выбирается ребро с минимальной ценой 1 и добавляется к изначально пустому множеству рёбер. Разбиение $PART$ модифицируется: $PART = \{\mathbf{a}, \mathbf{b}\} \cup \{c\} \cup \{d\} \cup \{e\} \cup \{f\} \cup \{g\} \cup \{h\} \cup \{i\}$.

На следующем шаге 2 к отобранному множеству рёбер добавляется ребро (b, c), имеющее наименьшую из оставшихся цену 2, и $PART$ модифицируется: $PART = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\} \cup \{d\} \cup \{e\} \cup \{f\} \cup \{g\} \cup \{h\} \cup \{i\}$. После этого шага ребро (a, c) выбрать уже нельзя: его вершины входят в одно и то же множество $\{a, b, c\}$ разбиения $PART$.

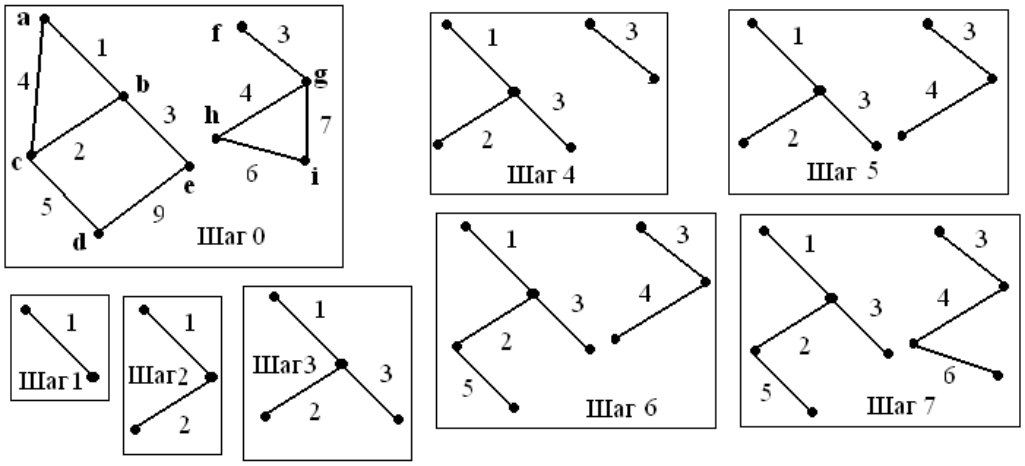


Рис. 6.47: Пример работы алгоритма Краскала по шагам

Продолжая шаги 3,4,5,6,7 обнаруживаем, что после выполнения шага 7 добавить ко множеству S ни одно ребро уже невозможно, не избежав появления цикла.

Обозначим $n = |X|$ – число вершин, а $m = |U|$ – число рёбер графа $\Gamma =< X, U >$.

На первом этапе выполнения алгоритма Краскала осуществляется сортировка рёбер. Сложность такой сортировки можно оценить как $O(m \log m)$. На втором этапе m раз повторяется проверка возможности добавления ребра к кратчайшему стягивающему лесу и его добавление, если возможно. Для этого используется последовательное разбиение вершин графа на непересекающиеся подмножества-блоки так, что нахождение пары вершин, являющихся концами анализируемого ребра, в одном блоке, говорит о том, что это ребро нельзя добавить во множество S, не нарушив требование отсутствия циклов в стягивающем лесе.

Сложность второго этапа алгоритма зависит от того, как организован процесс модификации блоков разбиения и проверки: находятся ли концевые вершины проверяемого ребра в одном блоке.

Простейший способ этой организации, достаточный для того, чтобы показать полиномиальность алгоритма Краскала, основан на обработке массива длины n по шагам выполнения алгоритма, как показано на рис. 6.48. За один проход по этому массиву можно определить: возможно ли добавление ребра и произвести модификацию массива. Поэтому сложность второго этапа можно оценить

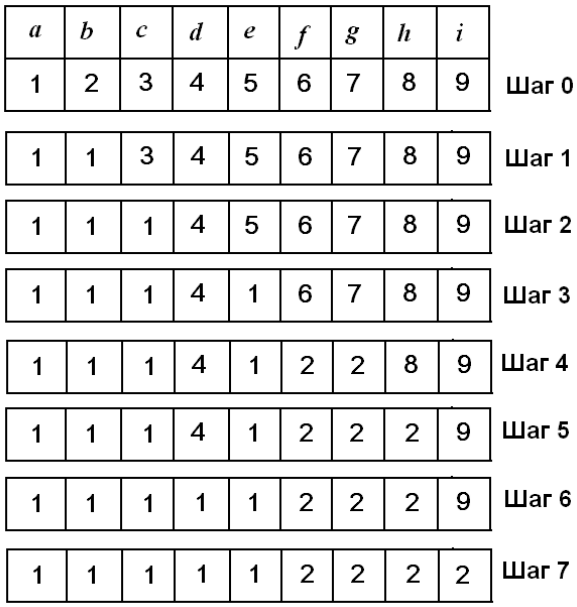


Рис. 6.48: Модификация разбиения вершин

как $O(mn)$ (включая в эту оценку инициализацию рассматриваемого массива).

Тогда сложность всего алгоритма может быть оценена как $O(m \log m) + O(mn)$.

Учитывая, что $m < n^2$, $\log m < 2 \log n$, $m \log m + mn < 2m \log n + mn < 3mn$, можно заключить, что сложность алгоритма оценивается как $O(mn)$ и, следовательно, *является полиномиальной.*

В нетривиальном случае $m > n - s$, где s – число компонент связности графа $\Gamma =< X, U >$, и $m > s$. Тогда $2m > n$ и $mn < 2m^2$, поэтому сложность алгоритма может быть оценена через число рёбер графа как $O(m^2)$.

Приведенный способ обработки блоков в алгоритме Краскала не является наилучшим. Известен подход на основе быстрого алгоритма объединения непересекающихся множеств, в котором каждый блок представляется ориентированным деревом, и тогда поиск в массиве, использованный выше, заменяется поиском по деревьям со сложностью $O(\log m)$, что позволяет получить реализацию алгоритма Краскала со сложностью $O(m \log m)$ [1, с. 201].

7 Колмогоровская сложность и количество информации

7.1 Введение

Рассмотренные во второй главе подходы к измерению количества информации – мера Хартли и мера Шеннона – широко применяются для количественной оценки информации. Однако можно указать особенности этих мер, которые в некоторых случаях ограничивают их применение.

Предположим, по каналу передачи данных переданы и получены три разных слова одинаковой длины – по 20 символов:

$$x_1 = 01010101010101010101;$$

$$x_2 = 12345123451234512345;$$

$$x_3 = 18379520415276083964.$$

Для упрощения рассуждений, не теряя общности, положим, что каждый символ занимает один байт памяти.

Пользуясь мерой Хартли, можно оценить, что в каждом случае получено по 20 байт информации, и, следовательно, все три слова (согласно этой мере) доставляют равное количество информации.

Но слово x_1 представляется очень простым: чтобы описать это слово, достаточно получить такую информацию: "повторить слово 01 десять раз подряд".

Слово x_2 описывается немного более сложной информацией: "повторить последовательность 12345 четыре раза".

Для слова x_3 никакая *закономерность* не обнаруживается: это слово *получено в результате равновероятного случайного выбора* одной из цифр множества $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ последовательно и независимо 20 раз. И то, что в нём первые четыре цифры случайно составили год смерти А. С. Пушкина, не поможет получить какое-нибудь простое описание всего этого слова.

Мера Шеннона, основанная на вероятностях появления символов ансамбля, из которых состоит передаваемое сообщение, неприменима для каждого отдельно взятого индивидуального слова.

Следовательно, для оценивания количества информации в индивидуальном объекте, например, в строке символов, меры Хартли и Шеннона не приспособлены.

Если *зафиксировать какой-либо способ описания строк символов*, то можно положить, что *кратчайшее из всех возможных описаний рассматриваемой строки x этим зафиксированным способом (обозначим его $K(x)$) будет мерой количества информации, требуемой для порождения слова x*, и одновременно мерой сложности этого слова.

Далее положим, что условная сложность $K(x|y)$ будет обозначать количество информации, которое необходимо добавить к информации, содержащейся в слове y, чтобы получить слово x. Тогда количество информации в y об x можно определить как

$$I(y:x) = K(x) - K(x|y).$$

Поскольку $K(x|x)$ по смыслу введенных понятий должно быть равно нулю (если дана информация x, то сама строка x уже получена), то

$$I(x:x) = K(x) - K(x|x) = K(x). \tag{59}$$

Поэтому *сложность $K(x)$ можно назвать количеством информации, содержащейся в объекте x о себе самом.*

Основываясь именно на алгоритмическом способе описания строк (и других конструктивных объектов), удалось создать алгоритмическую теорию сложности, алгоритмическую теорию вероятности и алгоритмическую теорию информации. Основные результаты в этом научном направлении были получены А. Н. Колмогоровым и его учениками.

В 1964 г. А. Н. Колмогоров ввел понятие сложности конечного объекта (например, слова в некотором алфавите). Сложность определялась как минимальное число двоичных знаков, содержащих всю информацию о заданном объекте, достаточную для его восстановления (декодирования). Это определение существенно зависело от метода декодирования, однако с помощью общей теории алгоритмов А. Н. Колмогорову удалось дать инвариантное (универсальное) определение сложности.

Близкие понятия рассматривались американским математиком Р. Дж. Соломоновым и советским математиком А. А. Марковым.

На базе понятия сложности А. Н. Колмогоров дал определение количества информации, содержащейся в конечных объектах, и определение понятия случайной последовательности, которое уточнил в своих работах его ученик, шведский математик П. Мартин-Лёф.

А. Н. Колмогоров писал: "Вполне традиционно представление, что "случайность" состоит в отсутствии "закономерности". Но, по-видимому, только сейчас возникла возможность основать точные формулировки условий применимости к реальным явлениям результатов математической теории вероятностей непосредственно на этой простой идее" [6, с. 243].

Действительно, каждое из слов x_1 и x_2 содержит простую закономерность и не представляются случайными. Слово x_3 на самом деле сгенерировано случайно, и закономерность в нём не обнаруживается.

7.2 Основные понятия колмогоровской теории сложности

Определение 7.1. *Колмогоровская сложность слова x при заданном способе описания – вычислимой функции (декомпрессоре) D есть*

$$KS_D(x) = \min\{l(p) | D(p) = x\},$$

если существует хотя бы одно двоичное слово p такое, что $D(p) = x$. Иначе полагается, что значение сложности не ограничено (∞), и в этом случае говорят, что колмогоровская сложность не определена.

Здесь и далее $l(p)$ обозначает длину слова p в битах.

Напомним, что вычислимыми функциями называются только такие, которые могут быть реализованы машинами Тьюринга, либо, что эквивалентно, являются частично-рекурсивными, либо представимы другими алгоритмическими моделями, эквивалентными двум указанным.

Определение 7.2. *Условная колмогоровская сложность слова x при заданном слове y есть*

$$KS_D(x|y) = \min\{l(p)|D(p,y) = x\};$$

если y – пустое слово, то $KS_D(x|y) = KS_D(x)$.

Декомпрессор $D = D(p, y)$ в определении условной колмогоровской сложности имеет два аргумента: кроме слова p добавляется дополнительное слово y. Можно сказать, что в этом случае декомпрессор получает больше информации для восстановления слова x.

Определение 7.3. *Говорят, что декомпрессор D_1 не хуже декомпрессора D_2, если $KS_{D_1}(x|y) \leq KS_{D_2}(x|y) + O(1)$. Декомпрессор называют оптимальным, если он не хуже любого другого декомпрессора.*

Теорема 7.1. (*Соломонова-Колмогорова*) *Существуют оптимальные декомпрессоры.*

Доказательство. Покажем, что найдется такая частично рекурсивная функция-декомпрессор A, что для любой другой частично рекурсивной функции-декомпрессора $D = D(p, y)$ будет выполнено неравенство

$$KS_A(x|y) \leq KS_D(x|y) + c_D. \tag{60}$$

Здесь c_D – константа, не зависящая от x и y. Значение этой константы определяется только тем, какой именно декомпрессор (способ описания) D использован в правой части неравенства (60).

Используя универсальную частично рекурсивную функцию [28] U с подходящим номером n, для любого декомпрессора D можно записать равенство

$$D(p, y) = U(n, (p, y)) = x. \tag{61}$$

Колмогоровская сложность относительно декомпрессора D определяется выражением

$$KS_D(x|y) = l(p).$$

[28] См. приложение С, стр. 221

Далее, осуществляя группировку аргументов, можно определить функцию A следующим образом:

$$A((n,p),y) = U(n,(p,y)) = x. \tag{62}$$

Пара слов (n,p) рассматривается как одна двоичная строка-аргумент, но при этом нужно сохранить возможность выделения из такой объединённой строки значений n и p. Следовательно, в строку (n,p), *возможно, понадобится вставить какой-нибудь разделитель подстрок* n *и* p. Обозначим этот разделитель δ, и тогда объединённую строку можно считать конкатенацией $n\delta p$.

Согласно равенствам (61) и (62) будем иметь

$$A((n,p),y) = D(p,y) = x$$

для любого допустимого декомпрессора D. Поэтому для любого номера функции n, определяющего декомпрессор D, найдется константа $c_D \geq l(n)$, зависящая только от выбора этого декомпрессора, такая, что

$$KS_A(x|y) = l(n\delta p) = \underbrace{l(n)}_{\leq c_D} + \underbrace{l(p)}_{KS_D(x|y)} + \underbrace{l(\delta)}_{\geq 0} \leq KS_D(x|y) + c_D,$$

где константа δ определяет дополнительное число бит, которое может потребоваться для того, чтобы входящий в конкатенацию np номер используемой универсальной функции n мог быть отделен от аргумента p. Это можно сделать разными способами независимо от слова p, например, при помощи специального так называемого самоограничивающего кодирования. $\square$

Поскольку доказано существование оптимальных декомпрессоров, далее можно рассматривать колмогоровскую сложность относительно произвольного оптимального декомпрессора D_{opt} *и обозначать её как*

$$KS(\cdot) = KS_{D_{opt}}(\cdot).$$

Таким образом, $KS(x)$ и $KS(x|y)$ обозначают колмогоровскую сложность относительно оптимального декомпрессора в отличие от обозначений $KS_{D_1}(x)$ и $KS_{D_2}(x|y)$, указывающих на использование некоторых фиксированных декомпрессоров D_1 и D_2.

Определение 7.4. *Функция $f(x)$ называется перечислимой сверху, если существуют вычислимая функция $F(x,k)$, определённая для всех слов x и всех натуральных чисел k, для которой $F(x,0) \geq F(x,1) \geq F(x,2) \geq ...$ и $f(x) = \lim_{k\to\infty} F(x,k)$ для каждого значения x. Причём при любом k значение $F(x,k)$ является верхней оценкой для $f(x)$.*

Функция $f(x)$ называется перечислимой снизу, если существует аналогичная нижняя оценка $L(x,k)$: $L(x,0) \leq L(x,1) \leq L(x,2) \leq ...$ и $f(x) = \lim_{k\to\infty} L(x,k)$ для каждого значения x.

Теорема 7.2. *Функция KS перечислима сверху, причём $|\{x : KS(x) < n\}| < 2^n$ для любого n.*

Доказательство. Покажем, что множество пар

$$\{\langle n, x\rangle : KS(x) < n\},$$

где n – натуральное число, а x – двоичное слово, перечислимо[29].

Если $KS(x) < n$, то существует фигурирующая в определении KS вычислимая функция – декомпрессор D. Используя установленный стандартный порядок двоичных слов, можно организовать вычисления, перебирая слова p по мере роста их длины, *соблюдая условие* $KS(x) < n$. В начале перечисления положим $k = 0$.

Будут перебираться все слова, длина которых не превышает n. Как только окажется, что $D(p) = x$ (при этом будет иметь место $KS(x) = l(p)$, причём должно соблюдаться условие $l(p) < n$), перечисляющий алгоритм будет выдавать пару $\langle l(p) + k, x\rangle$ и увеличивать k на единицу. Если первая выдача будет парой $\langle l(p)+0, x\rangle$, при этом первое значение n будет равно $l(p) + 1$, то выдаваемая перечисляющая последовательность будет иметь вид $\langle l(p)+0, x\rangle, \langle l(p)+1, x\rangle, \langle l(p)+2, x\rangle...$. Таким алгоритмом будут порождаться все пары множества $\{\langle n, x\rangle : KS(x) < n\}$ для любого слова x.

Поскольку для каждого n перебираются все слова длины не больше n, то всего таких различных слов будет

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1 < 2^n.$$

[29]Множество называется перечислимым, если существует алгоритм, который выдаёт в произвольном порядке все элементы этого множества и только их

Поэтому $|\{x : KS(x) < n\}| < 2^n$.

Укажем теперь функцию $F(x,k) = l(p) + n - k$ как последовательность оценок сверху сложности $KS(x)$, доопределив $F(k,x) = \infty$ при $k > n$. Тогда $F(x,0) \geq F(x,1) \geq F(x,2) \geq ...$ и $KS(x) = \lim_{k\to\infty} F(x,k)$, поскольку это предельное соотношение соответствует неравенству $k > n$ для любого заданного n. Действительно,

$$KS(x) = \begin{cases} \min_p l(p) & : D(p) = x; \\ \infty, & \text{если такого слова } p \text{ не существует,} \end{cases}$$

и условия определения 7.4 для функции

$$F(x,k) = l(p) + n - k \tag{63}$$

выполняются, поскольку в формуле (63) по построению $l(p)$ – минимальная длина, и случай $F(x,k) < l(p)$ соответствует неопределённости (∞) колмогоровской сложности $KS(x)$.

Лемма 7.1. *Лемма Для любой вычислимой функции $f(x)$ имеет место неравенство $KS(f(x)) \leq KS(x) + O(1)$ для всех тех значений x, когда $f(x)$ определена.*

Доказательство. Пусть D – оптимальный декомпрессор в определении $KS(x) = KS_D(x) = \min\{l(p) : D(p) = x\}$. Возьмём в качестве другого декомпрессора композицию вычислимых функций $f \circ D$ и рассмотрим

$$KS_{f\circ D}(f(x)) = \min\{l(p) : f(D(p)) = f(x)\}$$
$$= \min\{l(p) : D(p) = x\} = KS(x).$$

Для оптимального декомпрессора

$$KS(f(x)) \leq KS_{f\circ D}(f(x)) + O(1).$$

Учитывая, что $KS_{f\circ D}(f(x)) = KS(x)$, окончательно имеем

$$KS(f(x)) \leq KS(x) + O(1).$$

Теорема 7.3. *Любая частично рекурсивная (вычислимая) функция $L(x)$ такая, что $L(x) \leq KS(x)$ в тех точках, в которых $L(x)$ определена, ограничена некоторой константой C , то есть $L(x) \leq C$ для всех x.*

Доказательство. Предположим, что существует вычислимая функция $L(x)$, являющаяся оценкой снизу колмогоровской сложности: $L(x) \leq KS(x)$. Определим функцию $A(n)$, которая ставит в соответствие натуральному числу n минимальное в порядке "$\prec$" перечисления строк значение x такое, что $L(x) \geq n$:

$$A(n) = \overset{\prec}{\min}\ x : L(x) \geq n.$$

Функция $A(n)$ будет вычислимой в силу предположения, что $L(x)$ вычислима. Предположим, что вопреки утверждению теоремы функция $L = L(x)$ не ограничена. Тогда функция $A = A(n)$ определена для всех n, и

$$L(A(n)) \geq n$$

по построению для всех n.

В то же время, согласно сделанному предположению, что $L(x) \leq KS(x)$, имеем неравенство

$$L(A(n)) \leq KS(A(n)).$$

Согласно лемме 7.1,

$$KS(A(n)) \leq KS(n) + c_1.$$

Получается цепочка неравенств:

$$n \leq L(A(n)) \leq KS(A(n)) \leq KS(n) + c_1 \leq \log n + c_2,$$

которые должны выполняться для всех n; где c_1 и c_2 - некоторые константы. Но следующее из этой цепочки неравенство

$$n \leq \log n + c_2$$

не выполняется для всех n, больших некоторого значения n_0 . Полученное противоречие доказывает теорему.

Теорема 7.4. *Колмогоровская сложность KS не является вычислимой функцией.*

Доказательство. Предположив, что KS вычислима (для любой строки x), получим, что вычислима функция $f(x) = KS(x) - 1$, и тогда $f(x) \leq KS(x)$ для всех непустых строк x. Но такой нижней оценки для колмогоровской сложности не существует согласно теореме 7.3.

Действительно, с ростом строки x колмогоровская сложность $KS(x)$ может оказаться сколь угодно большой, но тогда и функция $f(x) = KS(x) - 1$ обязана иметь возможность оказаться сколь угодно большой, что противоречит доказанной ограниченности любой нижней оценки константой.

Теорема 7.5. *Значение $l(p)$ такое, что $D(p) = x$ для произвольной строки x и оптимального декомпрессора D в определении колмогоровской сложности*

$$KS_D(x) = \min\{l(p) | D(p) = x\}$$

определено тогда и только тогда, когда существует машина Тьюринга T_C (компрессор) такая, что $T_C(x) = p$.

Доказательство. Действительно, если существует машина Тьюринга D такая, что $D(p) = x$, то существует система подстановок Маркова M_D, алгоритмически эквивалентная МТ D (реализующая тот же самый алфавитный оператор). Применение M_D к слову p даст $x = M_D(p)$. Зафиксируем выполненную при этом последовательность марковских подстановок:

$$\tilde{S}(M_D, p, x) = \{s_1, ..., s_j, ..., s_\mu : s_j = \lambda_j \to \rho_j\},$$

где λ_j - левая часть подстановки (замещаемое подслово), а ρ_j - правая часть подстановки (замещающее подслово), вместе с последовательностью $k_1, ..., k_j, ..., k_\mu$ номеров символов текущего обрабатываемого слова, начиная с которых реализуются подстановки. Тогда компрессор T_C может быть композицией машин Тьюринга двух типов: подвода головки к символу с номером k_j (обозначим эти машины T_j^1) и заменой подслова ρ_j на подслово λ_j (обозначим их T_j^2). Применение к слову x последовательно машин $T_\mu^1, T_\mu^2, ..., T_j^1, T_j^2, .., T_1^1, T_1^2$ даёт композицию T_C такую, что $T_C(x) = p$ (машина T_1^2 должна быть снабжена заключительным состоянием).

Аналогично доказывается, что если для строки x существует машина Тьюринга T_C (компрессор) такая, что $T_C(x) = p$, где p – некоторая строка, то можно указать соответствующую ей машину-декомпрессор D_{T_C} такую, что $D_{T_C}(p) = x$, и тогда величина $l(p)$ такая, что $D(p) = x$ будет определена вследствие теоремы 7.1 Соломонова-Колмогорова. □

Определение 7.5. *Компрессор T_C называется оптимальным, если $T_C(x) = KS(x)$.*

Теорема 7.6. *Задача построения оптимального компрессора является алгоритмически неразрешимой.*

Доказательство. Действительно, разрешимость задачи построения оптимального компрессора повлекла бы вычислимость колмогоровской сложности, что противоречит теореме 7.4 о её невычислимости. □

Теорема 7.5 даёт обоснование применения алгоритмического подхода к получению (программированию) верхней оценки колмогоровской сложности. Для этой цели необходимо построить алгоритм, реализующий подходящий компрессор для оцениваемой строки x. Однако доказать, что этот компрессор является оптимальным, не удастся.

7.3 Колмогоровский подход к определению количества информации

Определение 7.6. (*А. Н. Колмогоров*) *Количество информации в слове y о слове x есть*

$$I(y : x) = KS(x) - KS(x|y).$$

Величину $KS(x)$ можно интерпретировать как минимальное количество информации, необходимое для алгоритмического восстановления слова x; величина $KS(x|y)$ интерпретируется как минимальное количество информации, которое необходимо добавить к информации, содержащейся в y, чтобы восстановить x.

Можно заметить, что определение 7.6 аналогично шенноновско-

му вероятностному определению количества информации

$$I_S(A:B) = I(A,B) = H(B) - H_A(B),$$

где A и B – источники (ансамбли) сообщений, а $H(B)$ и $H_A(B)$ безусловная и условная шенноновские энтропии.

Для описания свойств колмогоровской энтропии будем использовать следующие обозначения.

$F(\tilde{x}) \preceq G(\tilde{x})$ означает, что найдется такая константа C, что для любого допустимого набора значений переменных $\tilde{x}$ выполняется неравенство $F(\tilde{x}) \leq G(\tilde{x}) + C$.

Аналогично вводится отношение "$\succeq$".

$F \asymp G$ имеет место тогда и только тогда, когда $F \preceq G$ и $G \preceq F$.

Теорема 7.7.

1) $I(y:x) \succeq 0$.

2) $|I(x:x) - KS(x)| \asymp 0$.

Доказательство.

1) Убедимся, что $I(x:y) \succeq 0$.

Пусть D – оптимальный декомпрессор – частично рекурсивная функция одного аргумента, а D_2 – такой декомпрессор, что $D_2(p,y) = D(p)$. Тогда, если $D(p) = x$ и $KS(x) = p$, то, поскольку $D_2(p,y) = x$, имеем

$$KS(x|y) \preceq KS_{D_2}(x|y) = KS(x);$$

$$KS(x|y) \preceq KS(x);$$

$$I(y:x) = KS(x) - KS(x|y) \ \succeq \ 0.$$

2) Теперь докажем соотношение $|I(x:x) - KS(x)| \asymp 0$.

Пусть $D_2(p,x) = x$. Тогда должно выполняться $D_2(\Lambda, x) = x$ и $KS_{D_2}(x|x) = 0$. Поэтому

$$KS(x|x) \preceq KS_{D_2}(x|x) = 0;$$

$$KS(x|x) \preceq 0;$$

$$I(x:x) = KS(x) - KS(x|x);$$

$$I(x:x) - KS(x) = -KS(x|x);$$

$$0 \leq |I(x:x) - KS(x)| = KS(x|x) \preceq 0.$$

$$0 \leq |I(x:x) - KS(x)| \preceq 0;$$

$$|I(x:x) - KS(x)| \asymp 0;$$

Замечание. Доказанное соотношение $|I(x:x) - KS(x)| \asymp 0$ может вызвать недоумение: представляется, что должно выполняться $I(x:x) = KS(x)$, если ориентироваться на соотношение (59), которое содержится во введении 7.1. Но соотношение (59) было использовано для предварительного представления идеи колмогоровской сложности относительно какого-нибудь зафиксированного способа описания. В отличие от (59), равенство

$$|I(x:x) - KS(x)| \asymp 0$$

с точностью до константы является "платой" за использование в определении колмогоровской сложности $KS(x)$ оптимального декомпрессора.

8 Приложение А

8.1 Префиксные коды и неравенство Крафта-Макмиллана

Пусть имеется некоторый алфавит $A = \{a_1, ..., a_i, ..., a_r\}$, и каждой букве a_i этого алфавита поставлено во взаимно однозначное соответствие двоичное слово B_i, называемое *элементарным кодом*. Это соответствие называют *схемой кодирования* $\sum$, которую определяют следующим образом:

$$\sum : \begin{cases} a_1 \to B_1; \\ a_2 \to B_2; \\ ..... \\ a_r \to B_r. \end{cases}$$

Алгоритм преобразования любого слова $a_{j_1}a_{j_2}\dots a_{j_t}\dots a_{j_l}$ длины l в слово $B_{j_1}B_{j_2}\dots B_{j_t}\dots B_{j_l}$ путём замены каждой из букв $a_{j_t} \in A$ на соответствующий элементарный код $B_{j_t} \in \{B_1, B_2, \dots, B_r\}$ согласно схеме $\sum$ называют *алфавитным кодированием*.

Множество всевозможных слов вида $B_{j_1}B_{j_2}\dots B_{j_t}\dots B_{j_l}$, которые могут быть получены в результате кодирования по схеме $\sum$, называются *кодом* или *языком, порождённым схемой кодирования*.

Алгоритм алфавитного кодирования реализует словарную функцию

$$\mathscr{C} : S(A) \to S(H),$$

где $S(A)$ – множество слов над алфавитом A, а $S(H) = S(\{0, 1\})$ – множество слов над двухсимвольным алфавитом $\{0, 1\}$.

В общем случае алфавитное кодирование $\mathscr{C} : S(A) \to S(H)$ может основано на отображении слов алфавита A в слова не обязательно двузначного, а произвольного конечного алфавита H. Здесь изложен случай двоичного алфавитного кодирования, но почти все результаты легко переносятся на произвольный алфавит кодирования H.

Кодирование и соответствующий код называются *допускающими однозначное декодирование* или просто *однозначными*, если най-

дётся алгоритм $\mathscr{D}$, который по любому слову $\tilde{b} = \mathscr{C}(\tilde{a}) \in H$ однозначно определит слово $\tilde{a} = \mathscr{D}(\tilde{b}) \in A$.

Пусть произвольное слово $\tilde{\alpha}$ является конкатенацией трёх слов

$$\tilde{\alpha} = \tilde{\upsilon}\tilde{\tau}\tilde{\nu}.$$

Тогда слово $\tilde{\upsilon}$ называют *префиксом*, а слово $\tilde{\nu}$ – окончанием слова $\tilde{\alpha}$. Префикс и окончание в некоторых случаях можно считать пустыми, что соответствует пустому слову.

Определение 8.1. *Говорят, что алфавитное кодирование по заданной схеме Σ и соответствующий ему код является префиксным, если ни один элементарный код из $B_1, B_2, ..., B_r$ не является префиксом другого.*

Префиксное бинарное кодирование удобно описывать с помощью бинарных деревьев, в которых каждая левая ветвь, исходящая из вершины, кодируется нулём, а правая – единицей. Тогда ветвь дерева любой длины, заканчивающаяся листом, который помечен некоторым символом a_i алфавита A, "прочитывается" от корневой вершины дерева до этого листа как последовательность нулей и единиц, определяющая элементарный код B_i.

Теорема 8.1. *Кодирование и соответствующий код является префиксным тогда и только тогда, когда элементарные коды схемы алфавитного кодирования соответствуют различным ветвям некоторого конечного бинарного дерева.*

Доказательство. *Необходимость.* Пусть $\mathscr{L}$ – префиксный код, заданный схемой Σ. Тогда все элементарные коды $B_1, B_2, ..., B_r$ попарно различны и ни один из них не является префиксом другого. Отсортируем элементарные коды по невозрастанию их длин и возьмём первым самый длинный из элементарных двоичных кодов. Пусть это будет код $B_m = b_1^1 b_2^1 \dots b_{k_1}^1$. Зададим первую ветвь дерева, определённую двоичными значениями $b_1^1 b_2^1 \dots b_{k_1}^1$ и закончим её концевой вершиной с пометкой B_m. Возьмём следующий по упорядочению код. Пусть это код $B_j = b_1^2 b_2^2 \dots b_{k_j}^2$. Поскольку $B_m \neq B_j$ и ни один из этих кодов не является префиксом другого, то ветвь $b_1^2 b_2^2 \dots b_{k_j}^2$ не совпадёт с ветвью $b_1^1 b_2^1 \dots b_{k_1}^1$ и не будет её собственной частью. Тогда сравнивая слова B_m и B_j посимвольно слева направо, можно найти двоичное значение, различное в этих словах. Согласно

этому различию построим разветвление в конструируемом дереве (см. рис. 8.49) и вторую ветвь, заканчивающуюся концевой вершиной с меткой B_j.

Продолжая этот процесс до тех пор, пока не будут построены ветви дерева, соответствующие всем элементарным кодам, всякий раз находя в дереве ветвь, начальный отрезок которой совпадает с начальным отрезком очередного элементарного кода и аналогичным образом добавляя разветвление, получим конечное кодовое дерево.

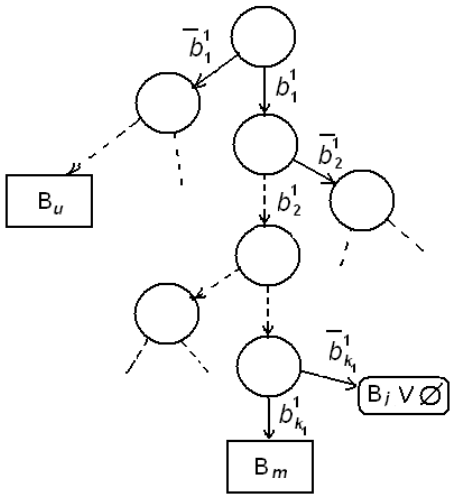


Рис. 8.49

Достаточность. Действительно, если бы код не был префиксным, то нашлась бы пара элементарных кодов, в которой один элементарный код был бы префиксом другого. Тогда меньший по длине код "лежал" бы на той же ветви, что и больший по длине код из этой пары кодов. Но тогда бы нарушилось условие принадлежности элементарных кодов различным ветвям дерева.

Теорема 8.2. *Если кодирование по схеме Σ является префиксным, то оно допускает однозначное декодирование.*

Доказательство. Предположим противное: пусть найдётся слово $\tilde{b} = b_1, b_2, ..., b_q$, полученное по схеме кодирования Σ, допускающее хотя бы две разные расшифровки – два разных разбиения на элементарные коды. На рис. 8.50 показаны два разных разбие-

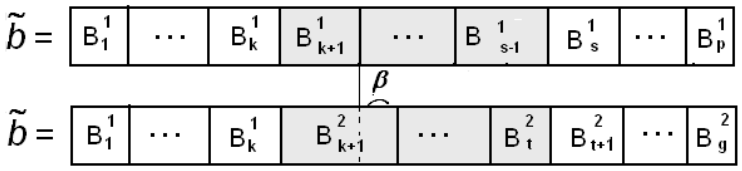


Рис. 8.50

ния слова $\tilde{b}$ на элементарные коды. Запись B_j^i означает $j-$й по порядку код в $i-$м варианте разбиения. Различие разбиений в общем

случае будет иметь место только в некоторой (затонированной на рисунке) части кода, и тогда будет совпадение префиксов и окончаний: $B_1^1 = B_1^2, ..., B_k^1 = B_k^2$; $B_s^1 = B_{t+1}^2, ..., B_p^1 = B_g^2$. Но тогда $B_{k+1}^1 \neq B_{k+1}^2$, причём элементарный код B_{k+1}^1 будет префиксом элементарного кода $B_{k+1}^2 = B_{k+1}^1\beta$, и в таком случае кодирование не является префиксным.

Теорема 8.3 (Л. Крафт)**.** *Для того, чтобы существовал префиксный код с длинами кодовых слов $l_1, l_2, ..., l_r$, необходимо и достаточно, чтобы выполнялось неравенство*

$$\sum_{i=1}^{r} 2^{-l_i} \leq 1.$$

Доказательство. *Необходимость.* Пусть произвольный префиксный код имеет длины кодовых слов $l_1, l_2, ..., l_r$. Поскольку код – префиксный, все кодовые двоичные слова попарно различны.

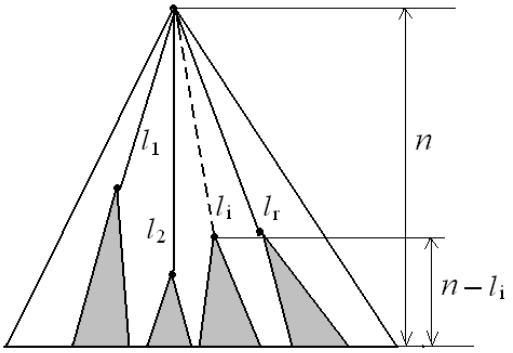


Рис. 8.51

Рассмотрим полное двоичное дерево зафиксированной высоты n, где $n > \max\{l_1, l_2, ..., l_r\}$.

Каждому двоичному кодовому слову $\tilde{b}_i = b_1^i, b_2^i, \ldots, b_{l_i}^i$ соответствует ветвь дерева длины l_i, причём эти "кодовые" ветви попарно различны, поскольку ни одно кодовое слово не является префиксом другого.

Последний символ $b_{l_i}^i$ $i-$той ветви можно рассмотреть как вершину поддерева, имеющего высоту $n - l_i$ (рис. 8.51). Все ветви этого поддерева будут определять продолжения двоичного слова $\tilde{b}_i$. Заштрихованные поддеревья для всех $\tilde{b}_i$, $i = \overline{1, r}$ не пресекаются, поскольку пересечение привело бы к циклу, что противоречит определению дерева.

Концевые вершины заштрихованных поддеревьев входят во множество всех концевых вершин полного бинарного дерева высоты n, число которых равно 2^n. Каждое поддерево содержит 2^{n-l_i} вершин.

Поэтому сумма $\sum_{i=1}^{r} 2^{n-l_i}$ не может превысить 2^n, но может оказаться равной 2^n, если все коды имеют равную длину. Действительно, если все коды имеют равную длину l, и для кодирования используются все $r = 2^l$ кодов, то $\sum_{i=1}^{r} 2^{n-l} = r2^{n-l} = 2^l 2^{n-l} = 2^n$.

Достаточность. Возьмём числовой отрезок единичной длины и будем последовательно разбивать его на равные части. Левой части очередного разбиения будем приписывать ноль, а правой – единицу, как показано на рис. 8.52. Тогда полное дерево высоты n будет соответствовать разбиению отрезка длины 1 на 2^n промежутков длины 2^{-n}. А если дерево не будет полным, то любой его ветви длины $l < n$ будет соответствовать промежуток разбиения единичного отрезка длины 2^{-l}.

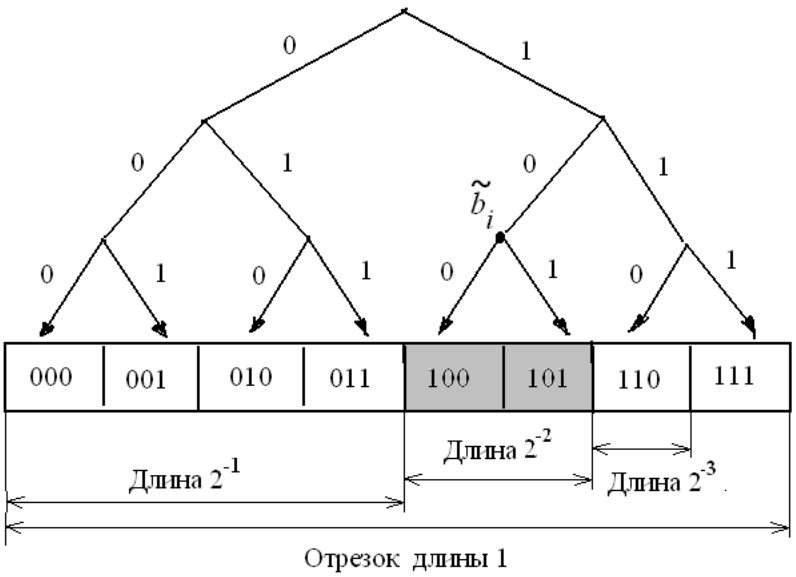


Рис. 8.52: К доказательству достаточности

Пусть для целых положительных чисел $l_1, l_2, ..., l_r$ выполняется неравенство

$$\sum_{i=1}^{r} 2^{-l_i} \leq 1.$$

Тогда в отрезке длины 1 можно выделить r непересекающихся промежутков, имеющих длины $2^{-l_1}, 2^{-l_2}, \ldots, 2^{-l_r}$. Обозначим $n = \max\{l_1, l_2, ..., l_r\}$. Тогда каждому из указанных отрезков и, соответственно, каждому числу l_i, можно сопоставить выделенную в полном

дереве высоты n ветвь $\tilde{b}_i$ длины l_i, получая двоичный код этой ветви – кодовое слово $\tilde{b}_i$. Выделенные таким образом ветви не пересекаются, и ни одна из них не является префиксом другой. Следовательно при выполнении неравенства существует префиксный код с длинами кодовых слов $l_1, l_2, ..., l_r$.

Теорема 8.4 (Б. Макмиллан)**.** *Для существования однозначно декодируемого кода с длинами кодовых слов $l_1, l_2, ..., l_r$ необходимо и достаточно, чтобы выполнялось неравенство*

$$\sum_{i=1}^{r} 2^{-l_i} \leq 1. \tag{64}$$

Доказательство. *Достаточность* непосредственно вытекает из теоремы 8.3, поскольку из условия (64) следует свойство префиксности кода, а из свойства префиксности – однозначность декодирования.

Необходимость. Рассмотрим произвольный однозначно декодируемый код с длинами двоичных кодовых слов $l_1, l_2, ..., l_r$. Обозначим кодовые слова (элементарные коды) $\omega_1, \omega_2, ..., \omega_r$. Запишем формальную сумму

$$(\omega_1 + \omega_2 + ... + \omega_r)^m,$$

где m – целое положительное число, и раскроем скобки, полагая, что вместо произведения слов используется их конкатенация, т.е. непосредственное приписывание справа к первому слову второго. Тогда

$$(\omega_1 + \omega_2 + \cdots + \omega_r)^m = W_1 + W_2 + \cdots + W_t + \cdots + W_{r^m}, \tag{65}$$

причём все слова $W_1, W_2, \ldots, W_{r^l}$ попарно различны, поскольку рассматриваемый код является однозначно декодируемым.

Заменим в выражении (65) каждый символ в каждом слове на число $\frac{1}{2}$, и тогда все слова как в левой, так и в правой части выражения превратятся в произведения. В левой части получим

$$(2^{-l_1} + 2^{-l_2} + \cdots + 2^{-l_r})^m,$$

а в правой –

$$2^{-k_1} + 2^{-k_2} + \cdots + 2^{-k_t} + \cdots + 2^{-k_{r^m}},$$

где k_t – длина слова W_t.

Следствием этой замены будет равенство

$$(2^{-l_1}+2^{-l_2}+\cdots+2^{-l_r})^m = 2^{-k_1}+2^{-k_2}+\cdots+2^{-k_t}+\cdots+2^{-k_{r^m}}. \quad (66)$$

Заметим, что для любого t количество слов длины t не превысит 2^t, так как все слова в правой части (65) различны. Самое длинное слово из $W_1, W_2, \ldots, W_{r^l}$ будет иметь длину

$$\max\{l_1, l_2, ..., l_r\} \cdot m.$$

Сгруппировав все слова одинаковой длины t (пусть их окажется μ_t) получим сумму в каждой группе равную

$$\mu_t 2^{-t} \leq 2^t 2^{-t} = 1.$$

Число различных длин не превысит $\max\{l_1, l_2, ..., l_r\} \cdot m$, поэтому

$$2^{-k_1}+2^{-k_2}+\cdots+2^{-k_t}+\cdots+2^{-k_{r^m}} \leq \max\{l_1, l_2, ..., l_r\} \cdot m.$$

Если предположить, что $2^{-l_1}+2^{-l_2}+\cdots+2^{-l_r} > 1$, то левая часть выражения (66) окажется растущей экспоненциально, а правая – не быстрее чем линейно по m. Это противоречит равенству левой и правой частей выражения (66). Поэтому

$$2^{-l_1}+2^{-l_2}+\cdots+2^{-l_r} \leq 1.$$

8.2 Код Хаффмана

Кодируемые схемой $\sum : \{a_i \to B_i|_{i=1}^r$ символы a_i алфавита A могут иметь различные вероятности в исходных текстовых сообщениях. Например, буква "Ы" в сообщениях на русском языке встречается значительно реже, чем буква "М". Естественно строить алфавитный код так, чтобы часто передаваемые символы имели как можно более короткие элементарные коды по сравнению с редко передаваемыми.

Пусть вероятности появления символов исходного алфавита a_i равны p_i, $\sum_{i=1}^r p_i = 1$, а длины элементарных кодов равны $l_i =$

$len(B_i)$. Средняя длина элементарного кода, зависящая от заданной схемы кодирования Σ, определяется по формуле

$$\bar{l}(\Sigma) = \sum_{i=1}^{r} l_i p_i = 1.$$

Коды, определяемые такими схемами кодирования Σ^*, что

$$\bar{l}(\Sigma^*) = \min_{\Sigma} \bar{l}(\Sigma),$$

т. е. схемами кодирования, обеспечивающими минимальную среднюю длину элементарного кода по сравнению со всеми другими схемами, *обеспечивающими однозначность декодирования*, называются *кодами с минимальной избыточностью или кодами Хаффмана.*

Легко убедиться, что для любого кода с минимальной избыточностью для любых двух символов исходного алфавита a_i и a_j таких, что $p_i < p_j$ должно выполняться условие $l_j \geq l_i$.

Действительно, если предположить противное, – что для некоторого кода с минимальной избыточностью и средней длиной $\bar{l}$ найдутся два символа a_i и a_j такие, что $p_i < p_j$, но $l_j > l_i$, то поменяв местами элементарные коды B_i и B_j можно получить схему кодирования с меньшей средней длиной $\bar{l}'$:

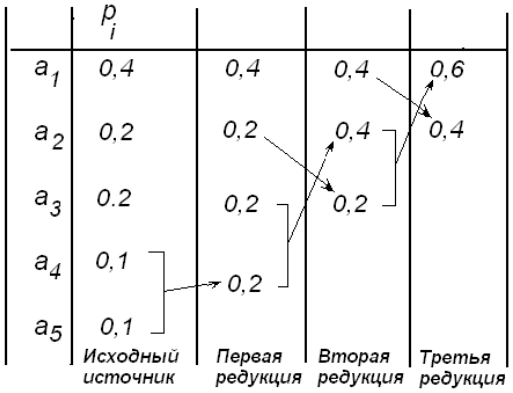


Рис. 8.53: Шаги редукции при построении кода Хаффмана

$$\bar{l} - \bar{l}' = p_i l_i + p_j l_j - (p_i l_j + p_j l_i) = \underbrace{(p_i - p_j)}_{<0}\underbrace{(l_i - l_j)}_{<0} > 0,$$

что противоречит безызбыточности исходного кода.

Рассмотрим случай построения бинарного дерева, определяющего префиксный двоичный кода Хаффмана с минимальной избыточностью.

Первый шаг. Два символа исходного алфавита a_k и a_m, имеющие наименьшие вероятности p_k и p_m, кодируются двоичными словами (элементарными кодами) одинаковой длины и различаются одним двоичным символом. Два этих закодированных символа можно условно заменить одним символом с суммарной вероятностью $p_k + p_m$. Тогда задача сводится к построению кода для источника информации, имеющего алфавит, который меньше на один символ чем исходный алфавит.

Последующие шаги логически повторяют первый шаг, но уже для укороченного, редуцированного источника (сокращаемого шаг за шагом алфавита сообщений).

Пример выполнения шагов редукции приведен на рис. 8.53.

После завершения всех шагов редукции (когда будет построен код для двухбуквенной схемы) в обратном направлении (относительно стрелок на рис. 8.53) "прочитываются" ветви бинарного дерева кодирования. В этом примере: $a_1 - 1$; $a_2 - 01$; $a_3 - 001$; $a_4 - 0001$; $a_5 - 0000$.

Подробнее о кодах Хаффмана см. [12, с. 276 - 288].

9 Приложение В: Доказательство теоремы Шеннона о блоковом кодировании

Лемма 9.1. *Для любых целых неотрицательных чисел $l_1, l_2, ..., l_k$ таких, что $l_1 + l_2 + ... + l_k = m$ выполняется неравенство*

$$\frac{m!}{l_1!l_2! \cdots l_k!} \leq 2^{mH(\frac{l_1}{m}, \frac{l_2}{m}, ..., \frac{l_k}{m})}.$$

Доказательство. Введем следующие обозначения:

$$C_m^{l_1,l_2,...,l_k} = \frac{m!}{l_1!l_2! \cdots l_k!};$$

$$D_m^{l_1,l_2,...,l_k} = \frac{m^m}{l_1^{l_1} l_2^{l_2} \cdots l_k^{l_k}}.$$

Докажем неравенство:

$$C_m^{l_1,l_2,...,l_k} \le D_m^{l_1,l_2,...,l_k},$$

используя метод математической индукции.

*Базис индукции.*При $m = 1$ имеем $l_i = 1$ или 0, $i = 1, ..., k$. Очевидно,

$$C_1^{l_1,l_2,...,l_k} = D_1^{l_1,l_2,...,l_k} = 1.$$

Индуктивный переход. Пусть

$$C_{m-1}^{l_1,l_2,...,l_{k-1},l_k-1} \le D_{m-1}^{l_1,l_2,...,l_{k-1},l_k-1}$$

для любого разбиения произвольного целого числа $m-1$, большего единицы, на k слагаемых $l_1, l_2, ..., l_k - 1$ (при разбиении числа $m-1$ по сравнению с числом m какое-либо ненулевое число, не теряя общности l_k, станет на единицу меньше, т.е. $l_k - 1$).

Рассмотрим отношение

$$\frac{D_m^{l_1,l_2,...,l_k}}{D_{m-1}^{l_1,l_2,...,l_k-1}} = \frac{m^m \cdot l_1^{l_1} l_2^{l_2} \cdots (l_k-1)^{l_k-1}}{(m-1)^{m-1} \cdot l_1^{l_1} l_2^{l_2} \cdots l_k^{l_k}}$$

$$\frac{D_m^{l_1,l_2,...,l_k}}{D_{m-1}^{l_1,l_2,...,l_k-1}} = \frac{m \cdot m^{m-1} \cdot (l_k-1)^{l_k-1}}{(m-1)^{m-1} \cdot l_k^{l_k}} = \frac{m \cdot (1+\frac{1}{m-1})^{m-1}}{l_k \cdot (1+\frac{1}{l_k-1})^{l_k-1}}.$$

С учётом того, что функция $(1+\frac{1}{n})^n$ монотонно возрастает с ростом n, и $m \ge l_k$, получаем

$$\frac{D_m^{l_1,l_2,...,l_k}}{D_{m-1}^{l_1,l_2,...,l_k-1}} \ge \frac{m}{l_k}.$$

Поскольку

$$\frac{C_m^{l_1,l_2,...,l_{k-1},l_k}}{C_{m-1}^{l_1,l_2,...,l_{k-1},l_k-1}} = \frac{m}{l_k},$$

получаем

$$\frac{D_m^{l_1,l_2,...,l_{k-1},l_k}}{D_{m-1}^{l_1,l_2,...,l_{k-1},l_k-1}} \ge \frac{C_m^{l_1,l_2,...,l_{k-1},l_k}}{C_{m-1}^{l_1,l_2,...,l_{k-1},l_k-1}},$$

$$D_m^{l_1,l_2,...,l_k} \geq C_m^{l_1,l_2,...,l_k} \cdot \frac{D_{m-1}^{l_1,l_2,...,l_{k-1},l_k-1}}{C_{m-1}^{l_1,l_2,...,l_{k-1},l_k-1}}.$$

По предположению индукции второй сомножитель в правой части последнего неравенства больше или равен единице, поэтому

$$D_m^{l_1,l_2,...,l_k} \geq C_m^{l_1,l_2,...,l_k}$$

$$\log D_m^{l_1,l_2,...,l_k} = m\log m - \sum_{i=1}^{k} l_i \log l_i = -\sum_{i=1}^{k} l_i(\log l_i - \log m) =$$

$$-m\sum_{i=1}^{k} \frac{l_i}{m} \log \frac{l_i}{m};$$

$$D_m^{l_1,l_2,...,l_k} = 2^{mH(\frac{l_1}{m},...,\frac{l_k}{m})},$$

$$C_m^{l_1,l_2,...,l_k} \leq D_m^{l_1,l_2,...,l_k},$$

$$C_m^{l_1,l_2,...,l_k} \leq 2^{mH(\frac{l_1}{m},...,\frac{l_k}{m})}.$$

В частности,

$$C_m^{l} \leq 2^{mH(\frac{l}{m})}.$$

Лемма 9.2. (О числе линейных кодов). *Существует ровно* $2^{k(n-k)}$ *линейных* n*-разрядных кодов с* k *информационными и* $n-k$ *контрольными двоичными разрядами.*

Доказательство. В линейных (n,k)-кодах из n разрядов k – информационных, а $(n-k)$ – контрольных. Каждый контрольный разряд имеет определяемый кодом номер, но в рамках данного доказательства можно, не теряя общности, перенумеровать эти контрольные разряды, обозначая их $\beta_i, \quad i = 1, ..., (n-k)$. Каждый контрольный разряд вычисляется как сумма некоторых информационных разрядов по модулю 2. Если обозначить информационные разряды $\alpha_j, \quad j = 1, ..., k$, введя и для них новую отдельную нумерацию, то можно записать эту сумму в виде

$$\beta_i = \bigoplus_{j \in J_i} \alpha_j,$$

где J_i – множество номеров информационных разрядов, входящих в сумму. Чтобы выделить эти номера из всех k номеров информационных разрядов, составим вектор

$$\tilde{r_i} = (r_{i_1}, r_{i_2}, ..., r_{i_k})$$

такой, что $r_{i,j} = 1$, если $j \in J_i$, иначе $r_{i,j} = 0$. Тогда

$$\beta_i = \bigoplus_{j=1,...,k} r_{i,j}\alpha_j = (\tilde{r}_i, \tilde{\alpha}),$$

$$\begin{bmatrix} \beta_1 \\ ... \\ \beta_i \\ ... \\ \beta_{n-k} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & ... & r_{1k} \\ ... & ... & ... & ... \\ r_{i1} & r_{i2} & ... & r_{ik} \\ ... & ... & ... & ... \\ r_{n-k,1} & r_{n-k,2} & ... & r_{n-k,k} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ ... \\ \alpha_j \\ ... \\ \alpha_k \end{bmatrix}.$$

Легко увидеть, что всевозможное кодирование контрольных разрядов исчерпывается числом $2^{k(n-k)}$ возможных значений матрицы $[r_{ij}]_{(n-k)\times k}$, равным искомому числу двоичных линейных (n,k)-кодов.

Лемма 9.3. *Заданный фиксированный n-разрядный набор содержится ровно в $2^{(n-k)(k-1)}$ линейных (n,k)-кодах.*

Доказательство. Подсчитаем число кодов из множества всех линейных (n,k)-кодов, содержащих фиксированный двоичный набор $\tilde{\gamma}$ длины n, в который входят k информационных разрядов, обозначаемых далее $\tilde{\alpha}$ (информационное слово), и $(n-k)$ контрольных разрядов, обозначаемых $\tilde{\beta}$ (контрольное слово). Несмотря на то, что контрольные и информационные разряды *чередуются в наборе* $\tilde{\gamma}$, можно условно считать их идущими подряд, полагая $\tilde{\gamma} = \tilde{\beta}\tilde{\alpha}$.

Любому информационному слову $\tilde{\alpha}$ в каждом линейном коде соответствует единственное контрольное слово, поскольку $\tilde{\beta}$ однозначно вычисляется по $\tilde{\alpha}$:

$$\tilde{\beta} = [r_{ij}]_{(n-k,k)} \times \tilde{\alpha}$$

(см. лемму 9.2), и у каждого кода имеется своя индивидуальная матрица $[r_{ij}]_{(n-k,k)}$. Меняя такие матрицы, каждому набору $\tilde{\alpha}$ можно

поставить в соответствие 2^{n-k} различных контрольных слов $\tilde{\beta}$ по всем линейным (n,k)-кодам.

Набор $\tilde{\gamma} = \tilde{\beta}\tilde{\alpha}$ будет одним из 2^{n-k} двоичных наборов длины n, содержащих заданное информационное слово $\tilde{\alpha}$ и входящих в различные линейные коды. Поэтому все множество $2^{k(n-k)}$ линейных кодов разбивается на 2^{n-k} групп, содержащих различные информационные слова $\tilde{\alpha}$. Следовательно, число линейных кодов, содержащих фиксированный набор – слово $\tilde{\gamma}$, будет равно

$$\frac{2^{k(n-k)}}{2^{n-k}} = 2^{k(n-k)-(n-k)} = 2^{(n-k)(k-1)}.$$

Теорема 9.1 (К. Шеннон)**.** *Для любого числа R, меньшего пропускной способности канала C, и любого $\varepsilon > 0$ существует способ блоковой передачи со скоростью не меньшей R и вероятностью ошибки, не превосходящей ε.*

Доказательство.[30] Рассмотрим передачу линейного кода длины n. Обозначим k – ближайшее к nR целое число, большее или равное nR, т.е. $\lceil nR \rceil$. Для упрощения доказательства примем $k = nR$. Пусть k – число информационных разрядов линейного кода длины n. Рассмотрим множество таких (n,k) кодов и покажем, что средняя по этому множеству вероятность ошибки стремится к нулю при $n \to \infty$. Поскольку минимальная вероятность ошибки для кодов этого множества не больше средней, то этим будет доказано существование кодов со сколь угодно малой вероятностью ошибки.

Согласно используемым обозначениям, скорость передачи равна $R = k/n$. Пусть $\mathcal{C}$ – некоторый линейный (n,k) код. Обозначим

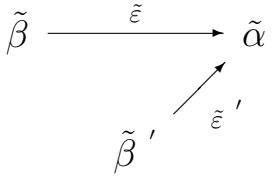
$$p_{\mathcal{C}}(E) = \sum_{\tilde{\varepsilon}} p(\tilde{\varepsilon}) \cdot p_{\mathcal{C}}(E/\tilde{\varepsilon})$$

вероятность ошибки передачи при использовании кода $\mathcal{C}$, где $\tilde{\varepsilon} = (\varepsilon_1, ..., \varepsilon_n)$, а $p(\tilde{\varepsilon})$ – вероятность того, что в канале при передаче произошли ошибки в соответствующих разрядах двоичного блока данных длины n; $p_{\mathcal{C}}(E/\tilde{\varepsilon})$ – вероятность того, что блок после передачи

[30] Приведенное здесь доказательство заимствовано из книги Л. А. Шоломова [11]

по каналу после декодирования будет принят с ошибкой при условии ошибки $\tilde{\varepsilon}$ в канале.

Будем полагать, что декодирование происходит по ближайшему кодовому вектору из $\mathcal{C}$. Принятый двоичный набор $\tilde{\alpha} = \tilde{\beta} \oplus \tilde{\varepsilon}$ искажает кодовый набор $\tilde{\beta}$ и приводит к ошибке лишь в случае существования другого кодового набора $\tilde{\beta}'$ и вектора ошибки $\tilde{\varepsilon}'$ таких, что $\tilde{\beta} \oplus \tilde{\varepsilon} = \tilde{\alpha} = \tilde{\beta}' \oplus \tilde{\varepsilon}'$ и при этом $\|\tilde{\varepsilon}'\| \leq \|\tilde{\varepsilon}\|$, где $\|\tilde{x}\| = \sum_{i=1}^{n} x_i$ – норма двоичного вектора $\tilde{x}$.



Но если $\tilde{\beta} \oplus \tilde{\varepsilon} = \tilde{\beta}' \oplus \tilde{\varepsilon}'$, то $\tilde{\beta} \oplus \tilde{\beta}' = \tilde{\varepsilon} \oplus \tilde{\varepsilon}'$, и в силу линейности кода $\mathcal{C}$ комбинация $\tilde{\varepsilon} \oplus \tilde{\varepsilon}'$ также принадлежит коду $\mathcal{C}$. Таким образом, ошибка возникает только тогда, когда найдется искажение $\tilde{\varepsilon} \neq \tilde{\varepsilon}'$ такое, что

$$\begin{cases} \tilde{\varepsilon} \oplus \tilde{\varepsilon}' \in \mathcal{C}; \\ \|\tilde{\varepsilon}'\| \leq \|\tilde{\varepsilon}\|. \end{cases}$$

Пусть $N_{\mathcal{C}}(\tilde{\varepsilon})$ – число всех таких векторов $\tilde{\varepsilon}'$. Тогда будет выполняться неравенство $p_{\mathcal{C}}(E/\tilde{\varepsilon}) \leq N_{\mathcal{C}}(\tilde{\varepsilon})$. Действительно, если $N_{\mathcal{C}}(\tilde{\varepsilon}) = 0$, то $p_{\mathcal{C}}(E/\tilde{\varepsilon}) = 0$, поскольку ошибки в этом случае нет. Иначе $N_{\mathcal{C}}(\tilde{\varepsilon}) \geq 1$ и неравенство $p_{\mathcal{C}}(E/\tilde{\varepsilon}) \leq N_{\mathcal{C}}(\tilde{\varepsilon})$ снова (тривиально) выполняется.

Пусть теперь m некоторое число такое, что $m \leq \frac{n}{2}$. Представим $p_{\mathcal{C}}(E)$ в виде

$$p_{\mathcal{C}}(E) = \sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|\leq m} p(\tilde{\varepsilon}) \cdot \overbrace{p_{\mathcal{C}}(E/\tilde{\varepsilon})}^{\leq N_{\mathcal{C}}(\tilde{\varepsilon})} + \sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|>m} p(\tilde{\varepsilon}) \cdot \overbrace{p_{\mathcal{C}}(E/\tilde{\varepsilon})}^{\leq 1},$$

$$p_{\mathcal{C}}(E) \leq \sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|\leq m} p(\tilde{\varepsilon}) \cdot N_{\mathcal{C}}(\tilde{\varepsilon}) + \sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|>m} p(\tilde{\varepsilon}).$$

Согласно лемме 9.2, число линейных (n, k)-кодов есть $2^{k(n-k)}$. Средняя вероятность ошибки по всем таким линейным кодам есть

$$\bar{p}(E) = \frac{1}{2^{k(n-k)}} \sum_{\mathcal{C}} p_{\mathcal{C}}(E) \leq$$

$$\leq \frac{1}{2^{k(n-k)}} \sum_{\mathcal{C}} \sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|\leq m} p(\tilde{\varepsilon}) N_{\mathcal{C}}(\tilde{\varepsilon}) + \sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|>m} p(\tilde{\varepsilon})$$

(вторая сумма не зависит от $\mathcal{C}$ и при усреднении не изменяется).

$$\bar{p}(E) \leq \frac{1}{2^{k(n-k)}} \sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|\leq m} p(\tilde{\varepsilon}) \sum_{\mathcal{C}} N_{\mathcal{C}}(E) + \sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|>m} p(\tilde{\varepsilon}).$$

Оценим сверху сумму $\sum_{\mathcal{C}} N_{\mathcal{C}}(\tilde{\varepsilon})$ при фиксированном $\tilde{\varepsilon}$. Пусть $\|\tilde{\varepsilon}\| = j$. Рассмотрим произвольный набор $\tilde{\varepsilon}\,' \neq \tilde{\varepsilon}$ такой, что $\|\tilde{\varepsilon}\,'\| = i \leq j$. Ненулевой набор $\tilde{\varepsilon}\,' \oplus \tilde{\varepsilon}$ принадлежит не более чем $2^{(k-1)(n-1)}$ различным кодам (см. лемму 9.3), поэтому вектор $\tilde{\varepsilon}\,'$ даёт вклад в сумму $\sum_{\mathcal{C}} N_{\mathcal{C}}(\tilde{\varepsilon})$, не превосходящий $2^{(k-1)(n-1)}$. Общий вклад в эту сумму от всех векторов веса i (их ровно C_n^i) не превышает величины $C_n^i 2^{(k-1)(n-1)}$. С учётом того, что $i \leq j$, получается оценка

$$\sum_{\mathcal{C}} N_{\mathcal{C}}(\tilde{\varepsilon}) \leq \sum_{i\leq j=\|\tilde{\varepsilon}\|} C_n^i 2^{(k-1)(n-1)},$$

где $i = \|\tilde{\varepsilon}\,'\|$.

Вероятность ошибки $p(\tilde{\varepsilon})$ веса j есть $\mathfrak{p}^j \mathfrak{q}^{n-j}$, где $\mathfrak{p}$ – вероятность ошибки в канале, а $\mathfrak{q} = 1 - \mathfrak{p}$.

$$\sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|\leq m} p(\tilde{\varepsilon}) \sum_{\mathcal{C}} N_{\mathcal{C}}(\tilde{\varepsilon}) \leq \sum_{j\leq m} C_n^j \mathfrak{p}^j \mathfrak{q}^{n-j} \sum_{i\leq j} C_n^i 2^{(k-1)(n-1)}.$$

При $i \leq j \leq m \leq \frac{n}{2}$ и $i \leq \frac{1}{2}$ величина C_n^i с ростом i возрастает, поэтому $C_n^i \leq C_n^j$. С учётом того, что $j + 1 \leq m + 1 \leq n$, имеем

$$\sum_{i\leq j} C_n^i \leq (j+1) C_n^j \leq n C_n^j$$

и получаем оценку

$$\sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|\leq m} p(\tilde{\varepsilon}) \sum_{\mathcal{C}} N_{\mathcal{C}}(\tilde{\varepsilon}) \leq n 2^{(k-1)(n-1)} \sum_{j\leq m} (C_n^j)^2 \mathfrak{p}^j \mathfrak{q}^{n-j}.$$

$$\sum_{\tilde{\varepsilon}:\|\tilde{\varepsilon}\|>m} p(\tilde{\varepsilon}) = \sum_{j>m} C_n^j \mathfrak{p}^j \mathfrak{q}^{n-j} \leq \sum_{j\geq m} C_n^j \mathfrak{p}^j \mathfrak{q}^{n-j}.$$

$$p(\bar{E}) \le \frac{n}{2^{n-k}} \sum_{j \le m} (C_n^j)^2 \mathfrak{p}^j \mathfrak{q}^{n-j} + \sum_{j \ge m} C_n^j \mathfrak{p}^j \mathfrak{q}^{n-j}.$$

Выберем параметр m так, чтобы в первой сумме максимальным было последнее слагаемое, а во второй – первое. Для этого найдем отношение j-го члена первой суммы к $(j-1)$-му:

$$\frac{(C_n^j)^2 \mathfrak{p}^j \mathfrak{q}^{n-j}}{(C_n^{j-1})^2 \mathfrak{p}^{j-1} \mathfrak{q}^{n-j+1}} = \frac{(n-j+1)^2 \mathfrak{p}}{j^2 \mathfrak{q}} \ge \frac{(n-j)^2 \mathfrak{p}}{j^2 \mathfrak{q}}$$

Если потребовать, чтобы выполнялось неравенство

$$\frac{(n-j)^2 \mathfrak{p}}{j^2 \mathfrak{q}} \ge 1,$$

то это приведет к условиям $(n-j)\sqrt{\mathfrak{p}} \ge j\sqrt{\mathfrak{q}}$ и $j \le n\frac{\sqrt{\mathfrak{p}}}{\sqrt{\mathfrak{p}}+\sqrt{\mathfrak{q}}}$; и тогда

$$m \le n\frac{\sqrt{\mathfrak{p}}}{\sqrt{\mathfrak{p}} + \sqrt{\mathfrak{q}}}.$$

Таким же образом, взяв отношение $(j+1)$-го члена второй суммы к j-му, получим

$$\frac{C_n^{j+1} \mathfrak{p}^{j+1} \mathfrak{q}^{n-j-1}}{C_n^j \mathfrak{p}^j \mathfrak{q}^{n-j}} = \frac{(n-j)\mathfrak{p}}{(j+1)\mathfrak{q}} \ge \frac{(n-j)\mathfrak{p}}{j\mathfrak{q}}.$$

Если потребовать, чтобы величина $\frac{(n-j)\mathfrak{p}}{j\mathfrak{q}}$ не превосходила единицы, получим $j \ge n\frac{\mathfrak{p}}{\mathfrak{p}+\mathfrak{q}} = n\mathfrak{p}$, откуда имеем второе ограничение для m в виде $m \ge n\mathfrak{p}$. Таким образом, получаем

$$n\mathfrak{p} \le m \le n\frac{\sqrt{\mathfrak{p}}}{\sqrt{\mathfrak{p}} + \sqrt{\mathfrak{q}}};$$

$$n\mathfrak{p} \le m \le n\frac{\mathfrak{p}}{\mathfrak{p} + \sqrt{\mathfrak{p}\mathfrak{q}}}.$$

Учитывая, что $\mathfrak{p} < \frac{1}{2}$, откуда следует, что $\mathfrak{p} < \mathfrak{q}$, получим

$$\frac{\mathfrak{p}}{\mathfrak{p} + \sqrt{\mathfrak{p}\mathfrak{q}}} > \frac{\mathfrak{p}}{\mathfrak{p} + \mathfrak{q}} = \mathfrak{p},$$

что подтверждает совместность двойного неравенства для m. Кроме этого, $m \le \frac{n}{2}$, поскольку $\mathfrak{p} < \mathfrak{q}$. Используя полученные оценки, имеем

$$\bar{p}(E) \le \frac{n}{2^{n-k}}(m+1)(C_n^m)^2\mathfrak{p}^m\mathfrak{q}^{n-m} + (n-m+1)C_n^m\mathfrak{p}^m\mathfrak{q}^{n-m} \le$$

$$\le \frac{n^2}{2^{n-k}}(C_n^m)^2\mathfrak{p}^m\mathfrak{q}^{n-m} + nC_n^m\mathfrak{p}^m\mathfrak{q}^{n-m} =$$

$$= \Big(\frac{n^2}{2^{n-k}}C_n^m + n\Big)C_n^m\mathfrak{p}^m\mathfrak{q}^{n-m}.$$

По условию теоремы

$$R - C = R - (1 - H(\mathfrak{p})) < 0.$$

Выберем константу ρ, удовлетворяющую ограничениям

$$\mathfrak{p} < \rho < \frac{\sqrt{\mathfrak{p}}}{\sqrt{\mathfrak{p}} + \sqrt{\mathfrak{q}}},$$

настолько близкой к $\mathfrak{p}$, чтобы выполнялось условие

$$R - 1 + H(\rho) < 0.$$

Положим $m = n\rho$, пренебрегая тем, что величина $n\rho$ может не быть целой, и считая, что берется ближайшее целое к $n\rho$. Используя лемму 9.1, получаем

$$\bar{p}(E) \le \Big(\frac{n^2}{2^{n-k}}2^{nH(\rho)} + n\Big)2^{nH(\rho)}\mathfrak{p}^{n\rho}\mathfrak{q}^{n(1-\rho)}.$$

$$\frac{2^{nH(\rho)}}{2^{n-k}} = 2^{n(H(\rho)-1+k/n)} = 2^{n(H(\rho)-1+R)} < 1,$$

поскольку $R - 1 + H(\rho) < 0$ (напомним, что $k = nR$). Поэтому

$$\bar{p}(E) \le (n^2 + n)2^{nH(\rho)}\mathfrak{p}^{n\rho}\mathfrak{q}^{n(1-\rho)} \le 2n^2 2^{nH(\rho)}\mathfrak{p}^{n\rho}\mathfrak{q}^{n(1-\rho)}.$$

Логарифмирование даёт

$$\log \bar{p}(E) \le \log(2n^2) + n(H(\rho) - (-\mathfrak{p}\log\rho - \mathfrak{q}\log(1-\rho))).$$

Согласно свойству энтропии из неравенства $\mathfrak{p} \neq \rho$ следует, что величина

$$H(\rho) - (-\mathfrak{p}\log\rho - \mathfrak{q}\log(1-\rho))$$

меньше нуля. Обозначим эту величину $-\Delta$, и тогда Δ будет являться некоторой положительной константой. С учётом этого обозначения

$$\log\bar{p}(E) \leq 1 + 2\log n - n\Delta = n(-\Delta + o(1)).$$

Тогда для всех достаточно больших n найдется положительная константа $\Delta_0 < \Delta$ такая, что $\log\bar{p}(E) \leq -n\Delta_0$ и

$$\bar{p}(E) \leq 2^{-n\Delta_0}.$$

Из последнего неравенства следует, что найдется (n,k)-код $\mathcal{C}$ такой, что

$$p_{\mathcal{C}}(E) \leq \bar{p}(E) \leq 2^{-n\Delta_0}.$$

И тогда, взяв достаточно большое n, можно получить вероятность ошибки $p_{\mathcal{C}}(E)$ меньше заданного числа ε. $\square$

10 Приложение C: Универсальные частично рекурсивные функции

Класс частично рекурсивных функций совпадает с классом функций, представимых на машинах Тьюринга (этот класс также называют классом *вычислимых функций*). Было доказано, что эти функции можно пронумеровать (шифры машин Тьюринга), и тогда каждая вычислимая функция будет иметь хотя бы один номер – натуральное число.

Вообще говоря, одна и та же функция может быть представлена разными машинами Тьюринга или разными рекурсивными формулами. Например,

$$Sq(x\dot{-}y)\cdot x + \bar{S}q(x\dot{-}y)\cdot y = max\{x,y\},$$

поэтому обозначив

$$f_1(x,y) = Sq(x\dot{-}y)\cdot x + \bar{S}q(x\dot{-}y)\cdot y,$$

$$f_2(x, y) = max\{x, y\},$$

получим две конструктивно различные формулы, задающие одно и то же отображение – вычислимую функцию. Очевидно, что можно построить как минимум две соответствующие МТ, реализующие эту функцию.

Далее будем говорить именно о вычислимых отображениях, а не об их представлениях в том или ином эквивалентном виде, т.е. рассматривать вычислимые функции (машины Тьюринга) с точностью до эквивалентности. Для этого будем использовать минимальные номера из множества номеров эквивалентных формул (машин Тьюринга) в перечислении номеров всех вычислимых функций, получая последовательность натуральных чисел

$$\varphi_{n_1}, \varphi_{n_2}, ..., \varphi_{n_k}, \dots ,$$

и называть эти номера номерами функций.

На основе приведенных рассуждений можно предположить, что существует универсальная вычислимая функция U такая, что для любой вычислимой функции $\varphi(x)$ одного аргумента

$$U(n, x) = \varphi(x)$$

во всех точках, где $\varphi(x)$ определена, если n – номер этой функции в указанном выше перечислении.

Тогда можно представить машину Тьюринга M_U, реализующую универсальную функцию U, которая работает следующим образом. МТ M_U сначала расшифровывает число n и преобразовывает его в шифр МТ M_φ, которая эквивалентна функции φ, а затем интерпретирует команды машины M_φ так, как происходила бы обработка входного слова x машиной M_φ.

Теорема 10.1. *Существует универсальная частично рекурсивная функция двух переменных $U = U(n, x)$ такая, что для любой частично рекурсивной функции $\varphi = \varphi(x)$ одной переменной выполняется равенство*

$$U(n, x) = \varphi(x)$$

для всех x, когда значение $\varphi(x)$ определено.

Доказательство. Пусть $\varphi = \varphi(x)$ – произвольная частично рекурсивная функция. Тогда она может быть реализована на эквивалентной ей машине Тьюринга M_n, для определённости – с наименьшим номером n (шифром) из всех эквивалентных этой функции машин.

Теперь укажем в явном виде другую машину Тьюринга – M_U, которая получает на входе два аргумента – число n и строку x. Строка x является значением аргумента функции $\varphi(x)$, представленного подходящим образом. Число n – шифр МТ M_n, эквивалентной функции φ. Машина Тьюринга M_U преобразует номер n в шифр машины M_n, а "запускает" эту машину M_n путём интерпретации её шифра на слове x. Тогда, если значение $\varphi(x)$ определено, то интерпретируемая МТ M_n выдаст на ленту значение $\varphi(x)$ и остановится. В противном случае, если на аргументе x функция φ не определена, то МТ никогда не остановится. Очевидно, такая машина M_U эквивалентна функции φ.

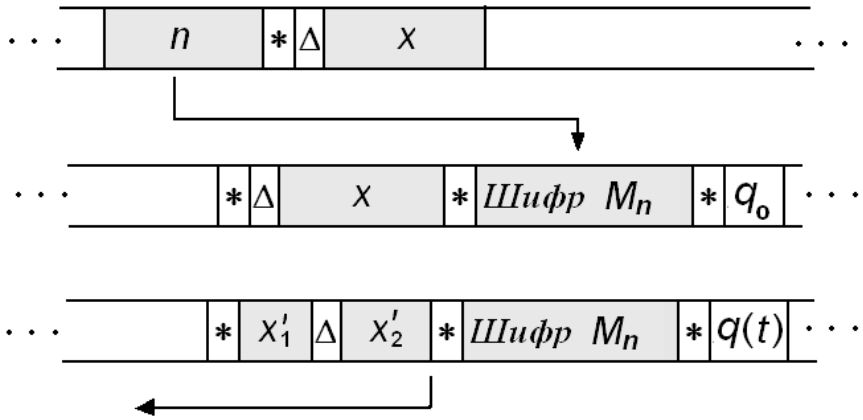


Рис. 10.54: К доказательству теоремы 10.1

МТ M_U представляет собой композицию нескольких машин Тьюринга, работа которых описывается следующим образом (рис. 10.54).

Машина M_1 по начальной информации $n\ *\ \Delta\ x$, где $*$ – разделитель, а Δ – маркер-указатель обрабатываемого символа, расшифровывает число n, преобразуя его в шифр МТ M_n. Это происходит путём последовательной генерации двоичных кодов в порядке 0, 1, 00, 01, 10, 11, 000, ... в отрезке ленты "Шифр M_n" с проверкой: яв-

ляется ли очередной код шифром МТ. Первый же код, оказавшийся по своей конструкции шифром и равный двоичному числу n, будет искомым шифром машины M_n. Шифры распознаются согласно правилам кодирования команд МТ по пятеркам компонент каждой команды. Получив шифр машины M_n, машина M_1 записывает начальное состояние q_0 машины M_n справа за её шифром.

Машина M_2 отыскивает подходящую команду в поле шифра машины M_n, начиная с состояния q_0 и первого слева входного символа символа $x(1)$ заданного слова $x = x(1)x(2)\dots x(r)$. Новое состояние записывается в отрезок ленты "q(t)".

Машина M_3 модифицирует входное слово согласно очередной команде, перемещает маркер-указатель Δ к нужной позиции и при необходимости расширения сегмента текущего состояния $x^{'}$ обрабатываемого слова x осуществляет такое расширение путём сдвига обрабатываемой строки.

Машина M_4 стирает все блоки кроме слова – результата. $\square$

Если $U = U(n, x)$ – универсальная рекурсивная функция, которая существует согласно доказанной теореме, то выполнив подстановку вместо x частично рекурсивной биекции p от пары аргументов (x_1, x_2), можно получить универсальную функцию трёх аргументов, используя равенство

$$U^*(n, x_1, x_2) = U(n, p(x_1, x_2)).$$

Действительно, пусть $\varphi(x_1, x_2)$ – любая частично рекурсивная функция двух аргументов, а p – частично рекурсивная биекция

$$\eta \overset{p}{\longleftrightarrow} (x_1, x_2): \quad p(x_1, x_2) = \eta;\ x_1 = l(\eta);\ x_2 = r(\eta).$$

Из функции $\varphi(x_1, x_2)$ можно получить частично рекурсивную функцию $h(\eta) = \varphi(l(\eta), r(\eta))$, для которой найдется номер n_h, не зависящий от аргументов x_1, x_2 и такой, что

$$h(\eta) = U(n_h, \eta).$$

Тогда

$$\varphi(x_1, x_2) = h(\eta) = h(p(x_1, x_2)) = U(n_h, p(x_1, x_2)) = U_2(n_h, x_1, x_2),$$

где U_2 – *универсальная функция трёх переменных такая, что для любой частично рекурсивной функции* $\varphi(x_1, x_2)$ *двух переменных выполняется равенство*

$$U_2(n, x_1, x_2) = \varphi(x_1, x_2).$$

Аналогичным образом можно показать, что *для любого числа* k *переменных существует универсальная функция такая, что для любой частично рекурсивной функции* $\varphi(x_1, x_2, \ldots, x_k)$ *выполняется равенство*

$$U_k(n, x_1, x_2, \ldots, x_k) = \varphi(x_1, x_2, \ldots, x_k).$$

Предметный указатель

Список литературы

[1] Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов.— М.: Мир, 1979. — 536 с.

[2] Верещагин Н.К., Шень А. Лекции по математической логике и теории алгоритмов. Часть 2. Языки и исчисления. — 4-е изд., испр. — М.: МЦНМО,2012. — 240 с.

[3] Винер Н. Кибернетика, или управление и связь в животном и машине. / Пер. с англ. И.В. Соловьева и Г.Н. Поварова; Под ред. Г.Н. Поварова. – 2-е издание. — М.: Наука, Главная редакция изданий для зарубежных стран, 1983. — 344 с.

[4] Журавлёв Ю.И. Дискретный анализ. Формальные системы и алгоритмы: Учебное пособие / Ю. И.Журавлёв, Ю. А. Флёров, М. Н. Вялый. — М: ООО КонтактПлюс, 2010. — 336 с.

[5] Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи.— М.: Мир, 1982. — 416 с.

[6] Колмогоров А. Н. Теория информации и теория алгоритмов. — М.: Наука, 1984. — 320 с.

[7] Линдон Р. Заметки по логике. — М.: Мир, 1968. — 128 с.

[8] Мендельсон Э. Введение в математическую логику. — М.: Наука, 1987. — 304 с.

[9] Смирнов В. И. Курс высшей математики. Том 2. — М.: Наука, 1974. — 656 с.

[10] Шеннон К. Работы по теории информации и кибернетике. — М.: Издательство иностранной литературы, 1963. — 830 с.

[11] Шоломов Л. А. Основы теории дискретных логических и вычислительных устройств. — М.: Наука, 1980. — 400 с.

[12] Яблонский С. В. Введение в дискретную математику. — М.: Наука, 1986. — 384 с.