МЕТОДЫ СОВРЕМЕННОЙ МАТЕМАТИКИ

ВВЕДЕНИЕ В СЛОЖНОСТЬ ВЫЧИСЛЕНИЙ

В. Н. Крупский

Выпуск 2

МЕТОДЫ СОВРЕМЕННОЙ МАТЕМАТИКИ

Выпуск 2

В. Н. Крупский

ВВЕДЕНИЕ В СЛОЖНОСТЬ ВЫЧИСЛЕНИЙ

Москва
ФАКТОРИАЛ ПРЕСС
2006

УДК 519.7
ББК 22.18
К 84

Крупский В. Н.

К 84 Введение в сложность вычислений/В. Н. Крупский — М.: Факториал Пресс, 2006. — 128 с. — (Методы современной математики; Вып. 2).

ISBN 5-88688-083-6

Учебное пособие написано по материалам полугодового спецкурса, читавшегося автором на механико-математическом факультете МГУ им. М. В. Ломоносова для студентов и аспирантов кафедры математической логики и теории алгоритмов, а также специальности «Защита информации». Излагаются основные идеи и методы теории сложности вычислений.

Для студентов, аспирантов и специалистов, занимающихся анализом эффективности алгоритмов.

УДК 519.7
ББК 22.18

Серия

МЕТОДЫ СОВРЕМЕННОЙ МАТЕМАТИКИ

Выпуск 2

Научное издание

Владимир Николаевич Крупский

ВВЕДЕНИЕ В СЛОЖНОСТЬ ВЫЧИСЛЕНИЙ

Формат 60 × 90/16. Усл. печ. л. 8. Бумага офсетная №1. Гарнитура литературная. Подписано к печати 25.04.2006. Тираж 1000 экз. Заказ № 3428.

Издательство «Факториал Пресс», 117449, Москва, а/я 331; ЛР ИД № 00316 от 22.10.99. e-mail: factorialco@mail.ru

Отпечатано с готовых диапозитивов издательства «Факториал Пресс» в ППП типографии «Наука» Академиздатцентра «Наука» РАН. 121099, Москва Г-99, Шубинский пер., 6.

ISBN 5-88688-083-6

ОГЛАВЛЕНИЕ

Часть I

МОДЕЛИ ВЫЧИСЛЕНИЙ

ГЛАВА 1

МАШИНЫ ТЬЮРИНГА

§ 1. Неформальное введение

Неформальное понятие *модели вычислений* предполагает задание *языка программирования* вместе с его операционной *семантикой*, т. е. объяснением того, каким образом программе сопоставляется *процесс вычислений*.

Процесс вычислений, развиваясь в дискретном времени, потребляет различные (разнородные) *ресурсы*. Предполагается, что величина потребленного к *данному* моменту ресурса измеряется натуральными числами и не может быть бесконечной, т. е. бесконечный ресурс требуется только для никогда не заканчивающегося процесса. Удобно, хотя не столь существенно, также считать, что каждый никогда не заканчивающийся процесс потребляет бесконечное количество каждого ресурса:

ресурс $< \infty \Leftrightarrow$ процесс вычисления заканчивается.

Обычно модель вычислений задает способ подсчета использованных ресурсов в каждом конечном вычислении. Это означает, что

- функция $Compl : (program, input) \mapsto resourse$ — вычислима,

- отношение $Compl(p, i) < n$ — разрешимо.

Основные (но не единственные) ресурсы — время и память. *Но для каждой модели вычислений они свои!*

Как используют модель вычислений для оценки трудоемкости алгоритмов? Хорошо реализуют алгоритм в виде программы p на языке программирования данной модели, выбирают ресурс r (из числа определенных в модели) и ищут верхние оценки $f(n)$ на величину затрат. Например, в худшем случае:

$$\max_{size(i)=n} Compl_r(p, i) < f(n).$$

Основное противоречие: точные оценки такого рода интересны для моделей вычислений с реалистичными языками программирования, для которых получение каких-нибудь теоретических оценок очень трудно.

Выходы:

- Численный эксперимент машинно зависим, т. е. зависит от «железа», а потому результаты быстро устаревают.

- Огрубления в определении ресурса. Например, в качестве времени для C — подсчет только количества выполняемых арифметических операций (или только сравнений).

- Построение честных оценок для более простой «игрушечной» модели и учет замедления при компиляции в реальный язык.

Пример. Пусть «игрушечная» модель вычисления M допускает компилятор программ в C

$$compiler : p_M \mapsto p_C,$$

про который *доказано*, что

$$Time_C(p_C, i) < (Time_M(p_M, i))^5 + 1000.$$

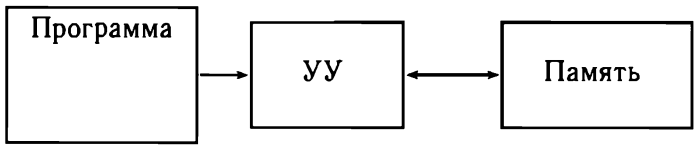
Если мы установим, что временная сложность некоторой задачи в M есть $O(n^2)$, то получим верхнюю оценку $O(n^{10})$ в C. Это загрубленная, но честная оценка.

Как относиться к таким оценкам? Конечно, показатель степени весьма неточен, но то, что время есть степенная функция, а не экспонента — абсолютно точно! Значит, использовать этот подход разумно тогда, когда важна разница между многочленом и экспонентой, а не между n^2 и n^3. Тогда и теорему про компилятор можно доказывать не в самой сильной форме (например, не уточняя показатель степени), когда она в большинстве случаев оказывается очевидной.

Наблюдение. Для реальных универсальных языков программирования и большинства универсальных «игрушечных» (не специально испорченных) языков компиляция одного в другой замедляет по времени не более, чем полиномиально, и увеличивает потребность к памяти не более, чем линейно. Именно на это положение дел следует рассчитывать в приложениях теории сложности и внутри самой теории.

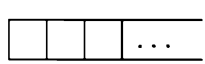
§ 2. Модели Тьюринга

Это семейство моделей вычислений наиболее честно отражает время вычислений. Возможных вариантов определения много. Машина Тьюринга состоит из управляющего устройства (УУ) и потенциально бесконечной внешней памяти, структура которой не меняется со временем. Она снабжена программой, задающей правила ее функционирования.

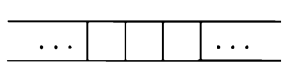


Память разбита на одинаковые ячейки, предназначенные для хранения букв фиксированного конечного алфавита (одна буква в ячейке). Имеется одна или несколько (конечное число) читающе-пишущих головок. В каждый момент времени головка обозревает одну ячейку памяти. За один такт работы головка может изменить содержимое этой ячейки и переместиться к одной из соседних (или остаться на месте). Функционированием головок управляет УУ. В каждый момент времени УУ находится в одном из фиксированного конечного множества внутренних состояний. Один такт работы состоит в следующем: машина читает содержимое обозреваемых ячеек и внутреннее состояние, выбирает из программы инструкцию, однозначно определяемую этими данными, и исполняет ее. В инструкции сказано, каким должно стать новое внутреннее состояние, какие буквы должны быть записаны в обозреваемые ячейки и куда должны переместиться головки.

Варианты определения различаются геометрией памяти. Это существенно, так как за один шаг головкам разрешается переместиться только в соседние ячейки. В простейших моделях память представляет собой ленту с одной головкой, бесконечную в одну



или обе стороны



В теории сложности обычно используют многоленточные машины Тьюринга — с конечным набором лент и одной го-

ловкой на каждой из них. Рассматривались также машины Тьюринга с «многомерной» памятью (например, в виде клетчатой плоскости) и различные «многоглавые» варианты (несколько головок на одной связной компоненте памяти). Можно показать, что все эти варианты моделируют друг друга с полиномиальным (квадратичным) замедлением по времени и линейным увеличением памяти.

Вычислительные возможности машин Тьюринга достаточно хорошо изучены. Их качественная характеризация состоит в описании класса всех функций, вычислимых на машинах Тьюринга. Речь идет о частичных словарных функциях $f : \Sigma^* \to \Sigma^*$, где Σ^* — множество всех слов в конечном алфавите Σ, а термин «частичная функция» означает, что значение функции $f(v)$ для некоторых слов $v \in \Sigma^*$ может оказаться неопределенным. Результат сравнительного изучения запасов вычислимых словарных функций для различных (не только «игрушечных») моделей вычислений может быть сформулирован в виде следующего неформального утверждения:

> **Тезис Тьюринга (неформальный):** Каждую вычислимую (программой какого угодно языка программирования) частичную функцию типа $\Sigma^* \to \Sigma^*$ можно вычислить на подходящей машине Тьюринга.

Неформальность Тезиса Тьюринга состоит в том, что он не может быть полностью обоснован математическими средствами (доказан как математическая теорема). В то же время все многочисленные попытки его опровергнуть, т. е. предложить язык программирования с большими вычислительными возможностями, оказались безуспешными, в результате чего в настоящее время подобные проекты считаются абсолютно бесперспективными. Причина невозможности полного математического обосно-

вания кроется в словах «какого угодно языка программирования» — они не определяют ни самого семейства языков, о которых идет речь, ни какого-либо свойства этого семейства. Если их заменить на достаточно информативное описание семейства языков программирования (которое необходимо окажется менее общим), то соответствующий частный случай Тезиса станет обычным, «поддающимся доказательству» математическим утверждением.

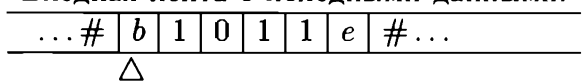
Пример. Пусть семейство состоит из языков, C и PASCAL, заданных полным описанием их синтаксиса и операционной семантики. Оба языка обладают компиляторами в ASSEMBLER, поэтому для обоснования соответствующего частного случая Тезиса Тьюринга достаточно построить компилятор, преобразующий ассемблерный код в программу для машины Тьюринга. Последнее является весьма трудоемкой, но абсолютно реалистичной задачей по программированию. Для аккуратного математического доказательства потребуется еще верифицировать все три компилятора, т. е. доказать, что они работают правильно.

§ 3. Многоленточные машины Тьюринга

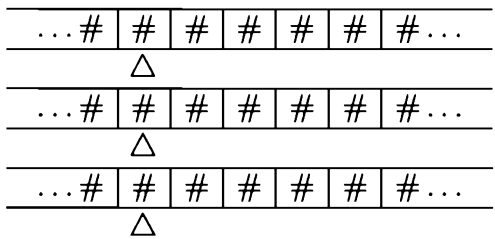
Ленты бесконечны в обе стороны. Пустые ячейки несут символ #. Есть одна или несколько входных лент (только для чтения), одна или несколько рабочих лент (можно читать и писать), некоторые из которых можно объявить выходными. Входной алфавит Σ_0 содержится в ленточном Σ и не содержит #. Считываемый с выходной ленты результат есть то слово в алфавите Σ_0, на котором или непосредственно перед которым остановилась головка. Удобно также требовать, чтобы на входных лентах головка не удалялась от границ входного слова наружу: например, можно ввести ограничительные буквы b и e для обозначения

границ входного слова, за которые головке вылезать запрещается. Начальная конфигурация лент выглядит так:

Входная лента с исходными данными:



Рабочие ленты:



Работа машины описывается тремя конечными функциями (k — количество лент, Q — фиксированное заранее конечное множество состояний УУ):

$\alpha : Q \times (\Sigma^*)^k \to Q$ — новое состояние
$\beta : Q \times (\Sigma^*)^k \to (\Sigma^*)^k$ — новое содержимое ячеек
$\gamma : Q \times (\Sigma^*)^k \to \{N, L, R\}^k$ — движения головок

Их удобно записывать в виде программы — набора непротиворечивых команд вида:

$$q\bar{a} \mapsto \alpha(q, \bar{a})\beta(q, \bar{a})\gamma(q, \bar{a})$$

Непротиворечивость набора означает, что в нем нет двух команд с одинаковой левой частью $q\bar{a}$. Для сокращения письма предполагают также, что если команда с левой частью $q\bar{a}$ не выписана явно, то она все-таки есть в программе, но имеет специфический вид $q\bar{a} \mapsto q\bar{a}N \ldots N$. Такие команды тривиальны, так как не вызывают никаких действий.

Пример. На рис. 1.1 приведена программа машины Тьюринга с одной входной и одной рабочей лентой, которая переписывает исходное двоичное слово в обратном порядке.

Полная спецификация многоленточной машины Тьюринга есть набор

$$\langle Q, \Sigma, \alpha, \beta, \gamma, q_{\text{нач.}}, q_{\text{закл.}} \rangle.$$

Если к ней добавлен входной алфавит $\Sigma_0 \subset \Sigma \setminus \{\#, b, e\}$, выделены входные ($k_0$ штук) и выходные (l_0 штук) ленты и α в самом деле не меняет содержимое входных лент, то получаем вычислитель частичной функции $f : (\Sigma_0^*)^{k_0} \to \to (\Sigma_0^*)^{l_0}$. Для вычисления значения функции — вектора $(w_1, \ldots, w_{l_0}) = f(v_1, \ldots, v_{k_0})$ надо записать на входных лентах слова bv_ie, запустить машину и дождаться остановки, после чего считать с j-й выходной ленты то слово в алфавите Σ_0, на которое указывает головка. Это w_j. Если головка остановилась не на букве алфавита Σ_0, то результат — пустое слово. Если машина работает бесконечно долго, то значение $f(v_1, \ldots, v_{k_0})$ не определено.

Замечание. Конечное множество состояний на самом деле может представлять любую ограниченную для данной машины структуру данных. Это имитирует статически распределенную память с возможностью за один шаг переприсваивать ее всю целиком. Единственное ограничение — новое состояние определяется однозначно по предыдущему состоянию и содержимому обозреваемых ячеек ленты.

Программа:

$$
\begin{array}{llll}
q_0\,b\# & \mapsto & q_0\,b\#\,RN & q_1\,0\# \mapsto q_1\,00\,LR \\
q_0\,0\# & \mapsto & q_0\,0\#\,RN & q_1\,1\# \mapsto q_1\,11\,LR \\
q_0\,1\# & \mapsto & q_0\,1\#\,RN & q_1\,b\# \mapsto q_2\,b\#\,NN \\
q_0\,e\# & \mapsto & q_1\,e\#\,LN &
\end{array}
$$

Функционирует она так:

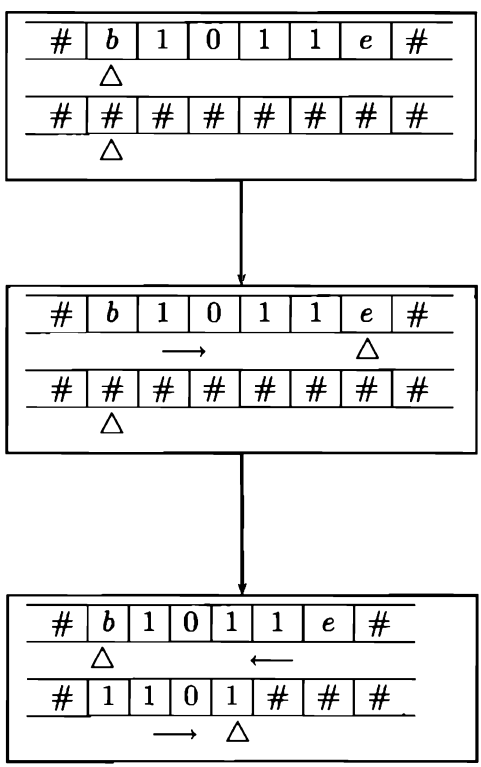


Рис. 1.1. Пример машины Тьюринга.

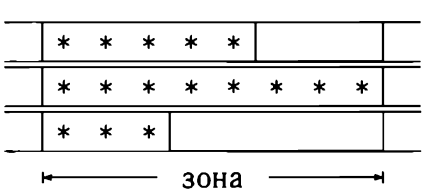
ГЛАВА 2

ВРЕМЯ И ПАМЯТЬ

§ 1. Время и зона машины Тьюринга

Время $T_M(input)$ — это число шагов в вычислении машины Тьюринга M на входе *input*. Шагом считается исполнение одной команды, т. е. соответствующие изменения состояния и содержимого обозреваемых ячеек памяти, а также перемещения головок, происходят за 1 времени.

Зона, память $S_M(input)$ определяется как максимум по всем рабочим лентам количества использованных ячеек к моменту остановки машины M на входе *input*. Если машина никогда не заканчивает работу, то полагаем зону бесконечной. Ячейка считается использованной, если в ней побывала головка.



Выбор максимума в определениии зоны вместо более естественной меры — суммарного количества использованных ячеек — значительно упрощает расчеты, но практиче-

ски не сказывается на получаемых оценках. В то же время оказывается существенным исключение входных лент из подсчета. Оно позволяет адекватно оценивать малые затраты памяти, когда вычисление требует рабочих ячеек меньше, чем длина входного слова.

Проблема с определением зоны: из определения непонятно, как алгоритмически проверять, что $S_M(input) < s$ (это было одним из желанных свойств определения меры сложности). На самом деле такой алгоритм существует. Он основан на следующей оценке:

Теорема 2.1. *По описанию машины Тьюринга M можно вычислить константу C такую, что если процесс вычисления M на некотором входе заканчивается за t шагов с зоной s, то*

$$s \leqslant t \leqslant n^{k_0} C^s,$$

где n — максимум длин входных слов, k_0 — число входных лент.

Доказательство. Левое неравенство: за один шаг на рабочей ленте расходуется не более одной ячейки.

Докажем правое неравенство. Для данного вычисления определим понятие конфигурации в данный момент времени: это состояние, содержимое всех лент и положение всех головок (относительно левой границы блока использованных на данной ленте ячеек). Легко видеть, что если в два разных момента времени мгновенные конфигурации совпадают, то вычисление зациклилось и никогда не закончится. Значит, для нашего (заканчивающегося) вычисления все конфигурации различны. Если зона вычисления есть s, вход фиксирован, то различных возможных конфигураций не более, чем

$$|Q| \cdot |\Sigma|^{(k-k_0)s} \cdot (n+2)^{k_0} \cdot s^{k-k_0} \leqslant n^{k_0} C^s.$$

Здесь $|Q|$ — количество состояний, $|\Sigma|$ — количество букв в ленточном алфавите, k — общее количество лент. Сомножитель $|\Sigma|^{(k-k_0)s}$ оценивает сверху число различных вариантов заполнения лент буквами алфавита $|\Sigma|$, а произведение $(n+2)^{k_0} \cdot s^{k-k_0}$ — количество возможных расположений головок.

Число шагов t не больше числа различных конфигураций, т. е. оценивается сверху этой же величиной $n^{k_0}C^s$. □

Алгоритм проверки условия «$S_M(input) < s$»:

1. По $M, input, s$ вычисляем $T = n^{k_0}C^s$ и моделируем вычисление $M(input)$ на T шагов.

2. Если оно закончилось и фактическая зона оказалась меньше s, то возвращаем $true$; иначе — $false$.

§ 2. Цена сокращения алфавита

Теорема 2.2. *Для вычисления функции типа*

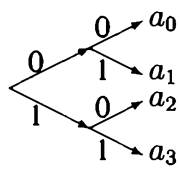
$$(\{0,1\}^*)^{k_0} \to (\{0,1\}^*)^{l_0}$$

достаточно ленточного алфавита $\{0, 1, \#, b, e\}$. *При переходе к такому алфавиту зона и время изменяются линейно. (Буквы* b *и* e *используются только как ограничители на входных лентах.)*

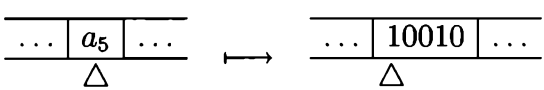
Замечание. При прямой конструкции добавится l_0 рабочих лент (выходных). Если не добавлять, то время может возрасти не более, чем квадратично.

Доказательство. Для простоты изложения рассмотрим случай одной рабочей ленты. Пусть исходная машина уже модифицирована так, что на рабочей ленте она не оставляет «пустыми» (т.е. #) ячейки, в которых бывала. (Это можно сделать с помощью введения новой буквы * — двойника для #.)

Фиксируем подходящее число $m \approx \log_2 |\Sigma|$ так, чтобы каждую букву ленточного алфавита Σ можно было однозначно закодировать блоком из m бит. Фиксируем такое кодирование. Его удобно представлять в виде бинарного дерева D глубины m: пути из корня к листьям — коды; на листе — соответствующая буква алфавита Σ.



Каждой ленточной (без состояния) конфигурации исходной машины сопоставим ленточную конфигурацию новой машины, заменив на рабочей ленте буквы алфавита Σ кроме # на их коды (головки в началах соответствующих блоков). Зона возрастет в m раз.



Каждый шаг исходной машины будет моделироваться $2m$ шагами (изменение буквы a_i на a_j) и еще m шагов потребуется для перемещения головки к соседнему блоку. Основное состояние моделирующей машины состоит из

1. состояния исходной машины;

2. переменной для хранения одного символа, принадлежащего $\{N, L, R\}$;

3. переменной-счетчика для хранения одного из чисел $0, \ldots, m-1$;

4. пометки «read», «write» или «go»;

5. бинарного дерева D с выделенной вершиной. В начальный момент моделирования шага выделен корень, а пометка есть «read».

Заметим, что число состояний конечно и не зависит от входа.

Этап «read». Первые m шагов моделирования состоят в том, что головка сканирует код буквы слева направо, а в состоянии меняется только выделенная вершина дерева D — она перемещается по пути, соответствующему считываемому коду. Когда она достигнет листа, на нем будет написана изменяемая буква a_i. Тогда пометка «read» меняется на «write» и выделенной вершиной становится лист с буквой a_j. В этот момент меняем хранимое состояние исходной машины и (если необходимо) производим перемещение головок на входных лентах так, как это делала моделируемая машина. Также определяем и запоминаем требуемое направление движения головки на рабочей ленте.

Этап «write». Следующие m шагов головка движется по блоку в обратном направлении (налево), а выделенная вершина — к корню дерева D. При этом путь, проходимый выделенной вершиной, определяет биты кода буквы a_j (справа налево) и они пишутся в соответствующие ячейки блока. Когда выделенная вершина достигает корня, пометка «write» меняется на «go».

Этап «go». Если запомнен символ движения L или R, то перемещаем головку на m позиций в соответствующем направлении, отсчитывая их с помощью счетчика. В конце меняем пометку «go» на «read» и переходим к модели-

рованию следующего шага. Если запомнен символ N, то просто меняем пометку.

Заметим, что очередное состояние всегда есть функция от предыдущего и очередной считанной буквы, как того и требует формализм.

Отдельные усилия требуются для моделирования поведения машины на границе рабочей зоны — когда головка моделирующей машины в начале шага находится в ячейке с буквой #. Тогда текущий и ближайшие вправо $m-1$ символов # (они обязательно будут такими!) надо заменить на код буквы # (понадобятся дополнительные состояния), а только потом выполнять основной шаг.

Так построенная моделирующая машина делает то же, что и исходная, но выдает ответ в виде последовательности кодов букв 0 и 1. Для декодирования добавляем одну дополнительную выходную ленту и в конце переписываем на нее результат, декодируя в процессе переписывания. Время на это — $m \cdot$ (длина результата), что не превосходит

$$m \cdot (\text{время исходного вычисления}).$$

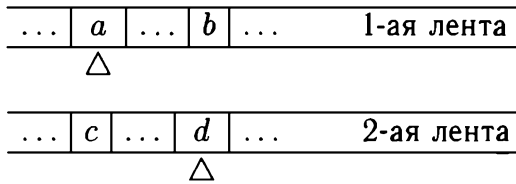
□

§ 3. Цена сокращения количества лент

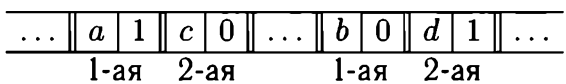
Замечание. Одна входная и одна выходная ленты (рабочих — много) моделирует общий случай за счет введения буквы-разделителя. Цена — время на переписывание входа с одной ленты на много и аналогичного переписывания результата.

Теорема 2.3. *Несколько рабочих лент моделируются одной с квадратичным замедлением по времени. Зона меняется линейно. Если ленточный алфавит был не однобуквенным, то его удается сохранить.*

Доказательство. (Набросок.) Пара рабочих лент



моделируется одной так:

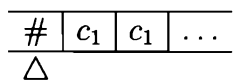


Одной ячейке на исходной ленте соответствуют две соседние ячейки на моделирующей. Левая содержит ту же букву, а правая (0 или 1) — признак того, что одна из головок исходной машины обозревает моделируемую ячейку. Такие пары чередуются, т. е. соседним ячейкам на одной из исходных лент соответствуют пары, расположенные на расстоянии 2. Легко видеть, что зона возрастает в два раза.

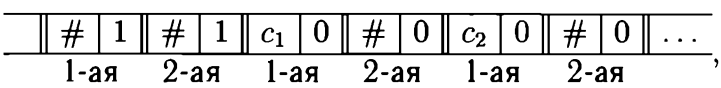
Один шаг исходной машины моделируется двойным проходом по всей рабочей зоне моделирующей ленты. При движении справа налево определяются буквы, обозреваемые исходной машиной, а при движении в обратную сторону имитируются необходимые изменения (замена букв и перемещение головок). Моделирующее состояние поддерживает счетчик для хранения остатка от деления на 4 величины текущего смещения головки, что позволяет определить, к какой из моделируемых лент относится обозреваемая ячейка. При этом в каждой ячейке рабочей зоны головка моделирующей машины бывает не более фиксированного числа (c) раз, поэтому время работы моделирующей машины не превосходит $c \cdot 2S \cdot T$, где T — время, а S —

зона исходного вычисления. Так как $S \leqslant T$, то это моделирование приводит к квадратичному замедлению. □

Замечание. Что изменится, если надо избавляться не только от многих рабочих лент, но и от входной ленты, т. е. моделировать вычисления функции типа $\{0,1\}^* \to \to \{0,1\}^*$ на одноленточной машине? Пусть моделируемая машина уже двухленточная, одна из которых — входная. Надо добавить 2 модуля: один будет переписывать входное слово «разряженно»,



в



,

а второй — выполнять обратное преобразование с результатом. Все остальное моделирование сохраняется. Добавление требует полиномиального времени и линейной зоны (от длины входа и результата).

Итог: Каждую вычислимую на многоленточной машине функцию типа $(\{0,1\}^*)^{k_0} \to \{0,1\}^*$ можно вычислить на машине с единственной рабочей лентой и ленточным алфавитом $\{0,1\}$ *с полиномиальным замедлением по времени и линейным увеличением зоны.* Для функций типа $\{0,1\}^* \to \{0,1\}^*$ моделирующая машина может быть выбрана одноленточной (т. е. и без входных лент тоже) с теми же оценками времени и памяти, если исходное вычисление по крайней мере читает все входное слово. Последнее условие нужно для того, чтобы вклад дополнительных модулей не превышал основного. Для этого достаточно, чтобы время и зона исходного вычисления оценивались снизу линейной функцией от длины входа.

Конечно, аналогичные факты справедливы и для произвольного алфавита мощности > 1, а также для вектор-функций с размерностью значений $l_0 \geqslant 1$, если разделить функции рабочей и выходных лент (например, разрешить на выходной ленте только писать с одновременным сдвигом головки вправо).

ГЛАВА 3

УНИВЕРСАЛЬНЫЕ МАШИНЫ ТЬЮРИНГА

§ 1. Машина Тьюринга, универсальная для класса C

Пусть фиксирован ленточный алфавит $\Sigma \supset \{0, 1\}$, наборы входных, рабочих и выходных лент. Устроим кодирование класса C всех таких машин Тьюринга словами в алфавите $\{0, 1\}$:

1. Состояния условимся записывать двоичными записями их номеров, взятыми в кавычках: '101' есть q_5. Пусть начальное состояние всегда имеет номер 1, а заключительное — 0.

2. Машину Тьюринга сначала запишем в виде программы, состоящей из команд вида

$$q\bar{a} \mapsto q'\bar{a}'D;$$

   Программа оказывается словом в фиксированном алфавите программ.

3. Фиксируем однозначное кодирование букв алфавита программ двоичными словами фиксированной дли-

ны и применим это кодирование к программе машины Тьюринга $M \in \mathcal{C}$. Так определяется *код машины* $Code(M)$.

Определение 3.1. Универсальной для класса $\mathcal{C}$ называется машина Тьюринга U с дополнительной входной лентой такая, что

$$U(Code(M), input) \simeq M(input) \tag{3.1}$$

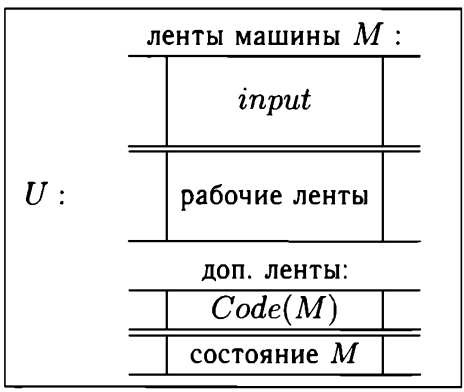
выполняется для всех $M \in \mathcal{C}$ и всех входов $input \in (\Sigma^*)^{k_0}$.

Замечание. Условие (3.1) означает, что оба вычисления $U(Code(M), input)$ и $M(input)$ либо зацикливаются, либо дают одинаковый результат. Использованный в определении символ $\simeq$ есть так называемое условное равенство. Это традиционный заменитель обычного отношения равенства, когда приходится сравнивать выражения, которые могут оказаться неопределенными. Утверждение $a \simeq b$ считается истинным в двух случаях: когда оба выражения a и b определены и равны между собой, либо когда они оба не определены. Во всех остальных случаях утверждение считается ложным. Отличие от обычного равенства состоит в том, что когда хотя бы одно из выражений a или b не определено, выражение $a = b$ не является корректным утверждением и не может быть ни истинным, ни ложным. Напротив, $a \simeq b$ всегда есть корректное утверждение.

§ 2. Конструкция универсальной машины

Факт существования универсальной машины немедленно следует из тезиса Тьюринга. По коду машины можно алгоритмически восстановить саму машину и применить ее к данному входу, т. е. функция, которую должна вычислять U, вычислима. Значит, ее можно вычислить на машине Тьюринга.

Нетрудно явно построить универсальную машину U в том же алфавите *с одной дополнительной рабочей лентой*, для которой время вычисления $U(code(M), input)$ лишь в константу раз больше времени вычисления $M(input)$. Сама константа зависит от M; зона может возрасти не более, чем на аддитивную константу.



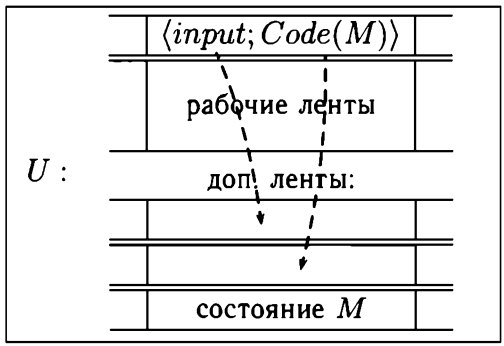
На дополнительной рабочей ленте U хранит двоичный код текущего состояния моделируемой машины M, а остальные ленты (кроме дополнительной входной) она использует так же, как M. Имитация одного шага M: машина U читает текущее состояние, ищет в Code(M) ту команду, которую надо выполнить и выполняет (измененное состояние записывается на место предыдущего). На все это уходит не более $(|Code(M)| + |Q|) \cdot c$ шагов.

Избавиться от дополнительной входной ленты (для $Code(M)$) позволяет следующее кодирование пар двоичных слов:

$$\langle \overline{a_1 \ldots a_n} \,;\, \overline{b_1 \ldots b_m} \rangle := \overline{a_1 \ldots a_n 01 b_1 b_1 \ldots b_m b_m}.$$

Код машины M можно подавать на вход, добавляя его к содержимому первой из входных лент в удвоенном виде (т. е. на ленту пишется $\langle input; Code(M) \rangle$). Тогда в начале

работы универсальная машина переписывает на две дополнительные рабочие ленты $input$ и $Code(M)$ по отдельности, а затем работает как указано выше.



Избавиться от дополнительных рабочих лент можно с помощью теоремы 2.3. Все это приведет к тому, что универсальная машина для класса C будет *присутствовать в самом классе*. Для класса с одной входной лентой определяющее ее равенство будет выглядеть

$$U(\langle input\ ;\ Code(M)\rangle) \simeq M(input), \quad M \in C, \qquad (3.2)$$

но время моделирования согласно теореме 2.3 возрастет квадратично, а зона — линейно. Тем самым установлена следующая теорема.

Теорема 3.2. *В классе C существует универсальная в смысле* (3.2) *машина U для этого класса. Для нее*

$$\begin{aligned} T_U(\langle input\ ;\ Code(M)\rangle) &= O(\ (T_M(input))^2\ ), \\ S_U(\langle input\ ;\ Code(M)\rangle) &= O(\ S_M(input)\ ). \end{aligned}$$

Замечание. Время моделирования можно несколько улучшить — до $O(T \log T)$ за счет того, что длина содержимого устраняемых рабочих лент в процессе моделирования машины M остается ограниченной и оставшейся головке удается таскать все эти данные по ленте с собой.

§ 3. Теоремы об иерархии по времени и по зоне

Вопрос. Когда добавочный ресурс в самом деле увеличивает вычислительные возможности?

Фиксируем класс C. Ограничимся случаем одной входной и одной выходной ленты. Пусть f — тотальная неубывающая неограниченная вычислимая функция из N в N. Определим классы $TIME(f)$ и $SPACE(f)$ как семейства всех функций типа $\Sigma^* \to \{0,1\}$, вычислимых на машинах из класса C с условием

$$T_M(v) < f(|v|) \text{ для достаточно больших } |v|$$

или

$$S_M(v) < f(|v|) \text{ для достаточно больших } |v|$$

соответственно.

Вопрос уточняется следующим образом: *найти условия на порядки роста функций f, g, достаточные для выполнения строгого включения*

$$\begin{gathered} TIME(f) \underset{\neq}{\subset} TIME(g) \\ (\text{соотв., } SPACE(f) \underset{\neq}{\subset} SPACE(g)\ ). \end{gathered} \tag{3.3}$$

Результаты, устанавливающие такие оценки, называются *теоремами об иерархии.*

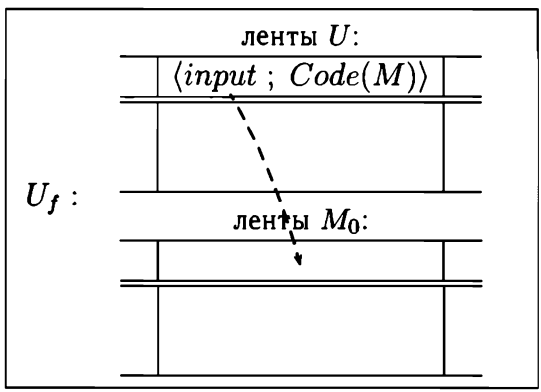
Способ получения теорем об иерархии.

Функция f называется конструируемой по времени (по зоне), если существует машина Тьюринга M, для которой

$$T_M(v) = f(|v|) \quad (\text{ соотв., } S_M(v) = f(|v|)\ ).$$

Пусть f конструируема по времени (или по зоне) машиной M_0. Рассмотрим следующую модификацию U_f универсальной машины U. К собственным лентам U в качестве

дополнительных рабочих добавлены ленты машины M_0. Сначала вход копируется на бывшую входную ленту M_0, после чего обе машины M_0 и U (в составе одной U_f) запускаются параллельно.



Машина M_0 служит таймером — как только она останавливается, прерываем работу U тоже. При этом синхронизируем работу машин U и M_0 таким образом, чтобы 1 шаг машины M_0 соответствовал 1 шагу моделируемой машины M, т. е. U делает все шаги, моделирующие 1 шаг M, после чего M_0 делает 1 шаг, и т.д. Результат читаем с выходной ленты U и меняем на противоположный: 0 на 1, а все остальное — на 0.

Легко видеть, что функция

$$h(v) = U_f(\langle v; v\rangle)$$

тотальна, вычислима, имеет тип $\Sigma^* \to \{0,1\}$, но не лежит в $TIME(f)$ (соответственно, в $SPACE(f)$). В самом деле, если бы $h \in TIME(f)$, то добавлением в программу вычисляющей ее машины «лишних» команд можно добиться выполнения условия $T_M(v) < f(|v|)$ (или аналогичного для зоны) уже при $v = Code(M)$. Но тогда

$$\begin{aligned} h(Code(M)) = M(Code(M)) &= U(\langle Code(M); Code(M)\rangle) \\ &\neq U_f(\langle Code(M); Code(M)\rangle) \end{aligned}$$

по построению. Противоречие.

Остается найти верхнюю оценку времени (соотв., зоны), достаточную для вычисления функции h. Рассмотренное ранее довольно грубое моделирование приводит к оценкам вида $(f(n))^C$ для времени и $C \cdot f(n)$ для памяти (константа C извлекается из конструкции). Это означает, что строгие включения (3.3) выполняются для всех функций g, растущих быстрее этих оценок. Более точные оценки следующие:

Теорема 3.3. *Для выполнения* (3.3) *достаточно, чтобы функция g удовлетворяла условию*

$$\frac{f(n)\log_2 f(n)}{g(n)} = o(1) \text{ (для времени)}$$

или

$$\frac{f(n)}{g(n)} = o(1) \text{ (для памяти)},$$

причем функция $f(n) \geqslant n$ должна быть конструируемой.

Можно показать непосредственным программированием, что задаваемые простыми формулами арифметические функции — конструируемы, поэтому

$$TIME(n) \underset{\neq}{\subset} TIME(n^2) \underset{\neq}{\subset} TIME(n^3) \underset{\neq}{\subset} \ldots$$
$$\ldots \underset{\neq}{\subset} TIME(2^n) \underset{\neq}{\subset} TIME(3^n) \underset{\neq}{\subset} \ldots,$$

$$SPACE(n) \underset{\neq}{\subset} SPACE(2 \cdot n) \underset{\neq}{\subset} \ldots$$
$$\ldots \underset{\neq}{\subset} SPACE(n^2) \underset{\neq}{\subset} \ldots \underset{\neq}{\subset} SPACE(2^n) \underset{\neq}{\subset} \ldots.$$

ГЛАВА 4

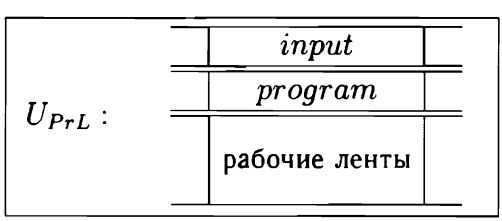
МОДЕЛИРОВАНИЕ ДРУГИХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

§ 1. Схема моделирования других языков программирования машинами Тьюринга

Пусть для некоторого языка программирования PrL программы и возможные входные/выходные данные удается представить словами некоторого конечного (м.б. достаточно большого) алфавита. Естественно также предположить, что отображение

$$program, input \mapsto output$$

окажется вычислимой словарной функцией. Тогда по тезису Тьюринга найдется машина Тьюринга U_{PrL}, вычисляющая это отображение. Она будет универсальной для языка PrL, т. е. $U_{PrL}(input, program) \simeq program(input)$.



Построение универсальной машины, расширенной блоком записи на ее «программную» входную ленту исходной PrL-программы, фактически есть компиляция PrL в язык машин Тьюринга.

§ 2. Моделирование RAM

В качестве примера рассмотрим простейший вариант языка BASIC (модели вычислений такого рода называются RAM, Random Access Machines, или машинами с неограниченными регистрами).

- Программа оперирует с конечным набором переменных V0, V1, ... типа int (двоичное представление, разрядность заранее не ограничена). Вход и выход записывается в несколько первых переменных.
- Команды программы пронумерованы и выполняются в порядке нумерации. В языке два вида команд — присваивание (например, `33: V5 <- V3+V1`) и условный переход (например, `125: if V5<0 goto 7`).

Для удобства обработки текста программы машиной Тьюринга выберем следующий «тэговый» вариант синтаксиса:

ПРИСВАИВАНИЕ:

```
<LET уникальный_номер> <V номер> выражение </LET>
        выражение::= константа
                   |<V номер>
                   |<V номер> op <V номер>
                op::= + | - | *
```

УСЛОВНЫЙ ПЕРЕХОД:

```
<IF уникальный_номер> <V номер> номер </IF>
```

команда:	ее запись:
33: V5 <- V3+V1	<LET33><V5><V3>+<V1></LET>
125: if V5<0 goto 7	<IF125><V5>7</IF>

Соответствующая универсальная машина Тьюринга в качестве своей части имеет «математический процессор» — набор лент и компонент состояний для реализации операций $+$, $-$ и $*$, рабочую ленту для хранения текущих значений переменных в формате **<V номер> значение </V>**, а также отдельную рабочую ленту для хранения уникального номера исполняемой команды. Исполнение очередной команды сводится к поиску ее текста на входной «программной» ленте, произведения мат. вычисления, записи его результата в соответствующую переменную и нахождения номера следующей команды.

Нетрудно заметить, что время моделирования одного шага оценивается полиномом от текущей зоны, которая, в свою очередь оценивается линейной функцией от s — максимума длин двоичных записей значений переменных (максимум — по всему вычислению). Таким образом, для моделирующей машины

$$TIME = O(t \cdot poly(s)), \quad SPACE = O(s), \tag{4.1}$$

где t — количество шагов, а s — максимум длин двоичных записей текущих значений переменных моделируемого вычисления.

Замечание. Полученные оценки не изменятся, если в язык добавить другие *представляемые словами* типы данных и дополнительные встроенные операции, при условии вычислимости последних на машинах Тьюринга за полиномиальное время на линейной зоне. Нетрудно также реализовать этими средствами управляющие конструкции:

IF vi>=0 THEN ... ELSE ... FI
Цена — дополнительных 2 шага.

`FOR vi:= vj DOWNTO 0 DO ... DONE`
Цена: +2 шага к каждой итерации, но само количество итераций есть `vj+1` и оценка (4.1) сохранится только в том случае, когда текущее значение переменной `vj` есть $poly(s)$ (т. е. его длина — $O(\log s)$).

`REPEAT ... UNTIL vi>= 0`
Цена: +1 шаг к каждой итерации, но для получения оценок времени необходим честный подсчет числа итераций.

`WHILE vi >= 0 DO ... DONE`
Цена: +2 шага к каждой итерации но для получения оценок времени необходим честный подсчет числа итераций.

Замечание. Некоторое неудобство для программирования представляет ограничение, состоящее в фиксированности для данной программы числа используемых переменных. Его легко преодолеть добавлением в синтаксис ссылок: **<REF номер>** означает содержимое переменой **<V номер'>**, где **номер'** есть содержимое переменной-указателя **<V номер>**. Разрешим использовать ссылки всюду, где могли стоять переменные. Например, если в переменной `V5` хранилось число 31, то команда
`25: REF5 <- 13`
приводит к присваиванию `V31 <- 13`. Если же значение `V5` не было определено заранее, то происходит ошибка. Нетрудно заметить, что в случае такого расширения оценки (4.1) сохранятся, если взять $s = m \cdot l$, где m — количество использованных программой переменных (прямо и косвенно), а l — максимум длин двоичных записей их текущих значений.

Эти оценки обычно используют для получения грубых верхних оценок времени, достаточного для решения той или иной задачи на машинах Тьюринга. Выбирают подходящий язык PrL и программируют соответствующий алгоритм на нем, оценивая порядки величин t, s. Для получения более точных оценок стараются улучшить оценку (4.1) в случае моделирования специфического алгоритма.

§ 3. Моделирование булевых схем

Булевы схемы — это более простой (не универсальный) язык программирования того же сорта. Основное отличие состоит в том, что возможные значения переменных здесь булевы, т. е. 0 или 1. Отсутствует нумерация команд и условные переходы, нет возможности оперировать с адресами команд и переменных. Программа состоит из последовательности операторов присваивания

`y <- <op> (x1,...,xk)`,

исполняемых последовательно. При этом фиксирована конечная полная система булевых функций F (например, $\{\neg, \vee, \wedge\}$) и `<op>`$\in F$. Все переменные, встречающиеся только в правых частях операторов присваивания, считаются входными. Некоторые из переменных схемы объявляются выходными. Таким образом, каждая схема вычисляет булеву вектор-функцию $\{0,1\}^n \to \{0,1\}^m$.

Другое представление булевых схем — это помеченный ориентированный граф без циклов (DAG). Вершины-источники (без входящих ребер) помечены входными переменными, а все остальные вершины — символами `<op>`$\in$ $\in F$. При этом в вершину, помеченную n-местной операцией, должно входить ровно n ребер. Некоторые из вершин отмечены как стоки (выходные переменные). Это представление соответствует программам, в которых в левых

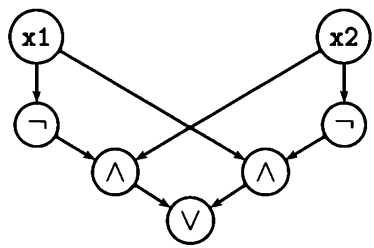
частях операторов присваивания переменные не повторяются.

Представления булевой схемы XOR:

программа

```
y1 <- ¬x1
y2 <- ¬x2
y3 <- y1 ∧ x2
y4 <- x1 ∧ y2
y5 <- y3 ∨ y4
```

DAG



В качестве временной сложности булевой схемы S используют размер схемы $size(S)$ — длину программы (т. е. количество операторов) в первом случае и количество помеченных символами операций вершин — во втором. Легко заметить, что по отношению к временной сложности эти два представления «эквивалентны» (в программе можно переименовать переменные так, чтобы имена переменных в левых частях операторов не повторялись, а программа вычисляла ту же функцию с той же временной сложностью).

Схемной сложностью функции $f : \{0,1\}^n \to \{0,1\}^m$ называется

$$c(f) = \min\{size(S) \mid S \text{ вычисляет } f\}.$$

Эта характеристика зависит от выбора полной системы F, т. е. $c(f) = c_F(f)$, но зависимость не очень существенна: при переходе к другой полной системе схемная сложность функций меняется на множитель O(1). В дальнейшем полную систему F будем считать фиксированной.

Универсальная для языка булевых схем машина Тьюринга строится полностью аналогично случаю RAM, но с соответствующими упрощениями (нет необходимости отдельно хранить номер текущей команды, не надо моделировать IF). Оценка времени работы моделирующей машины Тьюринга в этом случае будет

$$TIME = O(\ (size(S))^2 \cdot \log size(S)\ ). \qquad (4.2)$$

Она получается аналогично (4.1). Так как количество переменных ограничено сверху размером схемы, а на хранение одной переменной достаточно

$$\log size(S) + \text{const}$$

клеток ленты (не более $\log size(S)$ на номер переменной и const на булево значение и разделители), то для хранения текущих значений переменных уходит $s = size(S) \times$ $\times \log size(S)$ клеток ленты.

Моделирование исполнения одного оператора сводится к поиску значений операндов ($O(s)$ шагов), вычисления значения правой части оператора (булева операция вычисляется за $O(1)$ шагов) и записи результата в переменную, стоящую в правой части оператора (еще $O(s)$ шагов). Перемещение головки на «программной» ленте к следующему оператору дает незначительный вклад $O(\log size(S))$ шагов. Итого, на один оператор моделируемой программы уходит $O(s)$ шагов, а всего операторов $size(S)$, что и дает требуемую оценку.

Замечание. На самом деле логарифмический сомножитель в (4.2) можно убрать, организовав хранение значений всех переменных без имен (номеров). Поиск значения переменной номер i в этом случае осуществляется выбором i-го значения от начала. Этот же метод может быть применен и в случае BASIC, но не проходит при добавлении ссылок.

Часть II

СЛОЖНОСТНЫЕ КЛАССЫ

ГЛАВА 5

КЛАСС P

§ 1. Определение класса P

Будем говорить, что функция f — *полиномиального роста*, если для некоторых c и n_0 выполнено

$$f(n) < n^c \text{ при } n > n_0.$$

Время работы машины Тьюринга M на словах алфавита Σ (подмножество ленточного алфавита) *полиномиально*, если

$$T_M(n) = \max_{|\mathbf{x}|=n} T_M(\mathbf{x}), \text{ где } |\mathbf{x}| = |x_1| + \ldots + |x_k|,$$

есть функция полиномиального роста (в частности, такая машина всегда останавливается на словах алфавита Σ; можно также использовать $|\mathbf{x}| = \max |x_i|$ — получится эквивалентное определение).

Определение 5.1. *Класс P в широком смысле состоит из всех тотальных функций $f : (\Sigma^*)^k \to (\Sigma^*)^l$, вычислимых на машинах Тьюринга с полиномиальным временем работы. Класс P в узком (основном) смысле, состоит из всех предикатов $L : \Sigma^* \to \{0, 1\}$, вычислимых на машинах Тьюринга с полиномиальным временем работы.*

Замечание. Часто предикаты на множестве Σ^* отождествляют с их областями истинности

$$\{x \in \Sigma^* \mid L(x) = 1\}$$

и называют формальными языками. В этом случае элементы класса P (в узком смысле) есть языки, для которых условие «$x \in L$» распознается за полиномиальное время.

Замечание. Алфавит Σ в этом определении не фиксирован — в класс P входят все такие функции и предикаты для всевозможных конечных алфавитов:

$$P = \bigcup_{\Sigma} P_{\Sigma}.$$

В то же время для каждого алфавита Σ двоичное кодирование букв порождает естественное вложение $P_{\Sigma} \mapsto P_{\{0,1\}}$, поэтому $P_{\{0,1\}}$ фактически уже содержит в себе все функции и предикаты из P.

Замечание. Определение класса P в значительной мере не зависит от модели вычисления. Если модель вычислений допускает компиляцию в машины Тьюринга и обратно с полиномиальным замедлением (а таковы все реальные языки программирования), то класс P, определенный посредством этой модели совпадает с нашим. В частности, совершенно несущественен выбор варианта машин Тьюринга (сколько каких лент, головок и т. п.).

Замечание. Имеется устойчивое общественное мнение, что предикаты и функции не из P реально вычислять гораздо труднее, чем предикаты и функции класса P. Практические задачи, которые лежат в классе P, но требуют для своего решения времени, оцениваемого полиномом большой степени, встречаются крайне редко. Обычно приходится сталкиваться с одной из двух реалий: либо

задача решается за время — полином небольшой степени с не слишком большими коэффициентами, либо она требует экспоненциального времени, причем резкий рост временных затрат начинается уже на малых длинах входных данных. Эти различия наглядно проявляются в процессе эксперимента, поэтому факт принадлежности задачи классу P позволяет прогнозировать реальное поведение соответствующей программы.

§ 2. Примеры: целочисленная арифметика

Считаем, что числа представлены в двоичной записи. Тогда операции +, -, *, div (деление нацело), mod (остаток от деления), а также операция вычисления наибольшего общего делителя (a, b) чисел a и b принадлежат классу P. Здесь работают «школьные» алгоритмы и алгоритм Евклида: $(a, b) = (b,\ a \bmod b)$.

Заметим, что $\exp(x) = 2^x$ не принадлежит классу P, так как длина результата уже не полиномиальна как функция от длины x (т. е. от $\log_2 x$).

Например, для `d = a div b` и `r = a mod b` :

```
d:=0;
WHILE (a >= b) DO       // (log a) итераций
   x:=1;

   WHILE a-x*b >0 DO      // (log a) итераций.
      z:=x;               // Подбираем z -
      x:=x*2;             // наиб. степень 2 такую,
   DONE;                  // что a=z*b+y и y>=0

   d:=d+z; a:=a-z*b;
DONE
r:=a.
```

Представим себе реализацию этого алгоритма на описанном выше варианте BASIC'а. Здесь 7 переменных, а в реализации будет $7 + \text{const}$. Здесь 10 операторов, а в реализации будет $10 \cdot \text{const}$. Здесь длины двоичных записей текущих значений переменных не более $s = (\log a + \log b) \times$ $\times$ const и там — тоже. Здесь каждый оператор исполнялся не более $\log^2 a$ раз и там — тоже. В итоге для реализации будет $t = C_1 \cdot \log^2 a$, а $s = (\log a + \log b) \cdot C_2$. По (4.1) получаем, что при моделировании этого машиной Тьюринга время работы окажется ограниченным полиномом от $n =$ $= \log a + \log b$. Но такое значение n и есть длина входных данных, откуда div, mod $\in P$.

Алгоритм Евклида `d=(a,b)`:

```
IF b>a THEN x:= a; a:= b; b:= a FI;  //делаем a>=b

WHILE b>0 DO
   x:= a mod b;   // (a,b):= (b, a mod b)
   a:= b;
   b:= x
DONE;
d:= a.
```

На каждой итерации произведение ab уменьшается по крайней мере в 2 раза (так как $a \geqslant b + (a \bmod b) \geqslant$ $\geqslant 2 \cdot (a \bmod b)$), поэтому количество итераций не больше $\log_2 ab = \log_2 a + \log_2 b$, т. е. длины входных данных. Далее повторяем те же рассуждения про моделирование.

§ 3. Примеры: арифметика остатков

Операции $+$, $*$ в кольцах вычетов $\mathbb{Z}_m$ легко вычисляются за полиномиальное время с помощью целочисленной

арифметики. Алгоритм Евклида позволяет за полиномиальное время по a и m распознать обратимость элемента a в кольце $\mathbb{Z}_m$ (проверяем условие $(a, m) = 1$). Более тонкие факты: функции $f(x, n) = x^{-1}$ в $\mathbb{Z}_m$ (если не существует, то 0) и $\exp(x, y, m) = x^y \bmod m$ также лежат в классе P.

1. $x^y \bmod m$. Сначала заметим, что при $y = 2^k$ это выражение можно вычислить с помощью k итераций оператора `x:=x*x mod m`. Общий случай сводится к этому при помощи разложения

$$y = 2^{k_1} + \ldots + 2^{k_n}, \quad k_1 < \ldots < k_n, \quad n \leqslant k_n \leqslant |y|.$$

Достаточно следующего алгоритма:

```
exp:=1; z:=1;
WHILE y>0 DO                  // будет n итераций
   IF (y mod 2 =1) THEN
      u:= (x^z) mod m;        // z есть степень двойки
      exp:=exp*u
   FI;
   z:=z*2;
   y:=y div 2
DONE.
```

2. $f(x, n) = x^{-1}$ в $\mathbb{Z}_m$. Если x — обратимый элемент кольца $\mathbb{Z}_m$, то $(x, m) = 1$ и достаточно найти разложение единицы $\alpha x + \beta m = 1$ (тогда $x^{-1} = \alpha \bmod m$). Для поиска такого разложения следует модифицировать алгоритм Евклида, т. е. реализацию выполняемого в цикле присваивания

```
r := a mod b ;
(a,b) := (b, r) ;
```

Будем хранить текущие компоненты пары (a, b) в виде целочисленных линейных комбинаций исходных данных:

$$a = \alpha x + \beta m, \quad b = \gamma x + \delta m.$$

В начальный момент $\alpha = \delta = 1$, $\beta = \gamma = 0$. Тогда

$$r = a - (a \operatorname{div} b) \cdot b,$$

что позволяет найти коэффициенты разложения для следующей итерации. Алгоритм заканчивает работу на паре $((x, m), 0) = (1, 0)$, поэтому в результате будет построено требуемое разложение единицы. Каждая итерация увеличится на фиксированное число шагов, но количество итераций не изменится. Поэтому сохранится прежняя полиномиальная оценка на время работы.

§ 4. Примеры: сложение и умножение матриц

Для простоты ограничимся квадратными $(n \times n)$ матрицами с элементами из кольца $\mathbb{Z}_{2^m}$, представленными двоичными записями с лидирующими нулями длины $m = \text{const}$. На вход поступают число n и элементы матриц A,B (записанные подряд через разделитель). Длина входа есть полином от n, поэтому достаточно построить вычисление, ограниченное по времени другим полиномом от n. Рассмотрим, например, умножение.

Тьюрингово вычисление будет состоять в моделировании соответствующей программы обсуждавшегося ранее варианта BASIC'а со ссылками. Эта программа получает исходные данные в качестве начальных значений переменных $V0, \ldots, V(2n^2)$ и должна записать элементы произведения матриц в переменные $V(2n^2 + 1), \ldots, V(3n^2)$. Моделирующая машина Тьюринга должна проделать инициализацию переменных и в конце переписать результат на выходную ленту; очевидную реализацию этого мы опускаем.

Сама BASIC-программа получается как результат моделирования стандартного метода перемножения матриц ($O(n^3)$ итераций)

```
FOR i=0 TO n-1 DO
   FOR j=0 TO n-1 DO
      FOR k=0 TO n-1 DO
      c[i,j]:=c[i,j]+a[i,k]*b[k,j]
      DONE
   DONE
DONE
```

Единственное, что требует разъяснений — реализация массивов с помощью ссылок. Вместо выражения `a[i,k]` в BASIC-программе надо использовать ссылку `ref d` на значение `a[i,k]`. Оно хранится в переменной $V(i \cdot n + k)$, поэтому переменной Vd надо предварительно присвоить значение $(i \cdot n + k)$. Таким образом, оператор ... `a[i,k]` ... надо переводить так:

```
100: Vd <- i * n
101: Vd <- Vd + k
102: ... (ref d) ...
```

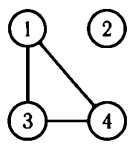
Здесь `i`, `k`, `n` обозначают переменные, хранящие значения i, k, n. Для остальных двух массивов поступаем аналогично, но учитываем, что соответствующие адреса есть $(n^2 + k \cdot n + j)$ и $(2 \cdot n^2 + i \cdot n + j)$. В итоге тело самого внутреннего цикла будет переведено фиксированным конечным набором операторов присваивания и временная оценка $t = O(n^3)$ при переводе всего блока сохранится. Суммарная длина содержимого всех переменных будет не более

$$s = 3 \cdot n^2 \cdot m + \text{const} \cdot \log n = O(n^2).$$

Применяем (4.1) и получаем полиномиальную оценку на время соответствующего тьюрингова вычисления.

§ 5. Примеры: связность в графе

Граф G с множеством вершин $\{1, \ldots, n\}$ задан матрицей смежности M. Это симметричная $n \times n$ матрица из 0 и 1; $m_{i,j} = 1$ означает, что в графе есть ребро (i, j). Например,

G: 

$$M = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}.$$

Надо по M, i, j выяснить, связаны ли вершины i, j в графе G.

Эта задача легко сводится к вычислению n-ой степени 0,1-матрицы $A = (M + E)$, если в качестве операций $+, \cdot$ взять булевы операции $\vee, \wedge$ (далеко не самый эффективный метод, но для доказательства принадлежности классу P работает). Степень матрицы A^q задает отношение связности за не более чем q шагов, поэтому для $(a_{i,j}^{(n)}) = A^n$

$$a_{i,j}^{(n)} = 1 \Leftrightarrow i \text{ и } j \text{ связаны в } G.$$

Вычислять A^n можно n-кратным перемножением матриц (см. предыдущий пример; использование булевых операций вместо $+, \cdot$ ничего не меняет).

ГЛАВА 6

КЛАСС $P/Poly$

§ 1. Распознавание языков последовательностями булевых схем

Напомним, что формальным языком мы назвали произвольное множество слов данного алфавита. Ограничимся языками $L \subset \{0,1\}^*$. С каждым таким языком можно связать последовательность булевых функций $f_n(x_1, \ldots, x_n)$, $n = 0, 1, \ldots$ такую, что

$$x_1 \ldots x_n \in L \Leftrightarrow f_n(x_1, \ldots, x_n) = 1.$$

Схемной сложностью языка L называется функция

$$c_L(n) = c(f_n),$$

где через $c(\cdot)$ обозначена схемная сложность булевой функции.

Пример. Оценить сверху схемную сложность языка, состоящего из всех слов-перевертышей. Схема для f_n в этом случае может быть такая:

```
f <- ( x1 <-> xn )
```

```
z <- ( x2 <-> x(n-1) )
f <- ( f & z )
z <- ( x3 <-> x(n-2) )
f <- ( f & z )
...
```

Поэтому $c_L(n) = O(n)$.

Следует заметить, что схемная сложность языка является огрубленной сложностной характеристикой — она не учитывает трудоемкость построения самой схемы. Но за счет огрубления становится принципиально возможным оценивать сложность произвольных языков (а не только разрешимых, как для машин Тьюринга).

Класс $P/Poly$ определяется как семейство всех таких языков $L \subset \{0,1\}^*$, схемная сложность которых есть функция полиномиального роста, т. е.

$$c_L(n) < n^C \text{ при } n > n_0$$

для некоторых C и n_0.

Класс $P/Poly$ замкнут относительно операций объединения, пересечения и дополнения языков.

§ 2. Континуальность класса $P/Poly$

В отличии от класса P, в класс $P/Poly$ входят не только разрешимые языки. Мощность класса $P/Poly$ равна континууму. Каждой (не обязательно вычислимой) функции $\varphi : N \to \{0,1\}$ можно инъективно сопоставить язык L схемной сложности $O(1)$:

$$x_1 \dots x_n \in L \Leftrightarrow \varphi(n) = 1.$$

Класс P представляет собой эффективизованную версию класса $P/Poly$: если к определению языка $L \in P/Poly$

добавить условие вычислимости за полиномиальное время отображения

$$n \xrightarrow{s} \begin{cases} \text{булева схема полиномиального раз-} \\ \text{мера, вычисляющая } f_n, \end{cases}$$

то класс таких языков в точности совпадает с $P_{\{0,1\}}$.

Для доказательства принадлежности языков L, лежащих в модифицированном указанным образом «равномерном» классе $P/Poly$, к классу $P_{\{0,1\}}$, достаточно реализовать следующий алгоритм проверки условия «$x \in L$»:

1. По входному слову $x = x_1 \dots x_n$ найти его длину n.
2. По n найти булеву схему S_n, правильно проверяющую условие «$x \in L$» на словах длины n.
3. Вычислить $S_n(x_1, \dots, x_n)$ и выдать полученный результат в качестве ответа.

Для реализации можно взять композицию машины Тьюринга, вычисляющей отображение s и универсальной машины Тьюринга для языка булевых схем. Получится машина Тьюринга, которая распознает язык L за полиномиальное время.

Обратное включение обеспечивает конструкция из доказательства теоремы 6.1 (см. ниже), которая на самом деле строит отображение s вычислимым за полиномиальное время.

§ 3. Включение $P \subset P/Poly$

Ранее обсуждалось, что с точностью до кодировки исходных данных класс P (в узком смысле) совпадает с $P_{\{0,1\}}$.

Теорема 6.1. $P_{\{0,1\}} \subset P/Poly$.

Доказательство. Пусть $L \in P$. Тогда существует одноленточная машина Тьюринга M, разрешающая L, и полином $p(n)$, ограничивающий сверху ее время работы на словах достаточно большой длины n. Этот же полином будет ограничивать и зону M. Занумеруем клетки ленты целыми числами, 0 — где стоит головка в начальный момент, входное слово — в клетках $1, \ldots, n$, число n — достаточно большое.

Протоколом работы машины Тьюринга M за время T на слове $x_1 \ldots x_n$ называется прямоугольная таблица следующего вида:

	$-S$		0	1		i			S
0	$\#, \emptyset$	$\ldots$	$\#, q_1$	$x_1, \emptyset$	$x_2, \emptyset$	$\ldots$	$x_n, \emptyset$	$\ldots$	$\#, \emptyset$
1	$\#, \emptyset$								$\#, \emptyset$
	$\vdots$								$\vdots$
j	$\#, \emptyset$					$\Gamma_{i,j}$			$\#, \emptyset$
	$\vdots$								$\vdots$
T	$\#, \emptyset$								$\#, \emptyset$

Номера столбцов соответствуют нумерации клеток ленты, номера строк — шагам вычисления. В клетке (i, j) таблицы записана пара $\Gamma_{i,j} = \langle a, q \rangle$, где a — содержимое i-й клетки ленты в момент $t = j$, а q есть соответствующее состояние машины, если в этот момент времени головка обозревает клетку i, и $\emptyset$ — в противном случае. Ширина таблицы $2 \cdot S + 1$ выбирается таким образом, чтобы головка машины в процессе работы все время находилась внутри куска ленты $|i| < S$. Если машина остановилась раньше момента T, то последние строки таблицы дублируют предыдущие. Для этого доказательства возьмем $T = S = p(n)$.

В клетке протокола может стоять произвольная пара $\langle a, q \rangle \in D$, где $D = \Sigma \times (Q \cup \{\emptyset\})$. Заметим также, что

содержимое клетки $(i, j+1)$ для $|i| < S$ функционально зависит от содержимого трех клеток над ней,

$\Gamma_{i-1,j}$	$\Gamma_{i,j}$	$\Gamma_{i+1,j}$
	$\Gamma_{i,j+1}$	

$$\Gamma_{i,j+1} = \varphi(\Gamma_{i-1,j}, \Gamma_{i,j}, \Gamma_{i+1,j}),$$

а содержимое клеток на верхней, левой и правой границах есть либо константа $\langle \#, \emptyset \rangle$ или $\langle \#, q_1 \rangle$, либо одна из следующих функций от n булевых переменных, представляющих входное слово:

$$\langle x_i, \emptyset \rangle = \pi_i(x_1, \ldots, x_n).$$

Входное слово принадлежит языку L тогда и только тогда, когда в нижней строке протокола встречается пара $\langle 1, q_0 \rangle$.

Добавим к языку программирования булевых схем переменные по конечному множеству D (новый тип данных), константы $\langle \#, \emptyset \rangle$, $\langle \#, q_1 \rangle \in D$ и дополнительные «встроенные» операции $\varphi : D^3 \to D$, $\pi_i : \{0,1\}^n \to D$, и $\chi : D \to \{0,1\}$, где

$$\chi(\langle a, q \rangle) = 1 \Leftrightarrow \langle a, q \rangle = \langle 1, q_0 \rangle.$$

В расширенном языке нетрудно написать программу полиномиального размера, отвечающую на вопрос «$x_1 \ldots x_n \in L$?». Она состоит из $(2 \cdot S + 1) \cdot T$ операторов присваивания, вычисляющих значения всех переменных $\Gamma_{i,j}$ по слоям сверху вниз и еще $2 \cdot (2 \cdot S + 1)$ операторов для вычисления ответа

$$f = \bigvee_{i=-S}^{S} \chi(\Gamma_{i,T}).$$

Остается промоделировать эту программу программой в исходном языке булевых схем (без типа D) с сохранением полиномиальной оценки на длину программы. Учитывая конечность множества D, можно выбрать число m

из условия $2^{m-1} < |D| \leqslant 2^m$ и имитировать каждую переменную Γ типа D набором из m булевых переменных. Использующие тип D дополнительные встроенные функции и константы можно заменить на конечные наборы булевых функций, вычисляющие биты результата по битам аргументов. Каждую из последних (их конечное множество) реализуем булевой схемой. Размеры всех этих схем ограничены некоторой константой d. Теперь каждый оператор программы в расширенном языке можно заменить на не более $m \cdot d$ операторов в исходном, производящих то же преобразование значений переменных, но в их двоичном представлении. Так как входные и выходные переменные нашей программы уже были булевыми, то это преобразование не изменит вычисляемой программой функции. При этом длина программы возрастет не более, чем в $m \cdot d$ раз, т.е. останется функцией полиномиального роста. $\square$

ГЛАВА 7

КЛАСС NP

§ 1. Определение класса NP

Определение 7.1. Язык $L \subset \Sigma^*$ принадлежит классу NP, если существует предикат $R(x, y) \in P$ и полином $q(n) = n^k$ такие, что для всех $x \in \Sigma^*$

$$x \in L \Leftrightarrow \exists y(|y| < q(|x|) \wedge R(x, y));$$

соответствующее слово y называется свидетелем принадлежности $x \in L$.

Замечание. Класс не изменится, если в определении ограничиться предикатами $R(x, y) \in P$ с дополнительным условием $R(x, y) \Rightarrow |y| < q(|x|)$, т. е. функцию отбраковки «длинных» свидетелей передать предикату R тоже. Тогда распознавание принадлежности языку L можно представить как интерактивный процесс, в котором человек пытается «убедить» машину (распознающую предикат R машину Тьюринга) в том, что «$x \in L$», а машина верифицирует его доводы. На первой входной ленте написано x. Человек пишет на вторую входную ленту потенциального свидетеля y, а машина вычисляет значение $R(x, y)$ и «соглашается», если результат вычислений есть 1. Слово x

принадлежит языку L, если у человека есть принципиальная возможность (но не обязательно эффективный метод) убедить в этом машину.

Замечание. В качестве алфавита для свидетелей достаточно использовать $\{0,1\}$ (он моделирует остальные). Можно также заменить условие «$|y| < q(|x|)$» на «$|y| = q(|x|)$», так как $\varphi(v10^k) = v$ есть вычислимая за полиномиальное время биекция множества $\{y \mid |y| = m\}$ на $\{y \mid |y| < m\}$ и вместо предиката $R(x,y)$ можно взять $R(x,\varphi(y))$.

Замечание. Каждый язык $L \in NP$ разрешим за экспоненциальное время с помощью перебора всех возможных свидетелей.

Другое эквивалентное определение класса NP использует интерактивную модель вычислений — *недетерминированные машины Тьюринга*. Отсюда происходит название класса NP — nondeterministic polynomial. Недетерминированная машина отличается от детерминированной тем, что в программе разрешается использовать несколько команд с одинаковой правой частью «$q\bar{a} \to \ldots$». Вычисление происходит интерактивно: человек выбирает одну из возможных в данный момент команд, а машина проверяет допустимость выбора и производит действия. Таким образом, для данного входа может быть не одно, а несколько вычислений. *Недетерминированная машина распознает (допускает) язык L за время $q(n)$*, если

$x \in L \;\Rightarrow$ на входе x существует вычисление длины $\leqslant q(|x|)$ с результатом 1;

$x \notin L \;\Rightarrow$ все вычисления на входе x не дают результата 1.

И в том и в другом случаях допускается наличие незаканчивающихся вычислений.

Определение 7.2. Язык L принадлежит классу NP, если существует полином q и недетерминированная машина Тьюринга, которая распознает L за время $q(n)$.

Лемма 7.3. *Определения 7.1 и 7.2 эквивалентны.*

Доказательство. ($\Rightarrow$) Считаем, что неравенство в определении 7.1 заменено на равенство. Процесс угадывания свидетеля y длины $q(|x|)$ моделируется недетерминированным заполнением ленты последовательностью из $q(|x|)$ нулей и единиц. Соответствующие команды с одинаковой левой частью:

$$q\bar{a} \to \ldots 0 \ldots$$
$$q\bar{a} \to \ldots 1 \ldots$$

На отдельной рабочей ленте организован счетчик числа повторений, обеспечивающий передачу управления дальше после $q(|x|)$ итераций. После них недетерминированная машина работает детерминированно и вычисляет значение $R(x, y)$.

($\Leftarrow$). Легко видеть, что неоднозначность в выборе команд недетерминированной машины можно свести к случаю двух команд с данной левой частью. Тогда каждое её вычисление длины $\leqslant q(|x|)$ однозначно определяется 0,1-последовательностью выборов y той же длины (0 означает, что выбран первый вариант очередной команды, 1 — второй). Рассмотрим детерминированную машину M, которая получает y в качестве дополнительного входа и моделирует интерактивную работу исходной недетерминированной машины, заменяя человеческое управление чтением очередного бита y. Позаботимся также, чтобы эта машина всегда останавливалась за $\leqslant q'(|x|)$ шагов (для некоторого полинома q') и возвращала 0 всякий раз, когда исходная не печатала 1 за $q(|x|)$ шагов. Такая машина M задает требуемый в определении 7.1 предикат $R \in P$;

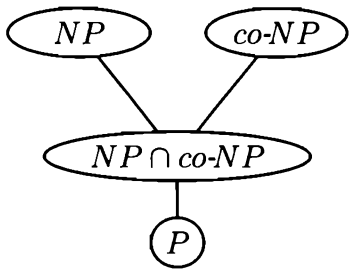


Рис. 7.1. Иерархия классов P, NP, $co\text{-}NP$.

в качестве оценки на длину свидетелей подходит полином q. □

§ 2. О проблеме $P \neq NP$

Имеет место включение $P \subset NP$. Для доказательства принадлежности языка $L \in P$ классу NP достаточно взять

$$R(x, y) = 1 \Leftrightarrow (x \in L), \quad q(n) \equiv 1.$$

Вопрос о строгости этого включения, то есть верно ли, что

$$P \neq NP?$$

Он остается центральной нерешенной проблемой в теории сложности вычислений уже более 30 лет. Хотя общественное мнение давно склонилось в пользу положительного ответа (т. е. классы различны) и полезность многих алгоритмов основывается на этом предполагаемом ответе, строгость включения до сих пор не доказана.

Имеется некоторая (частичная) аналогия между качественной теорией алгоритмов и количественной, т. е. теорией сложности. Классы P и NP соответствуют классам разрешимых и перечислимых множеств. Так, гипо-

теза о несовпадении классов P и NP есть аналог известной теоремы о существовании перечислимого неразрешимого множества. Другой известный результат качественной теории — критерий разрешимости Поста (множество A разрешимо тогда и только тогда, когда оно и его дополнение A^c перечислимы). Его аналогом является недоказанное и не опровергнутое (но сомнительное) равенство $P = NP \cap co\text{-}NP$, где $co\text{-}NP$ состоит из всех языков L, чье дополнение L^c принадлежит NP. В настоящее время известны лишь вытекающие непосредственно из определений нестрогие включения: $P \subset NP \cap co\text{-}NP \subset NP$.

§ 3. Примеры задач класса NP

Дальнейшие примеры задач из класса NP требуют кодирования рассматриваемых конечных объектов словами подходящего алфавита. Мы предполагаем, что подобное кодирование уже выбрано некоторым разумным (обычно, очевидным) образом.

Проблема выполнимости булевых формул (SAT). Булевы формулы — это правильные выражения, построенные из булевых переменных $p_1, p_2, \ldots$ с помощью булевых связок $\wedge$ (конъюнкция), $\vee$ (дизъюнкция), $\neg$ (отрицание) и скобок. Например,

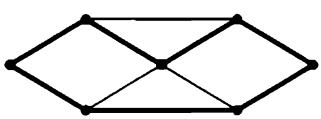
$$(\neg(p_1 \wedge p_2) \vee (\neg p_1 \wedge p_3)) \wedge p_2.$$

По булевой формуле $\varphi(p_1, \ldots, p_n)$ выяснить, существует ли выполняющий ее набор — набор истинностных (0 или 1) значений $b_1, \ldots, b_n$, для которого $\varphi(b_1, \ldots, b_n) = 1$.

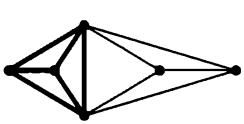
Свидетелем правильности ответа «да, выполнима» служит здесь любой выполняющий набор битов $(b_1, \ldots, b_n)$.

Его размер n не превосходит длины формулы, т. е. ограничен полиномом первой степени от ее длины. Свидетели имеются у всех выполнимых формул и только у них. Проверка того, что некоторый набор из n битов в самом деле свидетельствует о выполнимости формулы, сводится к вычислению ее истинностного значения при заданных значениях переменных, что проверяется за полиномиальное время. Отсюда, $SAT \in NP$.

Проблема существования гамильтонова цикла. По графу выяснить, существует ли в нем гамильтонов цикл (замкнутый путь по ребрам графа, проходящий через каждую вершину ровно 1 раз). Свидетель здесь — гамильтонов цикл.



Задача о клике. Подграф данного графа называется кликой, если он полный, т. е. каждая пара его вершин соединена ребром. Размер клики — это число ее вершин. По графу выяснить, существует ли в нем клика данного размера d. Свидетель здесь — клика.



Полимино, краевая задача. Полимино — это двухмерное домино. Костяшки — квадратные, на всех четырех сторонах — пометки (буквы фиксированного алфавита). Для

каждого варианта игры фиксирован свой набор типов квадратиков. Краевая задача: на сторонах клетчатого прямоугольника $m \times n$ пометки заданы. По краевой задаче и варианту игры определить, можно ли замостить весь прямоугольник по правилам домино. Для определенности считаем, что крутить костяшки нельзя (фиксирован верх). Свидетель — замощение.

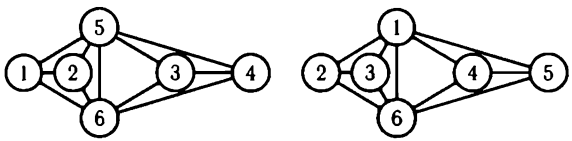
Существование целочисленного решения системы линейных неравенств. Дана система линейных неравенств с целыми коэффициентами. Выяснить, имеет ли она целочисленное решение. Свидетель — решение. Для доказательства принадлежности классу NP нужна полиномиальная оценка на длину свидетеля. Можно показать, что если такая система имеет целочисленные решения, то длина некоторого из них оценивается сверху полиномом от длины системы.

Другие примеры. Следующие задачи лежат в классе NP, но представляются более простыми, чем приведенные выше:

- Проверить, что данное натуральное число — составное[1]. Свидетель здесь — разложение натурального числа в произведение двух натуральных сомножителей, каждый из которых больше 1.

- Два графа называются изоморфными, если между их вершинами можно установить взаимно-однозначное соответствие, которое сохраняет смежность: если две вершины одного графа соединены ребром, то их образы в другом графе — тоже. Само соответствие

[1] В августе 2002 г. индийские математики M. Agrawal, N. Kayal и N. Saxena установили принадлежность этой задачи классу P.



$1 \mapsto 4$
$2 \mapsto 5$
$3 \mapsto 2$
$4 \mapsto 3$
$5 \mapsto 1$
$6 \mapsto 6$

Рис. 7.2. Изоморфизм двух графов.

называется изоморфизмом. Исходными данными задачи является пара графов. Проверить, что данные графы изоморфны. Свидетель здесь — изоморфизм.

ГЛАВА 8

ПРИМЕРЫ NP-ПОЛНЫХ ЗАДАЧ

§ 1. Сводимость $\leqslant_m^p$ (Карп), NP-полнота.

Общий подход к сравнению задач по трудности основан на понятии сводимости задач. Он позволяет сравнивать задачи, имеющие форму массовых проблем. Под массовой проблемой понимается семейство однотипных индивидуальных задач, различающихся лишь значениями исходных данных. Неформально, задача $\mathcal{A}$ сводится к задаче $\mathcal{B}$ (т. е. не труднее, «$\mathcal{A} \leqslant \mathcal{B}$»), если существует метод, позволяющий решать каждую индивидуальную задачу из $\mathcal{A}$, пользуясь решениями индивидуальных задач из $\mathcal{B}$, поставляемыми «оракулом» извне (по требованию). Естественно ожидать рефлексивность и транзитивность отношения сводимости $\leqslant$. Известно много различных по силе формализаций этого понятия. При изучении сложностных классов в первую очередь используют много-однозначную сводимость за полиномиальное время $\leqslant_m^p$ (сводимость Карпа).

Класс задач ограничивается массовыми проблемами распознавания языков. Для языка $L \subset \Sigma^*$ задача распознавания такова:

Вход:	Вопрос:
слово $x \in \Sigma^*$	$x \in L$?

Определение 8.1. *Говорят, что язык* $L_1 \subset \Sigma^*$ *сводится по Карпу к языку* L_2, *т. е.* $L_1 \leqslant_m^p L_2$, *если существует функция* $f \in P$ *такая, что для всех слов* $x \in \Sigma^*$ *выполнено:* $x \in L_1 \leftrightarrow f(x) \in L_2$.

$$L_1 \equiv {}_m^p L_2 \quad \Leftrightarrow \quad (L_1 \leqslant_m^p L_2) \wedge (L_2 \leqslant_m^p L_1).$$

Лемма 8.2.

1) *Отношение* $\leqslant_m^p$ *рефлексивно и транзитивно*;

2) $L_1 \leqslant_m^p L_2,\ L_2 \in P \ \Rightarrow\ L_1 \in P$;

3) $L_1 \leqslant_m^p L_2,\ L_2 \in NP \ \Rightarrow\ L_1 \in NP$.

Доказательство. Простое упражнение. □

Определение 8.3. Язык L называется NP-*трудным*, если все языки из NP к нему $\leqslant_m^p$-сводятся. NP-трудный язык называется NP-*полным*, если он сам также принадлежит классу NP.

Замечание. Понятие NP-полноты первоначально возникло в связи с попытками доказать гипотезу $P \neq NP$: если это так, то NP-полные языки должны заведомо лежать в разности $NP \setminus P$. В настоящее время известно много примеров NP-полных задач, но ни про одну из них не удалось доказать, что она не лежит в классе P. Сейчас характеризация задачи как NP-полной (и NP-трудной тоже) воспринимается как довод в пользу невозможности ее практического решения программными средствами в столь общей постановке: написать программу вполне реально, но обеспечить эффективность ее работы на всех исходных данных не удается. Естественный выход в этой ситуации — сужение задачи, переход к частным случаям,

как раз к тем, в которых программа работает приемлемым образом[1].

§ 2. NP-полнота проблемы выполнимости булевых формул

Теорема 8.4. *Проблема выполнимости булевых формул NP-полна.*

Доказательство. Мы видели, что язык SAT, состоящий из всех записей выполнимых булевых формул, принадлежит классу NP. Достаточно доказать, что произвольный язык $L \in NP$ сводится к SAT. Общий случай сводится к следующему: язык $L \subset \{0,1\}^*$ задается полиномом p и предикатом $R \in P$ так

$$x \in L \Leftrightarrow \exists y \in \{0,1\}^*(|y| = p(|x|) \wedge R(x,y))$$

и

$$|y| \neq p(|x|) \Rightarrow \neg R(x,y).$$

В доказательстве теоремы 6.1 о включении $P \subset P/Poly$ мы предложили метод, который для произвольного фиксированного предиката из класса P по длине входа n строит булеву схему полиномиального размера, вычисляющую этот предикат на словах длины n. Этот метод оказывается эффективным в следующем смысле: само отображение

$$n \mapsto \text{булева схема } S_n$$

оказывается вычислимым за полиномиальное время (см. его описание). Применим эту конструкцию к предикату R.

[1] Разработчикам программных продуктов автор настоятельно рекомендует делать это самостоятельно и заблаговременно, не дожидаясь уличения во лжи со стороны пользователей.

Тогда схема $S_{n+p(n)}$ вычисляет значение предиката $R(x, y)$ по набору битов его аргументов $x_1, \ldots, x_n, y_1, \ldots, y_{p(n)}$.

Переименовывая рабочие переменные схемы добьемся того, чтобы переменные в левых частях операторов присваивания не повторялись. После этого сопоставим схеме булеву формулу $\varphi(x_1, \ldots, x_n, y_1, \ldots, y_{p(n)}, z_1, \ldots, z_k)$ так: каждому оператору присваивания `z <- t` схемы сопоставим булеву формулу $(z \leftrightarrow t)$, возьмем конъюнкцию их всех и еще переменной, возвращающей значение всей схемы. Переменными формулы φ являются все переменные схемы, а длина φ также ограничена сверху полиномом от n.

Обозначим через $b(v)$, $v \in \{0,1\}^n$, набор формул той же длины n с компонентами

$$b_i = \begin{cases} w \wedge \neg w & , v_i = 0, \\ w \vee \neg w & , v_i = 1, \end{cases}$$

где w — не встречавшаяся ранее булева переменная. Рассмотрим следующее отображение f:

$$v \in \{0,1\}^* \mapsto \left\{ \begin{array}{l} n := |v| \\ b := b(v) \\ S_{n+p(n)} \end{array} \right\} \mapsto \psi(w, y, z) := \varphi(b, y, z).$$

Оно вычислимо за полиномиальное время, причем

$$v \in L \Leftrightarrow \text{ формула } \psi \text{ выполнима } \Leftrightarrow f(v) \in SAT.$$

□

Замечание. Если в качестве встроенных функций языка булевых схем использовать только одноместные и двухместные операции (например, $\neg, \wedge, \vee$), то в каждом операторе присваивания будет не более трех переменных. Тогда конъюнктивные члены формулы $\psi = \bigwedge \psi_j$ также будут

содержать не более трех переменных каждый. Приведем каждый из них к КНФ. Это за полиномиальное время даст приведение формулы ψ к КНФ специального вида, в котором каждый конъюктивный член содержит не более 3 переменных (так называемые 3-КНФ). Так доказывается

Следствие 8.5. *Проблема выполнимости 3-КНФ (обозначение: $3SAT$) NP-полна.*

Замечание. Заметим, что проблемы выполнимости 2-КНФ и ДНФ принадлежат классу P. Как следствие этого получим, что если $P \neq NP$, то задача приведения 3-КНФ к виду ДНФ за полиномиальное время не решается.

§ 3. NP-полнота задачи о клике

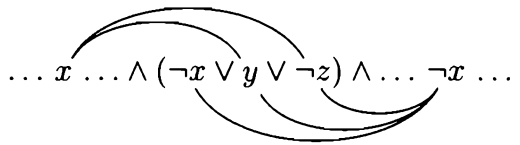
Теорема 8.6. *Задача о клике NP-полна.*

Доказательство. Язык $CLIQUE \in NP$ состоит из слов, кодирующих пары (G, d), где G — граф, d натуральное число и в графе G есть полный подграф из d вершин (клика). Достаточно доказать, что $3SAT \leqslant^p_m CLIQUE$.

Напомним, что литералом называется булева переменная или ее отрицание. 3-КНФ φ есть конъюнкция скобок, каждая из которых есть дизъюнкция не более трех литералов:

$$\varphi = \ldots \wedge (l_{i,1} \vee l_{i,2} \vee l_{i,3}) \wedge \ldots \wedge (l_{j,1} \vee l_{j,2} \vee l_{j,3}) \wedge \ldots$$

Сопоставим ей граф G, вершинами которого будут все вхождения всех литералов в формулу φ. Два вхождения соединим ребром, если они находятся в разных скобках и не являются «противоположными», т. е. не есть пара x и $\neg x$. Граф G будет выглядеть примерно так:



Для каждой клики в графе нетрудно подобрать истинностные значения переменных, обращающие все литералы клики в 1. Если при этом размер клики d равен количеству коъюнктивных членов в формуле, то вся формула обратится в 1. Верно и обратное: если для некоторой истинностной оценки переменных $\varphi = 1$, то в каждом конъюнктивном члене можно выбрать по литералу так, чтобы получилась клика (надо выбирать $l_{i,k} = 1$ по одному из скобки). То есть для функции $f(\varphi) :=$ код (G, d), где d количество дизъюнктивных членов φ, выполняется

$$\varphi \in 3SAT \Leftrightarrow f(\varphi) \in CLIQUE.$$

Сама f вычислима за полиномиальное время, если все кодирования выбраны достаточно естественно (например, когда граф кодируется матрицей смежностей, числа — двоичным представлением, формулы — своей записью). Заметим, что константа 3 в этом доказательстве не использовалась. □

§ 4. NP-трудность задачи целочисленного линейного программирования

Собственно говоря, задача линейного программирования — это задача поиска экстремума линейного функционала при наличии ограничений, выраженных системой линейных неравенств. Она не является задачей распознавания языков, поэтому формально нельзя говорить о ее NP-трудности. Но любые методы ее решения также вынуждены распознавать совместность ограничений, поэто-

му задача линейного программирования не менее трудна, чем задача распознавания совместности систем линейных неравенств. В целочисленном случае последняя задача уже оказывается (после надлежащей кодировки) задачей распознавания языка, причем NP-полной задачей. В этом смысле мы утверждаем NP-трудность задачи целочисленного линейного программирования.

Теорема 8.7. *Задача распознавания разрешимости в целых числах системы линейных неравенств с целыми коэффициентами NP-полна.*

Доказательство. Условие выполнимости КНФ можно эквивалентно переписать в виде условия разрешимости в целых числах следующей системы неравенств. Запишем условие булевозначности для каждой из переменных:

$$0 \leqslant x_j \leqslant 1,$$

а каждому конъюнктивному члену $(l_1 \vee \ldots \vee l_k)$ сопоставим неравенство

$$p_1 + \ldots + p_k > 0, \quad \text{где } p_i = \begin{cases} x_j, & \text{если } l_i \text{ есть } x_j, \\ 1 - x_j, & \text{если } l_i \text{ есть } \neg x_j. \end{cases}$$

□

ГЛАВА 9

КЛАСС BPP

§ 1. Вероятностные вычисления за полиномиальное время

Вероятностная машина Тьюринга — это машина Тьюринга с дополнительной лентой (только для чтения, головка движется только вправо), имитирующей (физический) датчик случайных чисел. Во всех ее клетках заранее записаны 0 или 1, причем предполагается, что это значения независимых испытаний случайной величины ξ с $Prob\{\xi = 0\} = Prob\{\xi = 1\} = 1/2$. Количество обращений к датчику в процессе вычисления есть величина зоны на этой ленте, в которой побывала головка; содержимое остальных клеток этой ленты не используется. Будем предполагать следующее:

- Вероятностная машина Тьюринга всегда останавливается и в качестве результата выдает 0 или 1.

- Время вычисления ограничено полиномом от длины входа. Случайная лента входной не является и учитывается при подсчете зоны вычисления. Как следствие, количество обращений к датчику случайных

чисел также ограничено сверху полиномом от длины входа.

- Вероятностная машина Тьюринга M есть модель вычисления за полиномиальное время случайной величины $\zeta_x = M(x)$ с законом распределения

0	1
$Prob\{M(x) = 0\}$	$Prob\{M(x) = 1\}$

как функции от входного слова x. Специфической мерой сложности таких вычислений является количество обращений к датчику случайных битов $Q_M(x)$ — максимум числа использованных клеток дополнительной ленты-датчика, взятый по всем вариантам вычисления на входе x.

- Случайная величина ζ_x распознает язык $L \subset \Sigma^*$ с вероятностью ошибки $\alpha(n)$, если

$$\begin{aligned} x \in L &\Rightarrow Prob\{\zeta_x = 1\} > 1 - \alpha(|x|), \\ x \notin L &\Rightarrow Prob\{\zeta_x = 1\} \leqslant \alpha(|x|). \end{aligned}$$

Определение 9.1. Язык $L \subset \Sigma^*$ принадлежит классу BPP (от «bounded probabilistic polynomial»), если существует вычислимая за полиномиальное время случайная величина, распознающая L с постоянной вероятностью ошибки $\alpha(n) = 1/3$.

§ 2. Частотные распознаватели

Оказывается удобным переформулировать определение класса BPP в комбинаторных терминах без использования понятия случайной величины. Рассмотрим полином $q(n)$, ограничивающий сверху число обращений к датчику

вероятностной машины M, распознающей язык L с вероятностью ошибки $\alpha(n)$.

Предикат $R(x, y)$

> «$|y| = q(|x|)$ и машина M на входе x с содержимым ленты-датчика y возвращает 1»

принадлежит классу P, причем

$$Prob\{M(x) = 1\} = \frac{|\{y \mid |y| = q(|x|),\ R(x, y) = 1\}|}{2^{q(|x|)}}.$$

Поэтому

$$\begin{aligned} x \in L \ &\Rightarrow \ \frac{|\{y||y|=q(|x|),\ R(x,y)=1\}|}{2^{q(|x|)}} > 1 - \alpha(|x|), \\ x \notin L \ &\Rightarrow \ \frac{|\{y||y|=q(|x|),\ R(x,y)=1\}|}{2^{q(|x|)}} \leqslant \alpha(|x|). \end{aligned} \tag{9.1}$$

Тройку (R, q, α), где $R \in P$, q — полином, а α — любая функция, для которых выполнено (9.1), условимся называть *частотным распознавателем* языка L. Мы только что установили, что если язык распознается на вероятностной машине Тьюринга с вероятностью ошибки α, то он обладает частотным распознавателем с той же α. Верно и обратное — частотный распознаватель легко переделать в вероятностную машину Тьюринга, которая будет распознавать тот же язык с той же вероятностью ошибки. Поэтому определение 9.1 эквивалентно следующему:

Определение 9.2. Язык $L \subset \Sigma^*$ принадлежит классу BPP, если он обладает частотным распознавателем (R, q, α) с $\alpha(n) = 1/3$.

Значение константы $1/3$ в этих определениях несущественно — класс не изменится, если вместо нее взять любое число c, $0 < c < 1/2$. За счет увеличения степени полинома q на 1 можно повысить надежность ответа гораздо больше:

Лемма 9.3. *Каждый язык L с частотным распознавателем $(R_0, n^d, 1/3)$ обладает также частотными распознавателями вида*

$$(R_1, n^{d+1}, (2\sqrt{2}/3)^n), \quad (R_2, n^{d+2}, (2\sqrt{2}/3)^{n^2}),$$

Доказательство. Вычисление предиката $R_1(x, z)$:

1. вычисляем $n := |x|$ и проверяем условие $|z| = n^{d+1}$ (если не равно, то результат 0);
2. разбиваем слово z на n кусков длины n^d каждый, $y_i :=$ i-й кусок;
3. $R_1(x, z) := MAJORITY(R_0(x, y_1), \ldots, R_0(x, y_n))$, где $MAJORITY$ — булева функция голосования (мнение большинства ее аргументов).

Частота правильного ответа для R_1 может быть вычислена, как вероятность более половины успехов в серии из n независимых испытаний в схеме Бернулли с вероятностью успеха $p = 2/3$. Она оценивается снизу величиной $1 - \lambda^n$, где $\lambda = 2\sqrt{p(1-p)} = 2\sqrt{2}/3$.

Конструкция R_2 аналогична, но слово z надо разбивать на n^2 кусков длины n^d каждый. □

Замечание. Возможности вероятностного распознавания языков с вероятностью ошибки 0 оказываются теми же, что и у обычного детерминированного вычисления (т. е. датчик случайных битов ничему не помогает). Это совершенно очевидно для формализма частотных распознавателей: для языка L, обладающего частотным распознавателем с нулевой вероятностью ошибки $(R, q, 0)$, справедливо

$$x \in L \Leftrightarrow \forall y_{|y|=q(|x|)} R(x, y) = 1 \Leftrightarrow R(x, 0^{q(|x|)}) = 1,$$

т. е. $L \in P$.

Замечание. То же самое имеет место и для более общего формализма вероятностных вычислений на машинах Тьюринга с дополнительной лентой-датчиком, отличающегося от приведенного выше отсутствием полиномиальных ограничений на время вычислений и число обращений к датчику случайных битов (требование к вычислению всегда заканчиваться — остается). В самом деле, если для некоторого x при некотором заполнении ленты-датчика бесконечной в обе стороны последовательностью y вероятностная машина M дает неверный ответ, то тот же (неверный) ответ она будет давать при заполнении ленты-датчика любой последовательностью z, совпадающей с y в битах с номерами $|i| \leqslant S_M(x)$. Мера множества таких последовательностей есть $2^{-2S_M(x)} > 0$. Поэтому вероятностное вычисление с нулевой вероятностью ошибки должно не давать ошибки ни при каком заполнении ленты-датчика. Это означает, что вместо случайного заполнения можно всю ленту-датчик заполнить (алгоритмически) нулями.

§ 3. Включение $BPP \subset P/Poly$

Мы видели, что каждый язык $L \in P$ обладает алгоритмом-распознавателем специального «схемного» вида:

> Для выяснения принадлежности входного слова x языку L сначала по $n = |x|$ за полиномиальное время вычисляется некоторая булева схема S_n от n входных переменных, а затем ей на вход подаются биты слова x. Результат $S_n(x)$ оказывается правильным ответом на вопрос «$x \in L$?».

Языки $L \in BPP$ удается распознавать аналогичными, но вероятностными «схемными» алгоритмами. Разница со-

стоит в том, что схема в этом случае имеет дополнительные входные переменные, $S_n = S_n(x, y)$, значения которых (т. е. y) надо задавать случайно, пользуясь датчиком. Тогда, с большой вероятностью, значение $S_n(x, y)$ совпадет с правильным ответом на вопрос «$x \in L$?».

Построить «схемный» разрешающий алгоритм для языка $L \in BPP$ можно из любого его частотного распознавателя $(R, q(n), \alpha(n))$. Достаточно в качестве $S_n(x, y)$ взять булеву схему, вычисляющую предикат $R(x, y)$ по битам слов x, $|x| = n$ и y, $|y| = q(n)$. Тогда $\alpha(n)$ будет вероятностью ошибочного ответа.

Следующая теорема показывает, что, в принципе, от ошибки можно избавиться вовсе с помощью хорошего частотного распознавателя и замены датчика случайности на специальный детерминированный выбор значений. Эффективный способ «улучшения» частотных распознавателей известен (лемма 9.3), но достаточно работоспособной методики выбора дополнительных битов нет.

Теорема 9.4. $BPP \subset P/Poly$.

Доказательство. Пусть $L \in BPP$. По лемме 9.3 для L существует частотный распознаватель вида

$$(R, q(n), (2\sqrt{2}/3)^{n^2}).$$

Обозначим через $Y(x)$ множество

$$\{y \mid |y| = q(|x|) \text{ и значение } R(x, y) \text{противоположно «} x \in L\text{»}\}.$$

Доля множества $Y(x)$ среди всех слов y длины $q(n)$ не больше $(2\sqrt{2}/3)^{n^2}$ и

$$(2\sqrt{2}/3)^{n^2} \cdot 2^n < 1 \text{ при } n > n_0.$$

Отсюда для каждого $n > n_0$ найдется двоичное слово y_n такое, что

$$|y_n| = q(n), \qquad y_n \notin \bigcup_{|x|=n} Y(x).$$

Для него выполняется

$$|x| = n > n_0 \;\Rightarrow\; (x \in L \Leftrightarrow R(x, y_n) = 1).$$

Пусть n достаточно велико. Так как $R \in P \subset P/Poly$, то существует булева схема размера $poly(|x| + |y|)$, вычисляющая предикат R по битам слова xy. При $|y| = q(|x|)$ размер этой схемы оценивается полиномом от длины x. Если взять $y = y_n$, где $n = |x|$, и добавить $q(n) + \text{const}$ операторов для вычисления битов слова y_n (оно одно для всех x длины n), то получится схема полиномиального размера, вычисляющая предикат «$x \in L$» на словах длины n. □

ГЛАВА 10

ВЕРОЯТНОСТНЫЙ АЛГОРИТМ РАСПОЗНАВАНИЯ ПРОСТЫХ ЧИСЕЛ

§ 1. Сведения из теории чисел

Факт 1. (*Следствие из китайской теоремы об остатках*). *Пусть* $n = u \cdot v$, $(u, v) = 1$. *Тогда*

$$(\mathbb{Z}_n, +, \cdot) \cong (\mathbb{Z}_u, +, \cdot) \oplus (\mathbb{Z}_v, +, \cdot)$$

и отображение $k \mapsto (k \bmod u,\ k \bmod v)$ *есть соответствующий изоморфизм.*

Факт 2. (*Малая теорема Ферма*). *Если* n *— простое и не является делителем* a, *то* $a^{n-1} \equiv 1 \pmod{n}$.

Факт 3. (*Тривиальный*). *Если* $b^2 \equiv 1 \pmod{n}$, *а* $b \not\equiv \pm 1 \pmod{n}$, *то* n *— составное.*

Доказательство. В кольце $\mathbb{Z}_n$ имеем разложение 0 в произведение ненулевых сомножителей, $(b-1)(b+1) = b^2 - 1 = 0$, поэтому $\mathbb{Z}_n$ не поле и n — составное. □

§ 2. Извлечение корней

Лемма 10.1. *Задача поиска решения уравнения*

$$x^k = n, \quad k, n \in N,$$

в натуральных числах разрешима за полиномиальное время.

Доказательство. Решаем это уравнение приближенно методом половинного деления с точностью 1. Начальное приближение корня — отрезок $[a, b] := [1, n]$. При вычислении середины отрезка округляем до целого числа, т. е. $c := [(a + b)/2]$. Для выбора очередного приближения (одного из отрезков $[a, c]$ или $[c, b]$ в зависимости от результата проверки условия «$c^k < n$») вычисляем c^k, пользуясь двоичным разложением числа k:

$$k = 2^{l_1} + \ldots + 2^{l_t}, \quad c^k = c^{2^{l_1}} \cdot \ldots \cdot c^{2^{l_t}},$$

а c^{2^l} вычисляем последовательным возведением в квадрат. При этом прерываем вычисление, если текущий результат стал больше n. Получится не более $(l_1 + \ldots + l_t + t) = O(\log\ k)$ операций умножения чисел, не превосходящих n. Количество итераций, достаточных для получения приближения точности $(b - a) = 1$ есть $O(\log\ n)$. Итого, $O((\log k + \log n)^2)$ операций. Остается проверить, не является ли одно из чисел a, b точным решением, что можно сделать таким же образом за то же время.

Метод легко программируется на языке типа BASIC. Доказательство завершается стандартным моделированием этого языка машинами Тьюринга. □

§ 3. Вероятностный алгоритм распознавания простых чисел

Вход: натуральное число $n > 2$.

Для получения правильного ответа с вероятностью ошибки $< 2^{-d}$ повторить следующую последовательность шагов d раз. Если ни разу не удалось установить, что число n — составное, то считать его простым.

1. Проверка n на четность (если четное, то составное).
2. Проверяем, что n не представимо в виде x^k для некоторых натуральных x и $k > 1$ (достаточно перебрать $k = 2, \ldots, [\log_2 n]$). Если представимо, то оно составное.
3. Представляем четное число $n - 1$ в виде $2^k \cdot l$, где $k > 0$, а l — нечетно.
4. Выбираем a случайно из чисел $1, \ldots, n - 1$.
5. Последовательно вычисляем $a^l, a^{2 \cdot l}, a^{2^2 \cdot l}, \ldots, a^{n-1}$ по модулю n и ищем такое $j < k$, что

$$a^{2^j \cdot l} \not\equiv \pm 1 \pmod{n}, \quad a^{2^{j+1} \cdot l} \equiv 1 \pmod{n}.$$

   Если нашли, то n — составное (Факт 3).
6. Шаг 5 закончился вычислением $a^{n-1} \bmod n$. Если $a^{n-1} \not\equiv 1 \pmod{n}$, то n — составное (Факт 2).

§ 4. Верификация алгоритма

Теорема 10.2. *Если число $n > 2$ — простое, то алгоритм установит это. Если число n составное, то алгоритм установит это с вероятностью, не меньшей, чем $1 - 2^{-d}$.*

Доказательство. Очевидно, что в случае простого n алгоритм никогда не объявит его составным. Неправильный ответ может получиться только в случае нечетного

составного $n = u \cdot v$, $(u, v) = 1$. Покажем, что вероятность неправильного ответа в этом случае (т. е. объявления данного составного числа n простым) при одной итерации шагов 1–6 не превосходит 1/2. Тогда после d итераций получим вероятность ошибки не более 2^{-d}.

Пусть $\equiv$ означает сравнение по модулю n. Заметим, что если $(a, n) > 1$ для выбранного на шаге 4 числа a, то $a^{n-1} \not\equiv 1$, так как при $a = a_1 \cdot c$, $n = n_1 \cdot c$

$$a^{n-1} \equiv 1 \quad \Rightarrow \quad n_1 \equiv a_1^{n-1} \cdot c^{n-1} \cdot n_1 \equiv 0.$$

На шаге 6 это будет обнаружено и алгоритм даст правильный ответ. Поэтому достаточно установить, что по крайней мере для половины чисел из множества

$$G = \{a \mid 1 \leqslant a < n,\ (a, n) = 1\}$$

шаги 5, 6 также приведут к правильному ответу (т. е. обнаружат непростоту n). Заметим, что G состоит в точности из всех обратимых элементов кольца $\mathbb{Z}_n$, т. е. является мультипликативной группой этого кольца.

Для абелевой группы $(H, \cdot)$ обозначим

$$H^i = \{x^i \mid x \in H\}.$$

Тогда H^i — ее подгруппа, а отображение $x \mapsto x^i$ есть гомоморфизм H на H^i. При этом мощность множества $\{x \in H \mid x^i = h\}$ одна и та же для всех $h \in H^i$. Заметим также, что $H^i \supset (H^i)^2 = H^{2i}$.

Пусть U и V — мультипликативные группы колец вычетов $\mathbb{Z}_u$ и $\mathbb{Z}_v$ соответственно. Из Факта 1 следует, что $G \cong U \times V$ посредством отображения

$$\varphi(x) = (x \bmod u,\ x \bmod v).$$

Рассмотрим убывающие последовательности подгрупп

$$G^l \supset G^{2 \cdot l} \supset G^{2^2 \cdot l} \supset \ldots \supset G^{2^k \cdot l} = G^{n-1} \supset \{1\},$$

$$U^l \supset U^{2\cdot l} \supset U^{2^2\cdot l} \supset \ldots \supset U^{2^k\cdot l} = U^{n-1} \supset \{1\},$$
$$V^l \supset V^{2\cdot l} \supset V^{2^2\cdot l} \supset \ldots \supset V^{2^k\cdot l} = V^{n-1} \supset \{1\},$$
$$\varphi(G^{2^j\cdot l}) = U^{2^j\cdot l} \times V^{2^j\cdot l}, \quad j = 0, \ldots, k.$$

При выборе каких $a \in G$ шаги 5, 6 не допускают ошибки, т. е. объявляют составное число $n = u \cdot v$ таковым? Это происходит, когда a есть решение одного из уравнений

$$\text{шаг 5:} \quad x^{2^j\cdot l} \equiv g, \quad \text{где} \quad g \in G^{2^j\cdot l},\ 0 \leqslant j < k,\ g \not\equiv \pm 1,\ g^2 \equiv 1, \tag{10.1}$$

$$\text{шаг 6:} \quad x^{n-1} \equiv g, \quad \text{где } g \in G^{n-1},\ g \neq 1. \tag{10.2}$$

Сам набор уравнений зависит от n.

Случай 1: $U^{n-1} \neq \{1\}$ или $V^{n-1} \neq \{1\}$. Тогда $G^{n-1} \cong \cong U^{n-1} \times V^{n-1}$ также содержит элементы, отличные от 1. Семейство множеств

$$M_g = \{x \in G \mid x^{n-1} = g\}, \quad g \in G^{n-1}$$

образует разбиение G на части по $|G|/|G^{n-1}|$ элементов каждая. Среди элементов $a \in G$ только элементы M_1 не удовлетворяют ни одному из уравнений (10.2). Частей не менее двух. Поэтому не менее половины элементов G удовлетворяют хотя бы одному из уравнений (10.2). Если на шаге 4 будет выбран один из них, то на шаге 6 число n будет объявлено составным (правильный ответ).

Случай 2: $U^{n-1} = V^{n-1} = \{1\}$. Заметим, что $U^l \neq \{1\}$ и $V^l \neq \{1\}$, так как l — нечетное и -1 принадлежит им обоим (под -1 понимается соответствующий вычет по модулю n, т. е. $-1 \equiv (n-1)$). Поэтому

$$j_0 = \min\{s \mid U^{2^s\cdot l} = V^{2^s\cdot l} = \{1\}\} - 1 \geqslant 0.$$

Пусть $t = 2^{j_0} \cdot l$. Множества G^t, U^t, V^t являются членами рассматриваемых последовательностей.

Подслучай 2.1: одно из множеств U^t, V^t есть $\{1\}$. В этом случае $-1 \notin G^t$, так как $(-1,-1) \notin U^t \times V^t$ (изоморфизм φ переводит G^t в $U^t \times V^t$, а $\varphi(-1) = (-1,-1)$). Рассуждаем аналогично случаю 1. Рассмотрим разбиение множества G на равномощные части

$$M_g = \{x \in G \mid x^t = g\}, \quad g \in G^t. \tag{10.3}$$

Частей не менее двух и только элементы одной из них (M_1) не удовлетворяют ни одному из уравнений (10.1) при $j = j_0$. Поэтому по крайней мере для половины элементов $a \in G$ верно, что если на шаге 4 будет выбран именно a, то на шаге 5 обнаружится, что $a^{2^{j_0} \cdot l} \not\equiv \pm 1$, $a^{2^{j_0+1} \cdot l} \equiv 1$, и число n будет объявлено составным.

Подслучай 2.2: $|U^t| > 1$, $|V^t| > 1$. Тогда $|G^t| = |U^t| \times \times |V^t| \geqslant 4$. Это означает, что в разбиении (10.3) частей не менее четырех и только элементы одной или двух из них (M_1 и может быть M_{-1}) не удовлетворяют ни одному из уравнений (10.1) при $j = j_0$. Далее повторяем те же рассуждения. □

§ 5. Оценка сложности

Теорема 10.3. *Задача распознавания простых чисел принадлежит классу* BPP[1].

Доказательство. Оценим временную сложность вероятностного алгоритма распознавания простых чисел. Число итераций d считаем фиксированным. Достаточно показать, что шаги 1–6 моделируются вероятностной машиной Тьюринга за время, оцениваемое сверху полиномом от $\log n$. Это очевидно для всех шагов, кроме шага 4, где

[1] См. сноску на стр. 65.

требуется вычислить значение случайной величины ξ с распределением

$$Prob\{\xi = a\} = 1/N, \quad a = 1, \ldots, N$$

(при $N = n-1$). Вероятностная машина Тьюринга снабжена лентой-датчиком случайных битов, что позволяет легко моделировать лишь распределения с двоично-рациональными вероятностями значений. Здесь же число N может быть произвольным.

В предыдущем разделе мы показали, что алгоритм на входе n допускает ошибку только тогда, когда выбранное значение a принадлежит некоторому (определяемому по n) множеству M, причем $|M| \leqslant N/2$. Отсюда была получена оценка сверху вероятности ошибки $(Prob\{\xi \in M\})^d \leqslant (1/2)^d$. Если взять любое другое распределение случайной величины ξ, то оценка

$$(\text{Вероятность ошибки}) \leqslant (Prob\{\xi \in M\})^d$$

сохранится. Поэтому для доказательства теоремы достаточно научиться вычислять на вероятностной машине Тьюринга за время $poly(\log N)$ значение какой-нибудь случайной величины $\xi' \in \{1, \ldots, N\}$ с $Prob\{\xi' \in M\} \leqslant p < 1$ (p не зависит от N), а затем выбрать в алгоритме параметр d из условия $p^d < 1/3$.

Алгоритм вычисления такой случайной величины ξ':

> Вход: число N. Выбираем число m из условия $2^{m-1} < N \leqslant 2^m$ и с помощью датчика случайных битов порождаем случайное двоичное слово v длины m. Пусть число k таково, что v есть двоичная запись числа $k-1$ (с лидирующими нулями). Если $k \leqslant N$, то результатом объявляем k, а в противном случае — число 1.

Величина ξ' принимает значения из множества $\{1, \ldots, N\}$, причем вероятность каждого значения не меньше 2^{-m}. Отсюда

$$Prob\{\xi' \in M\} \leqslant 1 - \frac{N}{2} \cdot 2^{-m} \leqslant 3/4.$$

Количество обращение к датчику случайных битов и общее время работы этого алгоритма есть $O(\log N)$. Если шаг 4 в алгоритме распознавания простых чисел реализовать таким образом, то d можно взять равным 4. □

ГЛАВА 11

КОНЕЧНЫЕ ИГРЫ И КЛАСС PH

§ 1. Конечные игры

Играют два игрока (A) и (M). Начальная конфигурация игры задается входным словом $x \in \{0,1\}^*$. Правила игры специфицируются следующими параметрами:

- фиксированным (конечным) количеством ходов k и указанием, кто начинает;
- полиномом p, задающим длину одного хода как функцию от $|x|$; ход — это слово в алфавите $\{0,1\}$ длины $p(|x|)$;
- предикатом $R \in P$, задающим условие выигрыша (M) при данной последовательности ходов w_1, b_1, w_2, b_2, ...,

$$R = R(x, w_1, b_1, w_2, b_2, \ldots).$$

(Количество аргументов фиксировано, оно на единицу больше числа ходов.)

Игроки по очереди обмениваются ходами. Каждый ход есть слово длины $p(|x|)$. После завершения партии с помощью предиката R вычисляется, кто выиграл. Такая игра

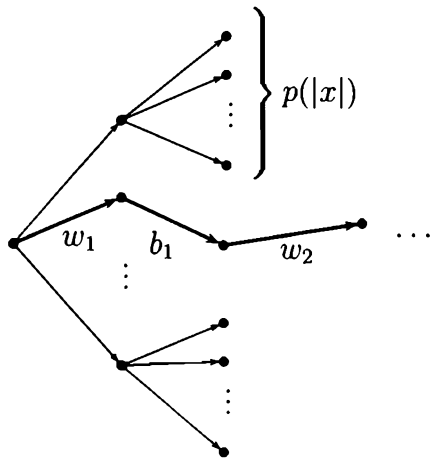


Рис. 11.1. Дерево игры.

называется *конечной игрой* или *ограниченным детерминированным интерактивным протоколом.*

Совокупность всех вариантов розыгрыша, которые начинаются с данной начальной конфигурации x, удобно представлять себе в виде *дерева игры.* Это дерево порядка ветвления $2^{p(|x|)}$, вершины которого соответствуют всевозможным «промежуточным» конфигурациям, т. е. состояниям игры, возникающим после i произвольных ходов, $i \leqslant k$.

§ 2. Определение класса PH

Пусть начальная конфигурация x фиксирована. Множество всевозможных партий конечно, поэтому для одного из игроков обязательно существует выигрышная стратегия — функция, определяющая его очередной ход по всем предыдущим ходам так, что если руководствоваться ей при выборе ходов, то игрок обязательно выиграет.

Пример. Если ходов 2 и начинает (A), то выигрышная стратегия F для (M) — это функция, которая по каждому возможному ходу w_1 игрока (A) вычисляет ответный ход b_1 для игрока (M). Она должна удовлетворять условию

$$\forall w_{|w|=p(|x|)} R(x, w, F(w)).$$

Такая функция существует, если

$$\forall w_{|w|=p(|x|)} \exists b_{|b|=p(|x|)} R(x, w, b).$$

Для той же игры выигрышная стратегия для (A) — это функция, которая по пустому слову (игрок (A) ходит первым) вычисляет правильный первый ход. Она должна удовлетворять условию

$$\forall b_{|b|=p(|x|)} \neg R(x, F(\,), b).$$

Выигрышная стратегия для (A) существует, если

$$\exists w_{|w|=p(|x|)} \forall b_{|b|=p(|x|)} \neg R(x, w, b).$$

Легко видеть, что условия существования этих двух стратегий взаимно дополняющие.

Определение 11.1. Язык $L \in \{0,1\}^*$ распознается игрой, если условие «$x \in L$» равносильно «для начальной конфигурации x игрок (M) обладает выигрышной стратегией». Семейство всех таких языков образует класс PH.

Замечание. Обозначение для класса PH происходит от термина полиномиальная иерархия (Polynomial Hierarchy), обсуждаемого в следующей лекции. Собственно полиномиальная иерархия есть набор сложностных классов, структурирующих класс PH, а элементы $L \in PH$ — это

все языки, сложность которых может быть измерена с помощью полиномиальной иерархии.

Интерпретация. Игроки — юный Артур и его учитель Мерлин. Великий Мерлин обладает неограниченными вычислительными возможностями, в частности, может вычислять выигрышные стратегии, когда они существуют. Он знает, что $x \in L$ и хочет передать это знание Артуру. Возможности Артура меньше — доступные ему вычисления полиномиально ограничены по времени. При этом он не вполне доверяет Великим и хочет получить не только сам факт, но и некоторое доступное ему подтверждение. Но он — Избранный и обычно добивается успеха, если только это возможно. За него фактически играет Провидение, которое не слабее Мерлина. Собственных сил Артуру хватает лишь на то, чтобы проследить за правильностью присуждения выигрыша после окончания игры. В такой ситуации (пока Провидение не покинуло Артура) в качестве подтверждения Мерлину оказывается достаточным выиграть у Артура в соответствующей игре, хотя на самом деле Артур получает знание

«если он все еще Избранный, то $x \in L$»

и вынужден вечно сомневаться.

Например, они изучают тактику захвата замков. Начальной конфигурацией игры служит план замка. Предполагается, что способов защиты и нападения экспоненциально много. Мерлин имитирует защитников и старается убедить Артура, что замок неприступен. Он расставляет силы защитников, а Артур предлагает план атаки, после чего разыгрывается сражение (за полиномиальное время). Получается двухходовая игра, распознающая предикат «замок x неприступен».

Замечание. Две игры будем считать эквивалентными, если они распознают один и тот же язык. Легко видеть,

что за счет добавления одного хода в начале можно преобразовать любую ограниченную игру в эквивалентную, в которой начинает заданный игрок, например, (М). Соответствующий предикат выигрыша просто будет зависеть от добавленного хода фиктивно. Аналогично можно управлять выбором последнего хода. Можно также при необходимости повысить степень полинома, задающего длину хода, и/или количество ходов, так как лишние биты может отбрасывать «сам» предикат выигрыша.

Замечание. Следующие модификации правил игры легко моделируются исходными (за счет модификации предиката выигрыша): (а) длина хода не в точности равна, а не превосходит величины $p(|x|)$; (б) количество ходов не фиксировано, а зависит от входного слова, но ограничено сверху константой. Более точно это означает, что если язык L распознается модифицированной игрой, то существует стандартная конечная игра, которая также распознает L. Идеи моделирования таковы:

(а) Длина хода в соответствующей стандартной игре будет n^{c+1}, где c есть степень многочлена p. Ход $v = v_1 \dots v_k \in \{0,1\}^*$, $|v| \leqslant p(|x|)$ модифицированной игры моделируется в стандартной игре ходом

$$v_1 v_1 \dots v_k v_k 01 \underbrace{00 \dots 0}_{|x|^{c+1}-2k-2} .$$

«Стандартный» предикат выигрыша сначала по своим аргументам восстанавливает последовательность ходов модифицированной игры, а затем вычисляет значение «модифицированного» предиката выигрыша на этой последовательности.

(б) При такой модификации предикат выигрыша имеет два аргумента — начальную конфигурацию x и по-

следовательность ходов $w_1 b_1 w_2 b_2 \ldots$, записанную одним словом без разделителей. Его значение вычисляется после каждого хода, а не только в конце партии. Окончание партии происходит, когда значение предиката выигрыша после очередного хода стало 1 или когда число ходов достигло максимального, допустимого для данной игры (k). В соответствующей стандартной игре все партии — по k ходов. «Стандартный» предикат выигрыша определяется следующим образом:

$$\underbrace{R(x, w_1) \vee R(x, w_1 b_1) \vee R(x, w_1 b_1 w_2) \vee \ldots}_{k \text{ раз}} .$$

§ 3. Замкнутость относительно $\cap$, $\cup$ и $(\cdot)^c$

Лемма 11.2. *Если* $L_1, L_2 \in PH$, *то* $L_1 \cap L_2, L_1 \cup L_2, L_1^c \in PH$.

Доказательство. Случай $\cap$. Для языков L_1, L_2 выберем распознающие их игры так, чтобы длины ходов в них были одинаковыми, а последний ход первой делал не тот игрок, который начинает вторую. Последовательность ходов для игры, распознающей $L_1 \cap L_2$, будет состоять из последовательности ходов первой игры $\overline{\omega_1}$, которая продолжается последовательностью ходов второй игры — $\overline{\omega_2}$. В качестве предиката выигрыша следует взять $R_1(x, \overline{\omega_1}) \wedge R_2(x, \overline{\omega_2})$, где R_1, R_2 — предикаты выигрыша первой и второй игр.

Случай $\cup$. Для языков L_1, L_2 выберем распознающие их игры так, чтобы у них совпадали длины и количества ходов, причем в обеих играх первый ход делал (A). Количество ходов в игре для $L_1 \cup L_2$ будет на единицу больше, а дополнительный первый ход z будет делать (M). Предикат

выигрыша $R(x, z, \overline{\omega})$ будет

$$((z = 0 \ldots 0) \wedge R_1(x, \overline{\omega})) \vee ((z \neq 0 \ldots 0) \wedge R_2(x, \overline{\omega})).$$

Случай дополнения. Достаточно передать право первого хода другому игроку, а в качестве предиката выигрыша взять $\neg R_1(x, \overline{\omega})$. $\square$

ГЛАВА 12

ПОЛИНОМИАЛЬНАЯ ИЕРАРХИЯ

§ 1. Классы полиномиальной иерархии

Условимся писать

$$\exists^p z \quad \text{вместо} \quad \exists z_{|z|=p(|x|)},$$
$$\forall^p z \quad \text{вместо} \quad \forall z_{|z|=p(|x|)}.$$

Определение 12.1. Рассмотрим логическую структуру условия существования выигрышных стратегий для (М) в n-ходовой игре (т. е. условия «$x \in L$»). Если в игре начинает (М), то оно имеет вид

$$\exists^p w_1 \forall^p b_1 \exists^p w_2 \ldots R(x, w_1, b_1, w_2, \ldots), \quad R \in P, \qquad (12.1)$$

где количество кванторов равно n. Такие игры (полином p и предикат $R \in P$ — любые) называются Σ_n^p-играми, а соответствующие языки образуют класс полиномиальной иерархии Σ_n^p. Для игры, в которой начинает (А), соответствующее условие будет

$$\forall^p w_1 \exists^p b_1 \forall^p w_2 \ldots R(x, w_1, b_1, w_2, \ldots), \quad R \in P \qquad (12.2)$$

с кванторной приставкой длины n. Эти игры называются Π_n^p-играми, а соответствующие языки составляют класс Π_n^p.

Замечание. Само понятие игры в определении классов полиномиальной иерархии формально не является необходимым — класс Σ_n^p состоит из всех предикатов вида (12.1), а класс Π_n^p — вида (12.2), где n есть длина кванторной приставки. Заметим также, что верхний индекс p в обозначениях классов происходит от слова «polynomial» и не обозначает конкретный полином. Например, в класс Σ_n^p попадают все предикаты вида (12.1) для всевозможных полиномов $p(n) = n^c$, $c = 1, 2, \ldots$

Замечание. Эквивалентные логические описания этих классов получатся также в следующих случаях:

- если вместо кванторов $\exists^p$, $\forall^p$ использовать (всюду или в некоторых местах) $\exists z_{|z|<p(|x|)}$, $\forall z_{|z|<p(|x|)}$;
- если разрешить в формулах (12.1), (12.2) несколько одноименных кванторов подряд (в этом случае параметр n есть количество чередований кванторов);
- и то и другое одновременно.

Замечание. Для доказательства принадлежности языка L классу полиномиальной иерархии достаточно представить предикат «$x \in L$» в виде (12.1) или (12.2) не для всех, а только для достаточно длинных слов x, $|x| > n_0$. Такое «неполное» представление легко переделать в «полное», переопределив значение предиката R для коротких слов x следующим образом:

$$R(x, \ldots) := \begin{cases} 1, & \text{если } |x| \leqslant n_0 \text{ и } x \in L, \\ 0, & \text{если } |x| \leqslant n_0 \text{ и } x \notin L. \end{cases}$$

§ 2. Структурные свойства классов полиномиальной иерархии

Структурные свойства классов Σ_n^p и Π_n^p собраны в следующей лемме и отражены на рис. 12.1

Лемма 12.2. (*Структура PH*).

1) $\Pi_n^p = co\text{-}\Sigma_n^p$;

2) $\Sigma_n^p \cup \Pi_n^p \subset \Sigma_{n+1}^p \cap \Pi_{n+1}^p$;

3) $\Sigma_0^p = \Pi_0^p = P$; $\Sigma_1^p = NP$;

4) $\bigcup_n(\Sigma_n^p \cup \Pi_n^p) = \bigcup_n \Sigma_n^p = \bigcup_n \Pi_n^p = PH$.

Доказательство. Непосредственно следует из определений. □

Замечание. В лемме речь идет про нестрогое включение. Про любую пару классов полиномиальной иерархии (кроме $\Sigma_0^p = \Pi_0^p$) неизвестно, совпадают они или нет. В настоящее время наиболее правдоподобной представляется гипотеза о невырожденности полиномиальной иерархии, утверждающая, что все эти классы различны. Из нее, в частности, вытекает, что $P \neq NP$.

§ 3. Пример

Рассмотрим следующий пример использования полиномиальной иерархии для получения (верхней) оценки сложности конкретной задачи.

Задача. Вершины единичного $2n$-мерного куба раскрашены в два цвета — белый и черный. По данной раскраске выяснить, имеется ли в кубе n-мерная грань, все вершины которой — белые.

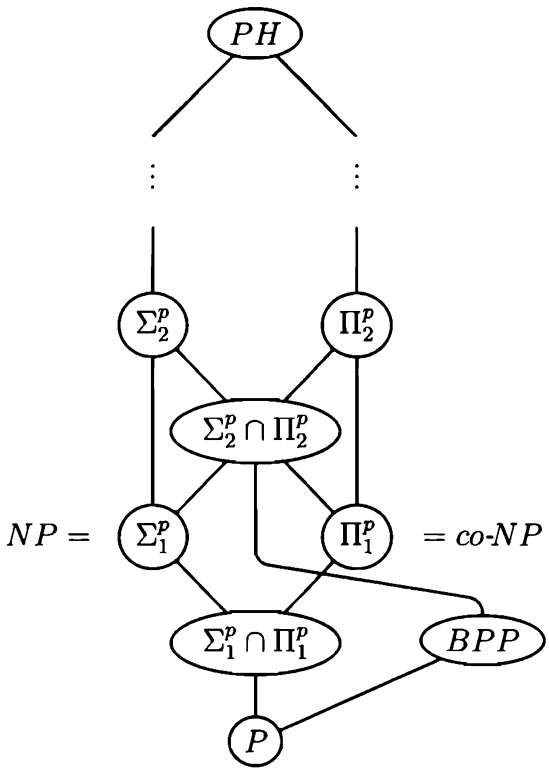
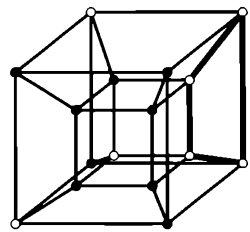


Рис. 12.1. Структура классов полиномиальной иерархии.



Координаты вершин единичного куба — числа 0 или 1. Для задания раскраски уместно использовать булеву формулу $\varphi(x_1, \ldots, x_{2n})$, для которой

$$\varphi(x_1, \ldots, x_{2n}) = 1 \quad \Leftrightarrow \quad \text{вершина } (x_1, \ldots, x_{2n}) \text{ — белая.}$$

Будем дополнительно предполагать, что все $2n$ переменных встречаются в формуле и ее размер не превосходит фиксированного полинома от n.

Рассмотрим язык L, состоящий из всех представленных таким образом раскрасок, для которых белая n-мерная грань существует. Покажем, что уточненная так задача попадает в класс Σ_2^p, т. е. что $L \in \Sigma_2^p$.

Двухходовая игра класса Σ_2^p, которая распознает язык L, такова. Своим первым ходом, (M) предлагает систему уравнений

$$\begin{cases} x_{i_1} = a_1, \\ \vdots \\ x_{i_n} = a_n, \end{cases} \tag{12.3}$$

которая, по его мнению, задает белую n-мерную грань. Ответный ход (A) состоит в выборе значений для всех остальных координат x_j, $j \neq i_1, \ldots, i_n$. После пары ходов вычисляется значение $\varphi(x_1, \ldots, x_{2n})$. Если оно равно 1, то выигрывает (M); в противном случае выигрывает (A).

Для (М) единственный способ (стратегия) обеспечить себе гарантированный выигрыш состоит в указании системы уравнений, которая на самом деле задает белую грань. Это возможно тогда и только тогда, когда такая грань существует. Таким образом, эта игра действительно распознает язык L.

Замечание. Следует иметь ввиду, что другие уточнения постановки задачи могут оказаться неэквивалентными рассмотренной, а потому иметь совершенно другую сложность. Например, если раскраски подавать на вход распознающему алгоритму в виде списка всех черных вершин (вершины задаются своими координатами), то очевидным улучшением оценки будет класс $NP = \Sigma_1^p$. Недетерминированному полиномиально ограниченному по времени допускающему алгоритму достаточно «угадать» систему уравнений (12.3), задающую белую n-мерную грань, а затем (детерминировано) проверить, что ни одна из вершин списка не принадлежит этой грани. В случае двухходовой игры похожая проверка имитировалась ходом (А), когда фактический перебор осуществляло Провидение, а (А) лишь оглашал самый худший для (М) вариант. Здесь же перебор заменяется одноразовым просмотром исходных данных, поэтому проверку может осуществить алгоритм, затратив на это время, ограниченное полиномом от их длины. (При аккуратной организации хранения системы (12.3) можно добиться линейной оценки времени.)

§ 4. Включение $BPP \subset \Sigma_2^p \cap \Pi_2^p$

Теорема 12.3. $BPP \subset \Sigma_2^p \cap \Pi_2^p$.

Сведение. Учитывая дополнительность классов Σ_2^p и Π_2^p достаточно установить, что для каждого языка $L \in BPP$ выполняется $L \in \Sigma_2^p$ и $L^c \in \Sigma_2^p$. Но так как сам

класс BPP замкнут относительно дополнений, то остается установить следующий факт:

Теорема 12.4. $BPP \subset \Sigma_2^p$.

Доказательство. Пусть язык $L \in BPP$ и (R, p, α) — его частотный распознаватель — первый из построенных в лемме 9.3, т. е.

$$p(n) = n^{d+1}, \quad \alpha(n) = \lambda^n, \quad 0 < \lambda < 1.$$

Для достаточно длинных слов x надо представить условие «$x \in L$» в виде эквивалентного, имеющего форму

$$\exists^q w \forall^q b \, Q(x, w, b),$$

где q — некоторый полином, а предикат Q принадлежит классу P. Вместо этого мы построим условие вида

$$\exists^p g_1 \ldots \exists^p g_k \forall^p g \, Q'(x, g_1, \ldots, g_k, g), \quad Q' \in P,$$

где величина k есть полином от длины x. Его легко свести к требуемому: $q(n) := k \cdot p(n)$, $w := g_1 \ldots g_k$, $b := g0 \ldots 0$, а предикат Q «сам» вычисляет k и разрезает w и b на нужные куски.

Обозначим через $G = G(x)$ множество всех двоичных слов длины $p(|x|)$. Тогда кванторы $\exists^p$ и $\forall^p$ есть просто кванторы по переменным, пробегающим G. Введем на G структуру абелевой группы с операцией $+$ — поразрядным сложением по модулю 2. Обозначим

$$Y = Y(x) = \{y \in G \mid R(x, y)\}$$

множество всех «свидетелей», используемых частотным распознавателем для проверки условия «$x \in L$». Мы покажем, что при подходящем выборе параметров искомое

представление есть

$$x \in L \Leftrightarrow (\exists g_1 \in G) \ldots (\exists g_k \in G)(\forall g \in G) \bigvee_{i=1}^{k} (g \in g_i + Y). \tag{12.4}$$

Сначала убедимся, что (при условии полиномиальности k) в правой части (12.4) под кванторами стоит предикат из класса P:

$$Q'(x, g_1, \ldots, g_k, g) \Leftrightarrow \bigvee_{i=1}^{k} (g \in g_i + Y) \Leftrightarrow \bigvee_{i=1}^{k} R(x, g - g_i).$$

Теперь займемся выбором параметров, обеспечивающим верность (12.4) и полиномиальность k. Правая часть (12.4) эквивалентна условию

$$\exists g_1, \ldots, g_k \in G \left(\bigcup_{i=1}^{k} (g_i + Y) = G \right), \tag{12.5}$$

утверждающему, что с помощью k сдвигов множества Y можно покрыть всю группу G. Естественно ожидать, что это условие выполнено, если доля $\delta = |Y|/|G|$ достаточно велика, и не выполнено, если мала.

Лемма 12.5. *Если $k \cdot \delta < 1$, то условие* (12.5) *не выполнено. Если $|G| \cdot (1 - \delta)^k < 1$, то условие* (12.5) *выполнено.*

Доказательство. (Специфика выбора группы не используется.) Легко видеть, что

$$\left| \bigcup_{i=1}^{k} (g_i + Y) \right| \leqslant k \cdot \delta \cdot |G|$$

откуда следует первое утверждение.

Для доказательства второго утверждения достаточно установить, что (при указанных условиях) если в качестве g_i взять значения, полученные в результате независимых испытаний случайной величины, принимающей произвольные значения из G с равными вероятностями $1/|G|$, то

$$Prob\{ \bigcup_{i=1}^{k}(g_i + Y) = G \} > 0.$$

Оценим вероятность противоположного события, учитывая, что $|g_i + Y| = |Y|$, а события «$g \notin (g_i + Y)$» для фиксированного g и различных i независимы:

$$Prob\left\{ \bigvee_{g \in G} \left(g \notin \bigcup_{i=1}^{k}(g_i + Y) \right) \right\} \leqslant |G| \cdot (1-\delta)^k < 1.$$

□

В нашем случае величина δ зависит от входного слова x и оценивается функцией от его длины $n = |x|$:

$$\begin{array}{lllll} x \notin L & \Rightarrow & \delta < \lambda^n & \Rightarrow & k \cdot \delta < k \cdot \lambda^n \\ x \in L & \Rightarrow & \delta > 1 - \lambda^n & \Rightarrow & |G| \cdot (1-\delta)^k < 2^{p(n)} \cdot \lambda^{k \cdot n} \end{array}$$

Положим $k = p(n)$. При достаточно больших n будет $k \times$ $\times \lambda^n < 1$ и $2^{p(n)} \cdot \lambda^{k \cdot n} < 1$. По лемме 12.5 отсюда следует, что эквивалентность (12.4) справедлива для всех достаточно длинных слов x. □

ГЛАВА 13

КЛАСС $PSPACE$

§ 1. Класс $PSPACE$ и игры с полиномиальным числом ходов

Определение 13.1. Класс $PSPACE$ состоит из всех языков $L \subset \Sigma^*$, распознаваемых на машинах Тьюринга с полиномиальным ограничением на зону вычисления, т. е. для которых существует машина Тьюринга M, вычисляющая значения предиката «$v \in L$» на словах $v \in \Sigma^*$, причем

$$S_M(n) = \max_{|v|=n} S_M(v)$$

есть функция полиномиального роста.

Это определение полностью аналогично определению класса P, поэтому к нему также следует отнести все соответствующие комментарии из лекции 5. В то же время класс $PSPACE$ гораздо богаче класса P и, по-видимому, уже содержит в себе большинство практически значимых вычислительных задач.

Класс $PSPACE$ содержит в себе все рассмотренные ранее сложностные классы (P, NP, BPP, PH) кроме «неравномерного» класса $P/Poly$. Это вытекает из следующего игрового описания класса $PSPACE$.

Оказывается, что все языки из $PSPACE$ и только они могут быть описаны конечными играми, в которых число ходов не фиксировано, а задается полиномом от длины начальной конфигурации. Такая игра описывается двумя полиномами p, q и предикатом выигрыша $R(x, \overline{\omega})$, принадлежащим классу P. Как и раньше, $x \in \{0,1\}^*$ — начальная конфигурация, а $\overline{\omega} = w_1 b_1 w_2 b_2 \ldots \in \{0,1\}^*$ — слово, составленное из чередующихся ходов двух игроков (М) и (А). Для данного x длины ходов одинаковы, $|w_i| = |b_i| = p(|x|)$, и количество ходов $q(|x|)$ — тоже. (Хотя, как обсуждалось раньше, эти условия можно ослабить без изменения класса распознаваемых языков до $|w_i| = |b_i| < p(|x|)$ и количества ходов $< q(|x|)$.) Для определенности будем считать, что первый ход всегда делает (М). Язык $L \subset \{0,1\}^*$ распознается игрой, если он состоит из всех начальных конфигураций, для которых у (М) есть выигрышная стратегия, т. е.

$$x \in L \Leftrightarrow \underbrace{\exists^p w_1 \forall^p b_1 \ldots}_{q(|x|)} R(x, w_1 b_1 \ldots).$$

Легко видеть, что игры с полиномиальным числом ходов моделируют ограниченные игры (из определения PH) — им соответствуют предикаты выигрыша, существенно зависящие только от фиксированного числа начальных ходов. Поэтому все языки из PH распознаются и такими играми. Ниже термин игра будет означать именно игры с полиномиальным числом шагов.

§ 2. Моделирование игры

Пусть даны игра (p, q, R) и начальная конфигурация x. Рассмотрим *дерево игры* (см. рис. 11.1). Оно состоит из

всевозможных $q(|x|)$-элементных последовательностей ходов $w_1 b_1 w_2 b_2 \ldots$ (ходы соответствуют ребрам; ребра ориентированы от корня к листьям). Каждый путь $\overline{\omega}$ из корня к листьям задает некоторую партию. На соответствующем листе запишем ее результат, т. е. значение $R(x, \overline{\omega})$.

Как выяснить существование выигрышной стратегии для (М): Все ребра-ходы (М) назовем $\vee$-ребрами, а все остальные — $\wedge$-ребрами. В вершинах, из которой выходят $\vee$-ребра, поместим функциональный элемент *OR*, а в вершинах, из которой выходят $\wedge$-ребра, поместим *AND*. Получим схему из функциональных элементов *AND, OR*, арности $2^{p(|x|)}$ каждый. Пометки на листьях (значения предиката R) будем считать входными и продолжим эту разметку на все вершины дерева согласно схеме. Если в корне получится 1, то у (М) есть выигрышная стратегия; если 0 — нет.

Как (М) должен играть, если в корне стоит 1 : Он каждый раз должен выбирать то ребро, которое ведет в вершину с 1.

Прямая реализация этого метода распознавания языков приводит к экспоненциальным затратам памяти на хранение дерева игры. Но на самом деле можно обойтись и полиномиальной памятью, реализовав вычисление булевых пометок обходом дерева «влево-вниз».

§ 3. Моделирование на полиномиальной памяти

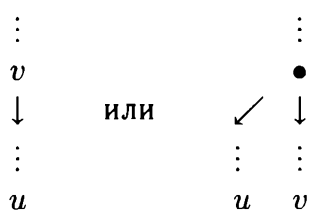
Теорема 13.2. *Если язык L распознается игрой с полиномиальным количеством ходов, то $L \in PSPACE$.*

Доказательство. Достаточно показать, что вычисление значения построенной выше $\{AND, OR\}$-схемы мож-

но реализовать на полиномиальной зоне, не распределяя память под всю схему целиком.

Итак, схема представляет собой корневое дерево высоты $q(|x|)$ с порядком ветвления $2^{p(|x|)}$. Вершины одного яруса — одинаковые функциональные элементы (либо все — AND, либо все — OR), причем AND и OR ярусы чередуются, а в корне стоит OR. Входные значения (на листьях) вычисляются на полиномиальной зоне с помощью предиката R, если известен путь из корня к листу. Предполагаем, что ребра, выходящие из одной вершины, упорядочены (слева направо).

На множестве всех вершин зададим линейный порядок «левее или ниже»: $u < v$ если путь из корня в u есть собственное продолжение пути из корня в v или есть собственное ответвление от этого пути влево:



Предлагается последовательно вычислять значения схемы в вершинах дерева в этом порядке (от левой нижней вершины к корню). При этом для вычисления значений в вершинах $w > u$, достаточно знать значения в вершине u и всех вершинах $v < u$, расположенных непосредственно ниже одной из вершин на пути из корня к u.

Размер подграфа на рис. 13.1 остается экспоненциальным за счет групп вершин, обозначенных тремя жирными точками. Но каждая такая группа расположена на одном ярусе, поэтому в значения схемы во всех вершинах $w > u$ войдет $AND(\bullet\bullet\bullet)$ или $OR(\bullet\bullet\bullet)$ в зависимости от четности номера яруса. Таким образом, в каждый момент времени

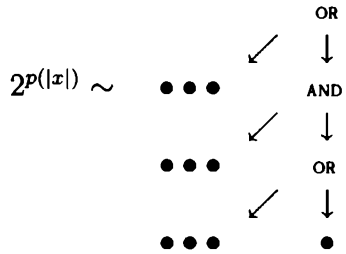
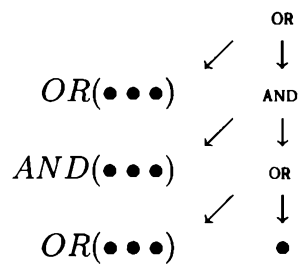


Рис. 13.1. Путь от корня к вершине u вместе с непосредственными предшественниками всех входящих в него вершин. Вершины u и $v <$ $< u$ отмечены жирными точками.

достаточно хранить подграф полиномиального размера:



□

§ 4. Игровая характеризация класса $PSPACE$

Теорема 13.3. *Каждый язык $L \in PSPACE$ распознается некоторой игрой с полиномиальным числом ходов.*

Доказательство. Пусть машина Тьюринга M распознает язык L на зоне, ограниченной полиномом p. По теореме 2.1 ее время работы можно оценить сверху функцией $2^{q(n)}$, где q некоторый полином. Для данного входного слова x рассмотрим последовательность мгновенных конфигураций $C(t)$, $t = 0, \ldots, 2^{q(|x|)}$, составляющих вычисление

M на входе x. $C(t)$ есть содержимое лент, положение головок и состояние машины после t шагов вычисления; если к моменту t вычисление уже закончилось, то полагаем $C(t) := C(t-1)$. По входу x однозначно восстанавливается начальная конфигурация $C(0)$. Будем предполагать, что заключительная конфигурация с ответом «да» также стандартизована (рабочие ленты очищены, на выходной — ответ 1, головки — в стандартных позициях), так что ее вид можно восстановить по x не проводя всех вычислений. Пусть $next(C)$ обозначает следующую за C конфигурацию.

Игра для распознавания принадлежности $x \in L$. Игрок (M) старается убедить (A), что он обладает правильной последовательностью конфигураций вычисления $M(x)$ и что это вычисление дает ответ «да». Они совместно пересчитывают значения двух числовых переменных l и r. В начальный момент $l = 0$, $r = 2^{q(|x|)}$, $C_l := C(0)$, $C_r :=$ стандартная заключительная конфигурация с ответом «да».

- Ход (M) — пара (C_t, t), причем он утверждает, что (M.1): $t = (r+l)/2$ (легко проверить) и что (M.2): $C_t = C(t)$, чего (A) непосредственно проверить не может.

- Ход (A) состоит в выборе варианта пересчета: $l := t$ или $r := t$ (выбирается одна из половин отрезка).

- Игра останавливается после $q(|x|)$ пар шагов. При честной игре в конце получается $r = l + 1$.

- Предикат выигрыша: Когда одного из игроков удается уличить в обмане (если нарушено (M.1) или допущена ошибка в пересчете l, r), то нарушивший первым — проиграл. Если то, что (M) выдает в конце

игры за $C(l)$ и $C(l+1)$ (т. е. C_l и C_r) в самом деле не есть 2 последовательные конфигурации машины Тьюринга M, то выиграл (А). В остальных случаях выиграл (М).

Докажем, что эта игра в самом деле распознает язык L. В случае $x \in L$ выигрыш (М) обеспечен тривиальной стратегией — ему надо просто играть честно.

Разберем случай $x \notin L$. Тогда «правильной» допускающей последовательности конфигураций не существует и (М) вынужден обманывать.

До первого хода уже определены две пары (C_l, l) и (C_r, r): с $l = 0$, $r = 2^{q(|x|)}$; конфигурация C_l — известная всем начальная, а C_r — та допускающая заключительная конфигурация, возможность получения которой в результате вычисления $M(x)$ утверждает (М). При этом на самом деле выполнено условие

$$C_l = C(l) \ \wedge \ C_r \neq C(r).$$

Если (М) не нарушит правила, то (А) своим ходом может добиться сохранения этого условия:

1. Если (М) нарушит правила, т. е. $t \neq (l+r)/2$, то (А) уже выиграл.

2. Пусть (М) выберет C_t так, что за $t-l$ шагов машина M не приводит C_l в C_t. Тогда (А) следует выбрать левую половину, т. е. присваивание $r := t$.

3. Пусть (М) выберет C_t так, что за $t-l$ шагов машина M приводит C_l в C_t. Тогда (А) следует выбрать правую половину, т. е. присваивание $l := t$.

Если досрочного выигрыша не произошло, то условие справедливо и в конечный момент, когда $r = l+1$. По правилам игры (А) в этом случае также выигрывает. □

ГЛАВА 14

ПОЛНЫЕ ЗАДАЧИ ДЛЯ КЛАССА $PSPACE$ И КЛАССОВ ПОЛИНОМИАЛЬНОЙ ИЕРАРХИИ

Легко видеть, что класс $PSPACE$ и каждый класс полиномиальной иерархии «замкнут вниз» относительно сводимости $\leqslant_m^p$: если C — один из них, то для любых двух языков L_1, L_2 выполняется условие

$$L_1 \leqslant_m^p L_2,\ L_2 \in C \Rightarrow L_1 \in C.$$

По аналогии с понятием NP-полных задач определим Σ_n^p- и Π_n^p-полные задачи как наибольшие (по отношению $\leqslant_m^p$) элементы соответствующих классов. Ниже мы приводим типовые примеры таких задач.

§ 1. Квантифицированные булевы формулы

Определение 14.1. *Квантифицированной булевой формулой* называется формула вида

$$Q_1 x_1 \ldots Q_k x_k \varphi, \tag{14.1}$$

где φ — булева формула, $Q_i x_i$, $i = 1, \ldots, k$ — кванторы существования или всеобщности по булевым переменным

x_i. Квантифицированная формула называется Σ_n-формулой (Π_n-формулой), если она начинается с квантора $\exists$ (соответственно $\forall$) и имеет $n-1$ чередований кванторов. Квантифицированная булева формула (14.1) *замкнута*, если все переменные ее бескванторной части φ лежат среди $x_1, \ldots, x_n$.

Примеры.

- Формула $\forall x \exists y\,(x \vee y)$ является истинной замкнутой квантифицированной булевой формулой.
- Формула $\exists x (x \wedge \neg x)$ является ложной замкнутой квантифицированной булевой формулой.
- Формула $\exists x\,(x \wedge \neg y)$ является незамкнутой квантифицированной булевой формулой, истинностное значение которой зависит от истинностного значения переменной y.

Заметим, что

$$\exists x \varphi(x) \Leftrightarrow \varphi(0) \vee \varphi(1), \quad \forall x \varphi(x) \Leftrightarrow \varphi(0) \wedge \varphi(1),$$

поэтому введение булевых кванторов в пропозициональный язык не изменяет выразительных возможностей, но позволяет записывать некоторые утверждения существенно короче.

Лемма 14.2. *Каждая булева функция $f(x_1, \ldots, x_n)$ схемной сложности $c(f)$ может быть (за полиномиальное время) представлена квантифицированной булевой формулой длины $O(c(f))$ в каждом из видов Σ_1 и Π_1.*

Доказательство. Пусть булева схема для вычисления f размера $c(f) = k + 1$ есть

```
z1 <- t1;
   ...
zk <- tk;
y  <- t.
```

Тогда представляющие f формулы таковы:

$$\Sigma_1\text{-вид:}\quad \exists z_1 \ldots \exists z_k \left((z1 \leftrightarrow t1) \wedge \ldots \wedge (zk \leftrightarrow tk) \wedge t\right),$$
$$\Pi_1\text{-вид:}\quad \forall z_1 \ldots \forall z_k \left((z1 \leftrightarrow t1) \wedge \ldots \wedge (zk \leftrightarrow tk) \rightarrow t\right).$$

□

§ 2. Полные задачи для классов полиномиальной иерархии

Теорема 14.3. *Пусть $n \geqslant 1$. Задача распознавания истинности замкнутых квантифицированных булевых Σ_n-формул Σ_n^p-полна. Аналогичная задача для Π_n-формул Π_n^p-полна.*

Замечание. Мы предполагаем фиксированным побуквенное кодирование формул словами в алфавите $\{0, 1\}$. Задачи распознавания истинности формул указанных видов формализуются как задачи распознавания языков, состоящих из кодов формул. Двоичный код формулы F ниже обозначается через «F».

Доказательство. Сначала проверим, что эти задачи лежат в соответствующих классах. Пусть дана квантифицированная булева формула F вида (14.1). Заменим в ней каждый блок одноименных булевых кванторов на один квантор по двоичным словам длины m, равной максимуму количества кванторов в таких блоках. Например,

$$F = \underbrace{\forall x_1 \forall x_2}\ \underbrace{\exists y_1}\ \underbrace{\forall z_1 \forall z_1}\ \varphi(x_1, x_2, y_1, z_1, z_2)$$

преобразуется в

$$F' = \forall x_{|x|=2} \exists y_{|y|=2} \forall z_{|z|=2}\ \varphi(\pi_1(x), \pi_2(x), \pi_1(y), \pi_1(z), \pi_2(z)),$$

где π_i — функции «i-й бит слова». Теперь заметим, что истинностное значение выражения под кванторами

(а) не изменится при удлинении слов x, y, z дописыванием справа произвольных битов до длины $n = |F|$,

(б) может быть вычислено за полиномиальное время по формуле F и словам x, y, z, т. е.

$$(F \text{ — истинна}) \Leftrightarrow \forall^p x \exists^p y \forall^p z\ R(\text{«}F\text{»}, x, y, z),$$

где полином $p(n) = n$, а R — соответствующий предикат из класса P.

Разумеется, то же самое проходит с любой квантифицированной формулой. При этом если F есть Σ_n (Π_n)-формула, то соответствующая правая часть будет Σ_n^p (Π_n^p)-предикатом.

Теперь покажем, что произвольный язык $L \in \Sigma_n^p$ будет $\leqslant_m^p$-сводиться к задаче распознавания истинности квантифицированных булевых Σ_n-формул. (Случай с Π_n^p полностью аналогичен.) Пусть

$$x \in L \Leftrightarrow \exists^p w \forall^p b \ldots R(x, w, b, \ldots), \tag{14.2}$$

где p — полином, а $R \in P$. По доказанной ранее теореме 6.1 $P \subset P/Poly$, откуда схемная сложность булевой функции f, вычисляющей истинностное значение предиката $R(x, w, b, \ldots)$ по битам его аргументов, есть функция полиномиального роста. Более того, доказательство теоремы 6.1 предлагало прямую (полиномиальную по времени) конструкцию построения соответствующей схемы S_f по

числу $m = |x| + |w| + |w| + \ldots$, которое, в свою очередь, легко восстановить по $|x|$.

Применим лемму 14.2 к схеме S_f и получим представление булевой функции f квантифицированной булевой Σ_1 или Π_1-формулой F_f. Причем выберем то из них, в котором кванторы совпадают с самым правым квантором в (14.2). Тогда

$$x \in L \Leftrightarrow \underbrace{\exists w^1 \ldots \exists w^{p(|x|)}\, \forall b^1 \ldots \forall b^{p(|x|)} \ \ldots \ F_f}_{F},$$

где формула F справа есть квантифицированная булева Σ_n-формула (новых чередований кванторов не добавилось), а отображение

$$x \mapsto |x| \mapsto m \mapsto S_f \mapsto F_f \mapsto F$$

(т. е. сводящая функция) вычислимо за полиномиальное время. □

§ 3. Пример $PSPACE$-полной задачи

Класс $PSPACE$ также содержит наибольшие элементы — $PSPACE$-полные задачи. К ним относится задача распознавания истинности замкнутых квантифицированных булевых формул. Она представлена языком TQBF, состоящим из (кодов) всех истинных замкнутых квантифицированных булевых формул

$$Q_1 x_1 \ldots Q_n x_n \varphi(x_1, \ldots x_n),$$

($Q_i x_i$ — квантор $\forall$ или $\exists$ по булевой переменной x_i).

Лемма 14.4. TQBF$\in PSPACE$.

Доказательство. Определение истинности квантифицированной формулы F в предваренном виде легко представить игрой. Добавлением фиктивного квантора $\exists$ слева можно свести общий случай к

$$F = \exists x_1^1 \ldots \exists x_1^k \forall y_1^1 \ldots \forall y_1^l \exists \ldots \varphi.$$

(M) играет за квантор существования, (A) — за квантор всеобщности. Они оба делают ходы длины $|F|$:

$$\begin{aligned} w_1 &= w_1^1 w_1^2 \ldots \\ b_1 &= b_1^1 b_1^2 \ldots \\ w_2 &= w_2^1 w_2^2 \ldots \end{aligned}$$

Количество ходов можно также взять $|F|$; оно мажорирует число чередований кванторов в формуле. Предикат выигрыша есть

$$R(\text{`}F\text{'}, w_1, b_1, w_2, \ldots) \Leftrightarrow \varphi[w_1^1/x_1^1, \ldots, b_1^1/y_1^1, \ldots]$$

(вместо переменных подставляются первые биты соответствующих ходов игроков). Истинность F эквивалентна условию существования выигрышной стратегии для (M). □

Теорема 14.5. *Язык* TQBF *является* $PSPACE$*-полным.*

Доказательство. Докажем, что произвольный язык $L \in PSPACE$ $\leqslant_m^p$-сводится к TQBF. Рассмотрим игру с полиномиальным числом ходов, распознающую L:

$$x \in L \Leftrightarrow \underbrace{\exists^p w_1 \forall^p b_1 \ldots}_{q(|x|)} R(x, w_1 b_1 \ldots),$$

где p, q — полиномы, а $R \in P$. Заменим кванторы по словам на блоки кванторов по булевым переменным, а преди-

кат R — на вычисляющую его булеву схему полиномиального размера:

$$x \in L \Leftrightarrow$$
$$\Leftrightarrow \underbrace{\exists w_1^1 \ldots \exists w_1^{p(|x|)} \forall b_1^1 \ldots \forall b_1^{p(|x|)} \ldots}_{p(|x|)\cdot q(|x|)} (S(x, w_1, b_1 \ldots) = 1).$$

Условие $S(x, w_1, b_1 \ldots) = 1$ заменим эквивалентной квантифицированной булевой Σ_1-формулой G полиномиальной длины (по лемме 14.2). Получим квантифицированную булеву формулу

$$F(x) = \exists w_1^1 \ldots \exists w_1^{p(|x|)} \forall b_1^1 \ldots \forall b_1^{p(|x|)} \ldots G,$$

для которой

$$v \in L \Leftrightarrow F(v) \in \mathrm{TQBF}.$$

Все преобразования $v \mapsto F(v)$ сами могут быть проведены за полиномиальное время, что и доказывает сводимость $L \leqslant_m^p PSPACE$. □

СПИСОК ЛИТЕРАТУРЫ

[1] *А. Ахо, Дж. Хопкрофт, Дж. Ульман* Построение и анализ вычислительных алгоритмов. — М.: Мир, 1979.

[2] *М. Гэри, Д. Джонсон* Вычислительные машины и труднорешаемые задачи. — М.: Мир, 1982.

[3] *А. Китаев, А. Шень, М. Вялый* Классические и квантовые вычисления. — М.: МЦНМО, ЧеРо, 1999.

[4] *Дж. Сэвидж* Сложность вычислений. — М.: Факториал, 1998.

[5] *P. Gacs, L. Lovasz* Complexity of Algorithms. 1999. `http://www.cs.yale.edu/HTML/YALE/CS/HyPlans/lovasz/complex.ps`

[6] *J. van Leeuwen ed.* Handbook of Theoretical Computer Science. Volume A. Algorithms and Complexity. — Amsterdam et al.: Elsevier / Cambridge, MA: MIT Press, 1990.

[7] *M. Sipser* Introduction to the Theory of Computation. — Boston: PWS Publishing Company, 1997.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ