6 ВЫЧИСЛИТЕЛЬНАЯ СЛОЖНОСТЬ АЛГОРИТМОВ

6.1 Понятие о сложности алгоритмов. Временная и емкостная сложности. Детерминизм и недетерминизм

Возникновение и развитие теории вычислительной сложности алгоритмов (иначе, количественной теории вычислений) относится к 50-м годам XX-го века, хотя отдельные работы в этом направлении появлялись и раньше. Это развитие стимулировалось появлением реальных вычислительных установок. Покуда речь шла о потенциальной алгоритмической осуществимости тех или иных задач, никого не интересовало качество получаемых алгоритмов. Но с появлением компьютеров стало ясно, что два алгоритма, тождественные с точки зрения входа-выхода, то есть дающие при одних и тех же данных одинаковые результаты, могут требовать существенно различного времени для своей работы на компьютере при одних и тех же данных, или же отличаться другими характеристиками работы.

Итак, теория сложности вычислений началась с определения критериев сложности, которые позволяют в математически точном смысле говорить, что один алгоритм проще другого с точки зрения исполнения.

Естественно, что такие определения имеют смысл, когда фиксирована "вычислительная установка"(или, как еще говорят, модель вычислений), то есть когда можно говорить об алгоритме в точном смысле слова.

В дальнейшем в качестве модели вычислений мы будем рассматривать некоторый вариант машины Тьюринга, а именно машину Тьюринга с одной входной, одной выходной и несколькими рабочими (бесконечными как вправо, так и влево) лентами, причем каждая из лент разбита на ячейки (клетки). Для простоты дадим определение *машины Тьюринга с одной рабочей лентой* (1-МТ).

1-МТ — это четверка

$$M = \langle \Sigma, Q, \pi, q_0 \rangle,$$

где Σ — входной алфавит, Q — алфавит состояний, $q_0 \in Q$ — начальное состояние, а π — программа, то есть отображение

$$\pi : (\Sigma \times \Sigma \times Q) \to (\Sigma \times Q \times \{L, R, N, S\} \times \{L, R, N\} \times \Sigma).$$

Алфавит Σ содержит букву $\wedge$, называемую "пусто". В дальнейшем будем считать, что Σ содержит, кроме $\wedge$, еще по крайней мере две буквы, для определенности 0 и 1.

Другими словами, программа 1-МТ состоит из команд. Каждая команда делится на левую и правую части. Левая часть состоит из буквы входной ленты, буквы рабочей ленты и состояния. Правая часть состоит из новой буквы рабочей ленты, нового состояния, указаний, куда двигаться на рабочей и входной лентах, и выходного символа. Разные команды имеют различные левые части.

В начальный момент времени все ячейки на рабочей и выходной лентах заполнены буквами $\wedge$, а на входной ленте записано некоторое слово $w \in (\Sigma \setminus \{\wedge\})^*$, называемое входным; справа и слева от входного слова все клетки также заполнены буквами $\wedge$; головка на входной ленте стоит на первой (самой левой) букве входного слова; 1-МТ M находится в начальном состоянии q_0.

Работа 1-МТ M состоит в последовательном выполнении такта за тактом. При выполнении такта M читает букву под головкой на входной ленте, читает букву под головкой на рабочей ленте, и, в зависимости от своего состояния, меняет свое состояние, двигает головки на входной и рабочей лентах, при этом может менять букву, записанную под головкой на рабочей ленте, возможно, пишет какую-либо букву на выходную ленту, обязательно сдвигая при этом головку на ней вправо, а также, возможно, останавливается.

Формально, пусть на входной ленте головка стоит над ячейкой с буквой $a \in \Sigma$, а на рабочей — над ячейкой с буквой $b \in \Sigma$. Пусть M находится в состоянии $q \in Q$.

Выполняя такт работы, M вычисляет

$$\pi(a, b, q) = \langle a', q', \mu, \nu, c\rangle.$$

Затем M производит следующие действия:

1. Пишет на рабочей ленте под головкой букву a'.

2. Переходит в состояние q'.

3. Если $\mu = R$, то M сдвигает головку на рабочей ленте на одну клетку вправо, если $\mu = L$, то влево, если $\mu = N$, то оставляет головку на месте.

4. Если $\nu = R$, то M сдвигает головку на входной ленте на одну клетку вправо. Однако если головка на входной ленте обозревала букву $\wedge$ и пытается перейти на клетку, на которой также записана буква $\wedge$, то головка на входной ленте остается на месте.

5. Если $\nu = L$, M сдвигает головку на входной ленте влево. Однако если головка на входной ленте обозревала букву $\wedge$ и пытается перейти на клетку, на которой также записана буква $\wedge$, то головка на входной ленте остается на месте.

6. Если $\nu = N$, то M оставляет головку на входной ленте на месте.

7. Если $c \neq \wedge$, то M пишет на выходную ленту букву c и сдвигает на этой ленте головку на одну клетку вправо.

8. Наконец, если $\mu = S$, то M останавливается.

Если M остановилась, то слово $v \in (\Sigma \setminus \{\wedge\})^*$, записанное к этому времени на выходной ленте, называют результатом работы M.

Итак, легко видеть, что M определяет частичное отображение

$$M : (\Sigma \setminus \{\wedge\})^* \to (\Sigma \setminus \{\wedge\})^*,$$

преобразуя слово w на входной ленте в слово $v = M(w)$ на выходной.

Машины M и M' называются *эквивалентными*, если определяемые ими отображения

$$M, M' : (\Sigma \setminus \{\wedge\})^* \to (\Sigma \setminus \{\wedge\})^*$$

совпадают.

Рассматривают также специальный класс 1-МТ, называемых 1-МТ распознавателями. 1-МТ M называется 1-МТ *распознавателем*, если для любого $w \in (\Sigma \setminus \{\wedge\})^*$ выполнены условия:

(1) $M(w)$ определено и

(2) $M(w) \in \{0, 1\}$.

Всякий 1-МТ распознаватель M определяет язык $L_M \subseteq (\Sigma \setminus \{\wedge\})^*$:

$$L_M = \{w \in (\Sigma \setminus \{\wedge\})^* \mid M(w) = 1\}.$$

Определенные выше машины называются детерминированными.

Наряду с детерминированными, рассматривают также недетерминированные многоленточные машины Тьюринга. *Недетерминированная одноленточная машина машина Тьюринга* (1-НМТ) определяется почти так же, как 1-МТ, но программа π у недетерминированной машины — это частичное отображение во множество всех подмножеств декартова произведения

$$\Sigma \times Q \times \{L, R, N, S\} \times \{L, R, N\} \times \Sigma,$$

а не в само это произведение, то есть

$$\pi : (\Sigma \times \Sigma \times Q) \to \mathfrak{P}(\Sigma \times Q \times \{L, R, N, S\} \times \{L, R, N\} \times \Sigma),$$

где $\mathfrak{P}(K)$ есть множество всех подмножеств множества K.

Другими словами, 1-НМТ может иметь несколько разных команд с одинаковой левой частью. Работа 1-НМТ M на слове $w \in (\Sigma \setminus \{\wedge\})^*$ может идти по различным путям (недетерминированно) и давать различные (вообще говоря) результаты, то есть здесь

$$M : (\Sigma \setminus \{\wedge\})^* \to \mathfrak{P}((\Sigma \setminus \{\wedge\})^*).$$

Определение 6.1.1 (конфигурации 1-НМТ). *Конфигурацией 1-МТ (1-НМТ)*

$$M = \langle \Sigma, Q, \pi, q_0 \rangle$$

будем называть пятерку

$$\langle w, v, i, j, q \rangle,$$

где:

$w \in (\Sigma \setminus \{\wedge\})^*$ *— входное слово;*

$v \in \Sigma^*$ *— слово, записанное на рабочей ленте, без букв* $\wedge$ *спереди и сзади;*

$i \in \{0, 1, \ldots, |w| + 1\}$ *— номер позиции головки на входной ленте (головка может находиться либо в пределах слова* w*, либо прямо перед* w *(*$i = 0$*), либо сразу после* w *(*$i = |w| + 1$*));*

$j \in \{0, \pm 1, \pm 2, \ldots\}$ *— номер позиции головки на рабочей ленте, считая от начала слова* v*; при этом если головка находится прямо перед первой буквой* v*, то* $j = 0$ *, если правее, то* $j > 0$ *, если левее, то* $j < 0$*;*

$q \in Q$ *— состояние* M*.*

Начальной конфигурацией 1-МТ (1-НМТ) $M = \langle \Sigma, Q, \pi, q_0 \rangle$ на слове $w \in (\Sigma \setminus \{\wedge\})^*$ называется конфигурация $\langle w, \lambda, 1, 0, q_0 \rangle$, где, напомним, λ обозначает пустое слово. Ясно, что конфигурация полностью определяет дальнейшее поведение машины Тьюринга. Выполняя такт работы, M переводит всякую конфигурацию в другую конфигурацию способом, описанным выше. Тем самым 1-МТ M определяет отображение на множестве всех своих конфигураций, а 1-НМТ — отображение из множества всех своих конфигураций во множество всех конечных подмножеств своих конфигураций.

Конфигурация называется результативной, если, начиная с нее, 1-МТ останавливается (а 1-НМТ может остановиться по некоторому пути вычислений).

1-НМТ M называется 1-НМТ *распознавателем*, если для любого $w \in (\Sigma \setminus \{\wedge\})^*$ выполнены условия:

(1) существует останавливающееся вычисление M на w, и

(2) $M(w) \subseteq \{0, 1\}$.

Всякий 1-НМТ распознаватель M определяет язык $L_M \subseteq (\Sigma \setminus \{\wedge\})^*$:

$$L_M = \{w \in (\Sigma \setminus \{\wedge\})^* \mid 1 \in M(w)\}.$$

Мы будем рассматривать две количественные характеристики МТ и НМТ: время работы и память.

Пусть M — 1-МТ и $w \in (\Sigma \setminus \{\wedge\})^*$. Временем работы M на слове w будем называть число тактов работы M на w, или бесконечность (∞), если M не останавливается на w. Время работы M на w будем обозначать через $t^*_M(w)$. Итак, $t^*_M(w) \in \{0, 1, 2, \dots\} \cup \{\infty\}$.

Если M — 1-НМТ, то временем работы M на слове w будем называть число тактов работы M на w в наихудшем случае, то есть для определения $t^*_M(w)$ мы должны перебрать все останавливающиеся пути вычислений M на w, при которых M не проходит дважды одной и той же конфигурации, и взять максимум по числу тактов. Если ни при каком способе вычислений M не останавливается на w, определим $t^*_M(w)$ как бесконечность. Итак, $t^*_M(w) \in \{0, 1, 2, \dots\} \cup \{\infty\}$ и для 1-НМТ.

Аналогично $t^*_M(w)$, определяется функция $s^*_M(w)$ — память при работе M на слове w. Однако при этом подсчитывается не число тактов работы M на w, а число клеток на рабочей ленте, над которыми побывала головка машины M при работе над словом w.

Итак, $s^*_M(w)$ — это число клеток, просмотренных 1-МТ M при работе на слове w (соответственно просмотренных 1-НМТ M при работе на w в худшем случае — максимум здесь берется уже не по числу тактов, а по числу посещенных клеток на рабочей ленте). Ясно, что $s^*_M(w) \in \{0, 1, 2, \dots\} \cup \{\infty\}$.

Определение 6.1.2 (временной сложности). *Временем работы 1-МТ (1-НМТ) M или временной сложностью M называется функция $t_M : \{0, 1, \dots\} \to \{\infty, 0, 1, \dots\}$, определяемая так:*

$$t_M(n) = max\{t^*_M(w) \mid w \in (\Sigma \setminus \{\wedge\})^*, |w| \leq n\}.$$

Определение 6.1.3 (ёмкостной сложности). *Памятью, используемой 1-МТ (1-НМТ) M, или емкостной (ленточной) сложностью M называется функция*

$$s_M : \{0, 1, \dots\} \to \{\infty, 0, 1, \dots\},$$

определяемая так:

$$s_M(n) = max\{s^*_M(w) \mid w \in (\Sigma \setminus \{\wedge\})^*, |w| \leq n\}.$$

Рассмотрение распознавателей позволяет перейти от определения (временной или емкостной) сложности машин Тьюринга к сложности языков.

Пусть $L \subseteq (\Sigma \setminus \{\wedge\})^*$ — некоторый язык. Будем говорить, что L *распознается с детерминированной (недетерминированной) временной сложностью* t, где $t : \{0, 1, ..\} \to \{0, 1, ..\}$, если существует 1-МТ (соответственно, 1-НМТ) распознаватель M такой, что $L = L_M$ и $t = t_M$.

Будем говорить, что L *распознается с детерминированной (недетерминированной) емкостной сложностью* s, если существует такой 1-МТ (соответственно, 1-НМТ) распознаватель M, что $L = L_M$ и $s = s_M$.

Легко видеть, что один и тот же язык может распознаваться с различной сложностью. Таким образом, говорить о сложности языка вполне строго нельзя. Этим обусловлено введение верхних и нижних оценок сложности языков.

Пусть $f, g : \{0, 1, ..\} \to \{0, 1, ..\}$. Будем говорить, что функция f *асимптотически мажорируется* функцией g (или что g является *асимптотической мажорантой* f), тогда и только тогда, когда

$$(\exists n)(\forall m \geq n) f(m) \leq g(m).$$

Пусть $L \subseteq (\Sigma \setminus \{\wedge\})^*$ — некоторый язык, $u : \{0, 1, \dots\} \to \{0, 1, \dots\}$. Будем говорить, что u есть *верхняя оценка детерминированной (недетерминированной) временной (емкостной) сложности* языка L, если существует 1-МТ (соответственно, 1-НМТ) распознаватель M такой, что $L = L_M$ и функция t_M (соответственно, s_M) асимптотически мажорируется функцией u.

Будем говорить, что

$$l : \{0, 1, \dots\} \to \{0, 1, \dots\}$$

есть *нижняя оценка детерминированной (недетерминированной) временной (емкостной) сложности* языка L, если для всякого 1-МТ (соответственно, 1-НМТ) распознавателя M такого, что $L = L_M$, функция t_M (соответственно, s_M) является асимптотической мажорантой функции l.

Класс всевозможных языков в алфавите $(\Sigma \setminus \{\wedge\})$ с верхней оценкой детерминированной временной сложности $f : \{0, 1, \dots\} \to \{0, 1, \dots\}$ обозначается $DTIME(f)$.

Класс всевозможных языков в алфавите $(\Sigma \setminus \{\wedge\})$ с верхней оценкой недетерминированной временной сложности $f : \{0, 1, \dots\} \to \{0, 1, \dots\}$ обозначается $NTIME(f)$.

Класс всевозможных языков в алфавите $(\Sigma \setminus \{\wedge\})$ с верхней оценкой детерминированной емкостной сложности $f : \{0, 1, \dots\} \to \{0, 1, \dots\}$ обозначается $DSPACE(f)$.

Класс всевозможных языков в алфавите $(\Sigma \setminus \{\wedge\})$ с верхней оценкой недетерминированной емкостной сложности $f : \{0, 1, \dots\} \to \{0, 1, \dots\}$ обозначается $NSPACE(f)$.

В следующих параграфах мы подробно рассмотрим соотношения между классами сложности языков и машин Тьюринга. Однако некоторые очевидные соотношения можно отметить сразу. Например, очевидно, что для любой 1-МТ (1-НМТ) M функция s_M асимптотически мажорируется функцией t_M.

Определение 6.1.4. *Пусть $f, g : \{0, 1, ..\} \to \{0, 1, ..\}$. Будем говорить, что функция f сильно мажорируется функцией g (или что g является сильной мажорантой f) тогда и только тогда, когда f асимптотически мажорируется функцией g и функция $(g - f)$ является неограниченной почти неубывающей функцией. Другими словами, когда*

$$(\exists n)(\forall m \geq n)(f(m) \leq g(m) \& g(m) - f(m) \geq g(m-1) - f(m-1)) \&$$

$$(\forall n)(\exists m)(g(n) - f(n) < g(m) - f(m)).$$

В дальнейшем мы будем часто использовать функцию

$$log : \{0, 1, \dots\} \to \{0, 1, \dots\},$$

определенную так: $log(n) =]log_2(n)[$, где $]x[$ является наименьшим целым, большим или равным x, log_2 — обычный логарифм по основанию 2.

1-МТ M называется тотальной, если отображение M всюду определено.

1-НМТ M называется тотальной, если M всегда останавливается, причем по конечному числу путей.

Рассмотрим теперь тотальную 1-МТ (1-НМТ). Из тотальности следует, что s_M является всюду определенной функцией на множестве натуральных чисел. Зададимся вопросом, сколько различных результативных конфигураций может возникнуть при работе M на слове w

длины $\leq n$? Простой подсчет показывает, что не более

$$(n+2)|Q|s_M(n)|\Sigma|^{s_M(n)}$$

конфигураций. Легко видеть, что тогда и $t^*_M(w)$ не превосходит указанного выше числа, значит $t_M(m)$ асимптотически мажорируется функцией

$$(n+2)|Q|s_M(n)|\Sigma|^{s_M(n)}.$$

Другими словами, если $f : \{0, 1, \dots\} \to \{0, 1, \dots\}$, $f(m)$ — сильная мажоранта функции $s_M(m)log(ms_M(m))$, то $t_M(m)$ асимптотически мажорируется функцией $2^{f(m)}$.

Этими рассуждениями доказана

Теорема 6.1. *Пусть M — 1-МТ (1-НМТ), s_M и t_M задают емкостную и временную сложности M и f является сильной мажорантой функции $s_M(m)log(ms_M(m))$. Тогда*

$$s_M \ll t_M \ll 2^f,$$

где $\ll$ — отношение асимптотической мажорируемости на арифметических функциях.

Напомним, что (всюду определенной) арифметической функцией называется отображение из ω^n в ω.

До сих пор мы говорили только об одноленточных машинах Тьюринга. Определение k-ленточной машины Тьюринга (k-МТ и k-НМТ) аналогично определению одноленточной, но она снабжена k рабочими лентами, пронумерованными натуральными числами от 1 до k и одна из этих лент в каждый момент времени является текущей. В начальный момент текущей является лента с номером 1. В определении программы машины Тьюринга добавляется, что машина обозревает букву на текущей рабочей ленте, и эта буква и фигурирует при вычислении π в качестве второго аргумента. π определяет также, какая из k рабочих лент станет текущей в следующий момент времени. При этом положение головки на каждой рабочей ленте сохраняется, даже если лента в данный момент и неактивна. Рутинные детали определения оставляются читателю.

При определении емкостной сложности для k-МТ (k-НМТ) принимаются в счет просмотренные клетки на всех рабочих лентах.

Несложно промоделировать работу k-МТ (k-НМТ) на 1-МТ (1-НМТ), записывая содержимое всех лент на одну с разделителями, за которыми следует записанный "номер ленты". При необходимости лента "раздвигается". Может показаться, что для этого необходима еще одна буква для использования ее в качестве разделителя. Однако это не так. Один из возможных приемов — это вначале промоделировать работу машины с k лентами на дублирующей машине с k лентами, которая каждую букву на рабочих лентах и пишет, и читает дважды (рядом). Тогда на рабочих лентах дублирующей машины никогда не встретится комбинация букв $\wedge 01$. Эта комбинация и может служить разделителем при моделировании дублирующей машины с k лентами на машине с одной лентой.

Простой подсчет позволяет оценить, как зависит сложность моделирующей машины с одной лентой от сложности моделируемой машины с k лентами. Резюмирует результаты наших рассуждений.

Теорема 6.2. *Пусть M — k-МТ (k-НМТ) для некоторого k. Существует эквивалентная ей 1-МТ (1-НМТ) M_1 и константы*

$$c, d \in \{0, 1, \dots\}$$

такие, что $s_{M_1} \ll cs_M$, $t_{M_1} \ll dt_M^2$, где $\ll$ — отношение асимптотической мажорируемости на арифметических функциях.

Упражнения

Упражнение 6.1.1. *Проведите формальное доказательство теоремы 6.2.*

6.2 Моделирование недетерминированных машин детерминированными. Теорема об иерархии. Основные классы сложности

Из главы 4 о машинах Тьюринга и вычислимых функциях читателю уже известно, что, с точки зрения вычисляемых функций, недетерминированные машины Тьюринга могут моделироваться детерминированными. Эта формулировка нуждается в некотором уточнении, но для распознавателей утверждение может быть сформулирована просто: язык тогда и только тогда распознается некоторой 1-НМТ, когда он распознается некоторой 1-МТ. Однако ситуация усложняется, когда мы хотим распознавать языки с некоторой сложностью.

Теорема 6.3. *Пусть M — 1-НМТ распознаватель, распознающий L_M. Пусть f — рекурсивная функция, являющаяся асимптотической мажорантой функции t_M (то есть временной сложности M), и t_f — временная сложность некоторой 1-МТ, вычисляющей f. Существует 1-МТ распознаватель M', распознающий язык L_M. Кроме того, для некоторой константы c_M (вычислимой по M), временная сложность $t_{M'}$ асимптотически мажорируется функцией*

$$2^{c_M(f(n)+n+log(t_f(n)))}.$$

Доказательство. Существует много способов построения M' по M. Видимо, наиболее тривиальный таков.

Получив слово w на входе, вычислим $f(|w|)$ и будем перебирать все пути исполнения M на слове w, длина которых не превосходит $f(|w|)$. Если хотя бы по одному такому пути M остановится и даст на выходе 1, то остановимся и дадим на выходе 1. Иначе, то есть если на всех таких путях M либо не останавливается, либо останавливается с выходом 0, остановимся и дадим на выходе 0.

Ясно, что такой алгоритм является детерминированным и может быть осуществлен на подходящей (и рутинно конструируемой) k-МТ M''. Ясно, что если $n = |w|$, то $t_{M''}(n)$ при подходящих константах c, c', c'' (не зависящих от n) не превосходит числа

$$c'n + c'' + c^{f(n)}f(n) + t_f(n).$$

Тогда можно подобрать константу c'_M так, что

$$t_{M''}(n) \leq 2^{c'_M(f(n)+n+log(t_f(n)))}.$$

Значит, в силу теоремы 6.2, можно подобрать такие 1-МТ M' и константу c_M, что $L_{M''} = L_{M'}$ и

$$t_{M'}(n) \leq 2^{c_M(f(n)+n+log(t_f(n)))}.$$

■

Аналогичная теорема может быть сформулирована и доказана для емкостной сложности с использованием теоремы 6.1. Однако в упражнениях к настоящему параграфу будет приведена лучшая оценка для емкостной сложности при моделировании недетерминированной машины детерминированной (так называемая теорема Сэвиджа).

Мы хотим теперь показать, что существуют языки сколь угодно большой сложности, то есть что иерархия языков по сложности нетривиальна. В силу доказанных нами результатов, достаточно доказать нетривиальность иерархии языков для какой-нибудь одной сложности. Мы докажем это для детерминированной емкостной сложности. Нетривиальность остальных иерархий (по недетерминированной емкостной и временной, и по детерминированной временной) будет следовать из теорем 6.1 и 6.3

В дальнейшем будем рассматривать языки в некотором фиксированном конечном алфавите, содержащем по крайней мере буквы 0 и 1.

Теорема 6.4. *Пусть f — произвольная рекурсивная функция, вычислимая с емкостной сложностью s_f. Существует язык L_f с нижней оценкой детерминированной емкостной сложности $f(n) - 1$, и константа r такие, что $r(f(n) + s_f(n) + n)$ является верхней оценкой детерминированной емкостной сложности L_f.*

Доказательство. Напомним, что мы требуем, чтобы входной алфавит Σ рассматриваемых машин Тьюринга содержал, кроме буквы $\wedge$ (пусто), еще по крайней мере две различные буквы (для определенности 0 и 1).

Ниже мы приводим способ записи произвольной 1-МТ с входным алфавитом Σ в виде слова в алфавите $\Sigma \setminus \{\wedge\}$. Этот способ намеренно сделан неоднозначным, то есть одной 1-МТ M соответствует бесконечно много попарно различных записей. В то же время каждому слову w в алфавите Σ соответствует не более, чем одна 1-МТ. Если слову w соответствует 1-МТ, мы обозначаем эту 1-МТ через M_w .

Идея доказательства состоит в диагонализации всех 1-МТ распознавателей в пределах емкостной сложности f, то есть в построении 1-МТ распознавателя U_f, который, получив на входе запись 1-МТ распознавателя M, выдает на выходе либо инверсию результата работы M на записи M (то есть результата работы машины M, на вход которой подана она сама), если M на записи M выдает результат с использованием не более $f(M)$ клеток рабочей ленты, либо 1 в противном случае. Инверсией 0 мы называем 1, а инверсией 1 называем 0. Инверсия буквы, отличной от 0 и 1, совпадает с этой буквой. Инверсия слова получается из этого слова заменой каждой его буквы на инверсию этой буквы.

Уточним понятие записи машины Тьюринга на ленте.

Итак, пусть дана 1-МТ M, то есть четверка

$$M = \langle \Sigma, Q, \pi, q_0 \rangle,$$

где Σ — алфавит, Q — алфавит состояний, $q_0 \in Q$ — начальное состояние, а π — программа, то есть отображение

$$\pi : (\Sigma \times \Sigma \times Q) \to (\Sigma \times Q \times \{L, R, N, S\} \times \{L, R, N\} \times \Sigma).$$

Без ограничения общности можно считать, что $Q = \{1, 2, 3, \ldots, m\}$ для некоторого натурального m, то есть что Q представляет собой начальный отрезок множества положительных целых чисел, и что, кроме того, $q_0 = 1$.

Если i — положительное целое число, то через $[i]$ будем обозначать слово в алфавите $\{1\}$ длины i.

Рассмотрим также кодировку букв L, R, N, S из алфавита движений, именно, $L' = 00, R' = 01, N' = 10, S' = 11$.

В качестве кода M определим слово $1\pi'$, где π' — код программы π, представляющий собой конкатенацию всевозможных слов

$$\alpha\beta[i]0\gamma[j]0u'v'\delta$$

для каждой тройки $\langle\alpha, \beta, i\rangle \in (\Sigma \times \Sigma \times Q)$, где $\pi(\alpha, \beta, i) = \langle\gamma, j, u, v, \delta\rangle$.

Мы хотим обеспечить, чтобы для каждой машины M имелось бесконечно много кодов M. Для этого, если слово w является кодом M, будем считать слово $0w$ также кодом M.

Итак, мы рассмотрели кодировку всевозможных 1-МТ в алфавите Σ словами в алфавите $\Sigma \setminus \{\wedge\}$. Ясно, что не каждое слово в этом алфавите будет кодом какой-нибудь 1-МТ M. Если w является кодом 1-МТ, то эту 1-МТ мы будем обозначать M_w.

Рассмотрим 8-МТ распознаватель V_f в алфавите Σ, действующий следующим образом. Пусть w — входное слово.

Сначала V_f пропускает на входной ленте стоящие в начале слова буквы 0 вплоть до, включительно, первой буквы 1, а оставшуюся часть слова переписывает на первую рабочую ленту.

Далее V_f вычисляет число $f(|w|)$ и записывает это число в унарной системе счисления на 7-ю рабочую ленту. Далее V_f вычисляет число

$$(|w| + 2)mf(|w|)|\Sigma|^{f(|w|)}$$

и записывает это число в двоичной системе счисления на 2-ю рабочую ленту. Рутинные детали этих вычислений здесь опустим.

Далее V_f устанавливает головку на входной ленте на первую букву слова w. Затем V_f пытается осуществить рутинную процедуру моделирования работы 1-МТ M_w, используя свою входную ленту как входную для M_w, 6-ю рабочую ленту как выходную для M_w, 3-ю рабочую ленту для запоминания текущего состояния M_w (которое в начале есть 1), и свою четвертую рабочую ленту как рабочую для M_w. При этом вторая рабочая лента используется для определения значения отображения π_{M_w} — программы M_w. Описание этой рутинной процедуры мы опустим.

В начальный момент времени перед моделированием на пятой ленте записывается 0. После каждого шага моделирования M_w машина V_f увеличивает число в двоичной системе счисления, записанное на 5-ой рабочей ленте, на 1. Кроме того, после каждого шага моделирования M_w машина V_f проверяет, что на третьей рабочей ленте использовано не более $f(|w|)$ клеток. Для этого головка на 8-й ленте движется синхронно с головкой на 3-ей, и на каждой обозреваемой клетке пишет 1. После каждого шага проверяется, что число клеток, заполненных единицами на 8-й ленте, не превосходит числа клеток, заполненных единицами, на седьмой ленте (число последних именно равно $f(|w|)$).

Моделирование работы M_w проводится не более

$$(|w|+2)mf(|w|)|\Sigma|^{f(|w|)}$$

шагов. Для этого после каждого шага моделирования V_f сравнивает вторую рабочую ленту, на которой записано число

$$(|w|+2)mf(|w|)|\Sigma|^{f(|w|)},$$

и 5-ю, на которой записано число осуществленных шагов моделирования M_w.

При работе V_f могут возникнуть следующие ситуации, которые она распознает:

1). w оказывается не кодом 1-МТ. В этом случае V_f останавливается и выдает 1 на выходную ленту.

2). Сделано уже

$$(|w|+2)mf(|w|)|\Sigma|^{f(|w|)}$$

шагов моделирования M, но M еще не завершила работу. В этом случае V_f останавливается и выдает 1 на выходную ленту.

3). M_w использовала более $f(|w|)$ клеток на рабочей ленте (то есть на третьей рабочей ленте V_f). Механизм этой проверки описан выше. В этом случае V_f останавливается и выдает 1 на выходную ленту.

4). На 6-й рабочей ленте V_f оказывается записанным более чем однобуквенное слово. Содержательно это означает, что M_w не оказалась распознавателем. В этом случае V_f останавливается и выдает 1 на выходную ленту.

5). M_w останавливается. В этом случае если на 6-й рабочей ленте записана 1, V_f останавливается и выдает 0 на выходную ленту. В противном случае V_f останавливается и выдает 1 на выходную ленту.

Ясно, что

$$\begin{aligned} s^*_{V_f}(w) \quad &\leq \quad 4f(|w|) + log((|w|+2)mf(|w|)|\Sigma|^{f(|w|)})s_f(|w|) + \\ |w| + m + 2 \quad &\leq \quad k(f(|w|) + |w|) + s(|w|) \end{aligned}$$

при подходящей константе k. То есть при любом n

$$s_{V_f}(n) \ \leq \ k(f(n)+n) + s_f(n).$$

Рассмотрим 1-МТ U_f в алфавите Σ, моделирующую V_f. Согласно теореме 6.2 и предыдущему соотношению, U_f может быть выбрана так, что для некоторой константы r

$$s_{U_f}(n) \ \ll \ r(f(n) + n + s_f(n)),$$

где $\ll$ — отношение асимптотической мажорируемости.

Рассмотрим язык $L = L_{U_f}$. Выше замечено, что верхняя граница сложности L есть $r(f(n)+n+s_f(n))$. Покажем, что L не распознается никаким 1-МТ распознавателем с емкостной сложностью, асимптотически мажорируемой f.

Пусть это не так. Тогда существует 1-МТ распознаватель M такой, что $L_M = L$, и $s_M \ll f$. Рассмотрим код M, то есть такое слово $w \in (\Sigma \setminus \{\wedge\})^*$, что $M = M_w$. Рассмотрим работу M на всевозможных словах вида $00\ldots0w$. Из асимптотической мажорируемости функции s_M функцией f следует, что для некоторого конкретного такого слова $u = 00\ldots0w$ (и даже для бесконечного множества таких слов) $s^*_M(u) \leq f(|u|)$. Это слово u зафиксируем и рассмотрим работу M_u на слове u. Отметим при этом, что $M_u = M_w = M$. Следовательно, $s^*_{M_u}(u) \leq f(|u|)$.

Возможны два случая при работе V_f на u.

1). $u \in L_{M_u} = L$. Тогда $M_u(w) = 1$. Так как $s^*_{M_u}(u) \leq f(|u|)$, то, как отмечено в доказательстве теоремы 6.1,

$$t^*_{M_u}(u) \leq (|u|+2)mf(|u|)|\Sigma|^{f(|u|)},$$

где m — число состояний машины M_u. Кроме того, $M_u = M$, значит, M_u является распознавателем. Итак, при работе V_f на слове u может встретиться только ситуация 5 из числа описанных. Тогда $V_f(u) = 0$.

2) $u \notin L_{M_u}$. Аналогично получаем, что $M_u(w) = 0$, а $V_f(u) = 1$.

Итак, мы показали, что в наших предположениях $L_{V_f} \neq L_{M_u}$. Но этим самым мы получили противоречие с тем, что с одной стороны

$$L_{M_u} = L_{M_w} = L_M,$$

а с другой

$$L_{V_f} = L_{U_f} = L_M$$

и, значит, $L_{V_f} = L_{M_u}$.

Итак, мы показали, что язык L_{U_f} не распознается никаким 1-МТ распознавателем с емкостной сложностью, асимптотически мажорируемой функцией f.

Следовательно, L_{U_f} удовлетворяет условиям теоремы: имеет нижнюю оценку детерминированной емкостной сложности $f(n) - 1$ и верхнюю оценку $r(f(n) + n + s_f(n))$, и можно положить $L_f = L_{U_f}$. ■

Как уже говорилось, теоремы об иерархии могут быть доказаны для недетерминированных машин, а также для временной сложности. Для этого можно использовать теорему 6.1 и теоремы 6.3 и 6.4.

Теорема 6.4 является инструментом, позволяющим доказывать несовпадение различных классов языков.

Дадим определения классов языков по сложности.

Пусть фиксирован некоторый алфавит Δ и пусть f — некоторая рекурсивная функция. Мы будем рассматривать языки в алфавите Δ и следующие классы таких языков.

$DTIME(f)$ — класс всех таких языков L, что существует такой 1-МТ распознаватель M_L, что $L = L_{M_L}$ и $t_{M_L} \ll f$.

$NTIME(f)$ — класс всех таких языков L, что существует такой 1-НМТ распознаватель M_L, что $L = L_{M_L}$ и $t_{M_L} \ll f$.

$DSPACE(f)$ — класс всех таких языков L, что существует такой 1-МТ распознаватель M_L, что $L = L_{M_L}$ и $s_{M_L} \ll f$.

$NSPACE(f)$ — класс всех таких языков L, что существует такой 1-НМТ распознаватель M_L, что $L = L_{M_L}$ и $s_{M_L} \ll f$.

Из доказанных выше теорем видно, что определенные выше классы весьма неустойчивы, относительно, например, числа лент у машины Тьюринга. Поэтому обычно рассматривают классы мажорирующих функций с точностью до мультипликативной константы и соответствующие им классы языков. Это нивелирует уже различия в числе лент, а также делает эти классы устойчивыми относительно операций объединения и пересечения.

В теории сложности рассматривают и классы, мажорируемые очень медленно растущими функциями. Однако мы не будем рассматривать слишком "маленькие"классы.

Фактически нас интересуют часто встречающиеся на практике важные классы сложности, для которых структурная теория хорошо разработана. Укажем некоторые из таких классов.

Часто встречающиеся классы емкостной сложности:

$$DLOGSPACE = \bigcup_{k>0} DSPACE(klog(n));$$

$$NLOGSPACE = \bigcup_{k>0} NSPACE(klog(n));$$

$$DLINEARSPACE = \bigcup_{k>0} DSPACE(kn);$$

$$NLINEARSPACE = \bigcup_{k>0} NSPACE(kn);$$

$$PSPACE = \bigcup_{k>0} DSPACE(n^k) = \bigcup_{k>0} NSPACE(n^k);$$

$$ELEM = \bigcup_{k>0} DSPACE(\mathrm{pow}_k(n)) = \bigcup_{k>0} NSPACE(\mathrm{pow}_k(n)).$$

Мы полагаем $\mathrm{pow}_1(n) = 2^n$ и $\mathrm{pow}_{k+1}(n) = 2^{\mathrm{pow}_k(n)}$.

В классах $PSPACE$ и $ELEM$ совпадение детерминированных и недетерминированных версий следует из теоремы Сэвиджа (упражнение 6.2.1). Мы не будем подробно останавливаться на этом факте, ибо в дальнейшем изложении это совпадение никак не будет нами использоваться.

Часто встречающиеся классы временной сложности:

$$DLINEARTIME = \bigcup_{k>0} DTIME(kn);$$

$$NLINEARTIME = \bigcup_{k>0} NTIME(kn);$$

$$\mathcal{P} = PTIME = \bigcup_{k>0} DTIME(n^k);$$

$$NP = NPTIME = \bigcup_{k>0} NTIME(n^k);$$

$$ELEM = \bigcup_{k>0} DTIME(\mathrm{pow}_k(n)) = \bigcup_{k>0} NTIME(\mathrm{pow}_k(n)).$$

Следующие три класса считаются классами "практически"распознаваемых языков: $DLOGSPACE$, $NLOGSPACE$, $\mathcal{P}$. Привести резоны для этого довольно сложно, и мы не будем этого делать. Между этими тремя классами имеют место очевидные соотношения:

$DLOGSPACE \subseteq NLOGSPACE \subseteq \mathcal{P}$.

Однако неизвестно даже, совпадают ли $DLOGSPACE$ и $\mathcal{P}$. Эта цепочка включений может быть очевидно продолжена:

$DLOGSPACE \subseteq NLOGSPACE \subseteq \mathcal{P} \subseteq NP \subseteq PSPACE$.

При этом по теореме 6.4 $DLOGSPACE \subset PSPACE$, но на каком именно шаге цепочки включение строгое, неизвестно. Фактически эти вопросы являются основными в структурной теории сложности. В следующем параграфе мы более подробно остановимся на самом известном из этой серии вопросе о соотношении классов $\mathcal{P}$ и NP.

Для детерминированных машин Тьюринга определяются также классы сложности функций, аналогично классам сложности языков и с теми же обозначениями. К примеру, если $f : \{0, 1, \dots\} \to \{0, 1, \dots\}$, то $f \in \mathcal{P}$ тогда и только тогда, когда существует 1-МТ M_f, вычисляющая f, и t_M асимптотически мажорируется функцией n^k для некоторого $k > 0$ (то есть, фактически, оценивается сверху некоторым полиномом; отсюда и исходное обозначение класса $\mathcal{P}$).

Упражнения

Упражнение 6.2.1 (**; теорема Сэвиджа). *Покажите, что для емкостной сложности верхняя оценка при моделировании недетерминированной 1-МТ детерминированной может быть улучшена до квадратичной. Точнее, докажите следующую теорему, принадлежащую Сэвиджу:*

Пусть M — 1-НМТ распознаватель, распознающий L_M, пусть рекурсивная функция f является верхней оценкой функции s_M, являющейся емкостной сложностью M, и пусть s_f — емкостная сложность некоторой 1-МТ, вычисляющей f. Существует 1-МТ распознаватель M', распознающий язык L_M. Кроме того, для некоторой константы c_M (вычислимой по M), емкостная временная сложность $s_{M'}$ асимптотически мажорируется функцией $c_M(f^2(n) + n + t_f(n))$.

6.3 Сводимость языков. Универсальные задачи

Начнем с рассмотрения одного простого примера. Пусть L — некоторый язык в алфавите $\{0,1\}$ и $L' = 1 \times L = \{1w \mid w \in L\}$. Понятно, что сложность языка L' в некотором смысле почти равна сложности языка L. А именно, всякий НМТ-распознаватель L легко превратить в распознаватель L' с несущественным увеличением сложности (то же и для МТ).

Однако, вместо того, чтобы заниматься преобразованием вычислителя, можно преобразовать сам язык.

Для приведенного выше примера рассмотрим следующую 1-МТ R с алфавитом $\{0,1,\wedge\}$. Пусть $r \in \{0,1\}$, $r \notin L$ (считаем, что $L \neq \{0,1\}^*$). R читает первую букву со входной ленты. Если эта буква — 0, то R пишет на выходную ленту слово r и останавливается. В противном случае R переписывает на выходную ленту оставшуюся часть слова со входной ленты и останавливается. Итак, R вычисляет функцию $R : \{0,1\}^* \to \{0,1\}^*$,

$$R(w) = \begin{cases} w', \text{ если } w = 1w' \\ r \text{ в остальных случаях.} \end{cases}$$

Легко видеть, что $t_R(n) \leq n$ для всякого $n > |r|$, значит,

$$R \in DLINEARTIME.$$

С другой стороны, видно, что

$$w \in L' \;\Leftrightarrow\; R(w) \in L.$$

Мы будем говорить, что L' сводится к L в $DLINEARTIME$ или что L' $DLINEARTIME$-сводим к L, и писать $L' \leq_{DLINEARTIME} L$.

Дадим полное определение, обобщающее ситуацию с L и L'.

Определение 6.3.1. *Пусть Σ — некоторый алфавит; $L, L' \subseteq \Sigma^*$; $R : \Sigma^* \to \Sigma^*$. Пусть R принадлежит некоторому классу функций $\mathfrak{K}$ из Σ^* в Σ^*. Будем говорить, что L' сводится к языку L в классе $\mathfrak{K}$ или что L' $\mathfrak{K}$-сводим к L, и писать $L' \leq_{\mathfrak{K}} L$, если*

$$(\forall w \in \{0,1\}^*)(w \in L' \Leftrightarrow R(w) \in L).$$

Ниже для простоты мы ограничимся рассмотрением только детерминированных сводимостей по времени, то есть в качестве $\mathfrak{K}$ будем рассматривать классы $\bigcup_{f \in \mathfrak{F}} DTIME(f)$ для некоторых классов $\mathfrak{F}$ арифметических функций.

Определение 6.3.2. *Пусть $\mathfrak{F}$ — класс арифметических функций. Будем говорить, что $\mathfrak{F}$ мажорантно замкнут относительно*

сложения, если $(\forall f, g \in \mathfrak{F})(\exists h \in \mathfrak{F})\ (f+g) \ll h$;

подстановки, если $(\forall f, g \in \mathfrak{F})(\exists h \in \mathfrak{F})\ (f \bullet g) \ll h$, где $(f \bullet g)(n) = f(g(n))$;

умножения, если $(\forall f, g \in \mathfrak{F})(\exists h \in \mathfrak{F})\ (fg) \ll h$.

Напомним, что $\ll$ — отношение асимптотической мажорируемости функций.

Ясно, что классы функций

$$\{n^c \mid c \in \{0, 1, ..\}\}, \quad \{pow_c(n) \mid c \in \{0, 1, \ldots\}\}$$

являются мажорантно-замкнутыми относительно сложения, подстановки и умножения.

Определение 6.3.3. *Пусть $\mathfrak{F}$ — класс арифметических функций. Если $\mathfrak{F}$ является мажорантно замкнутым относительно сложения, подстановки и умножения, то классы сложности*

$$\bigcup_{f \in \mathfrak{F}} DTIME(f), \quad \bigcup_{f \in \mathfrak{F}} NTIME(f)$$

будем называть плотными.

Ясно, что в силу предыдущего замечания, классы $\mathcal{P}$, $ELEM$ являются плотными.

Лемма 6.5. *Пусть Σ — некоторый алфавит; $L, L' \subseteq \Sigma^*$; $\mathfrak{F}$ — класс функций, мажорантно замкнутый относительно сложения, умножения и подстановки; $\mathfrak{G} \subseteq \mathfrak{F}$; $\mathfrak{J} = \bigcup_{g \in \mathfrak{G}} DTIME(g)$ — класс функций.*

Пусть $\mathfrak{K} = \bigcup_{f \in \mathfrak{F}} DTIME(f)$ (или $\mathfrak{K} = \bigcup_{f \in \mathfrak{F}} NTIME(f)$) — плотный класс языков.

Если $L \in \mathfrak{K}$ и $L' \leq_{\mathfrak{J}} L$, то $L' \in \mathfrak{K}$.

Доказательство. Итак, пусть M — 1-МТ, вычисляющая такую функцию $R : \Sigma^* \to \Sigma^*$, что $w \in L' \Leftrightarrow R(w) \in L$, причем $t_M \ll g$ для некоторой $g \in \mathfrak{G}$. Пусть M_L — 1-МТ распознаватель (соответственно 1-НМТ), $L_{M_L} = L$ и $t_{M_L} \ll f$ для некоторой $f \in \mathfrak{F}$. Сконструируем 3-МТ-распознаватель M_3 языка L' по M_L и M.

Сначала M_3 работает на слове как машина M, но пишет не на выходную ленту, а на вторую рабочую ленту; первую ленту M_3 использует как рабочую для M. Затем M_3 работает как M_L, но в качестве входной ленты использует выход M, то есть вторую рабочую ленту, а третью ленту использует как рабочую для M_L.

Ясно, что $L_{M_3} = L'$.

M_3 будет детерминированной, если M_L детерминирована.

Построим верхнюю оценку времени работы машины M_3, то есть функции t_{M_3}.

Ясно, что $t^*_{M_3}(w) = t^*_M(w) + t^*_{M_L}(R(w))$. С другой стороны, очевидно, что $|R(w)| \le t^*_M(w)$. Значит, $t_{M_3}(n) \le t_M(n) + t_{M_L}(t_M(n))$.

Рассмотрим теперь 1-МТ M', моделирующую работу M_3. Как мы знаем, M' можно подобрать так, чтобы $t_{M'}(n) \le kt^2_{M_3}(n)$, при некотором k. Другими словами, существует такое k, что $t_{M'}(n) \le (kt_M(n) + kt_{M_L}(t_M(n)))^2$.

Рассмотрим мажоранты g и f функций t_M и t_{M_L} (зафиксированные в начале доказательства). Имеем, в связи с мажорантной замкнутостью $\mathfrak{F}$ относительно сложения, умножения и подстановки:

$$t_{M'}(n) \le (kg(n) + kf(g(n)))^2 \le (g_1(n) + g_2(n))^2 \le (g_3(n))^2 = g_4(n)$$

для подходящих $g_1, g_2, g_3, g_4 \in \mathfrak{F}$.

Следовательно, $t_{M'}$ мажорируется $g_4 \in \mathfrak{F}$, но $L_{M'} = L_{M_3} = L'$, то есть:

$L' \in DTIME(g_4) \subseteq \bigcup_{f\in\mathfrak{F}} DTIME(f) = \mathfrak{K}$, если L_M была детерминирована, и

$L' \in NTIME(g_4) \subseteq \bigcup_{f\in\mathfrak{F}} NTIME(f) = \mathfrak{K}$, если L_M была недетерминирована. ∎

Лемма 6.5 дает инструмент понижения верхней оценки сложности языка L. Например, пусть мы знаем, что некоторый язык $B \in \mathfrak{P}$, и подозреваем, что другой язык L также принадлежит $\mathfrak{P}$. Для доказательства последнего мы можем вместо явного построения распознавателя для L показать, что L $\mathfrak{P}$-сводим к B. Из этого, по лемме 6.5, следует, что $L \in \mathfrak{P}$.

Определение 6.3.4. *Пусть $\mathfrak{K}$ — некоторый класс языков в некотором алфавите Σ, $\mathfrak{I}$ — некоторый класс функций из Σ^* в Σ^*. Будем говорить, что язык $K \in \mathfrak{K}$ является универсальным в $\mathfrak{K}$ относительно $\mathfrak{I}$-сводимости, если* $(\forall L \in \mathfrak{K})(L \le_{\mathfrak{I}} K)$.

Нетрудно видеть, что, например, если $K \in \mathcal{P}$, $K \ne \emptyset$ и $K \ne \Sigma^*$, то K является универсальным в $\mathcal{P}$ относительно $\mathcal{P}$-сводимости.

Аналогичные тривиальные наблюдения можно привести и для других классов сложности. Интерес, однако, представляет изучение "простых"сводимостей в "сложных"классах языков. Здесь уместно чуть коснуться истории. К концу 70-х годов созрел тезис о том, что класс $\mathcal{P}$ является классом "практически могущих быть распознанными"языков. Этот тезис подкреплялся тем, что многие задачи, которые приходится решать, принадлежат именно $\mathcal{P}$.

С другой стороны, для некоторых задач тривиально показывалось, что они лежат в NP, но доказать их принадлежность $\mathcal{P}$ не удавалось.

Если $L \subseteq \Sigma^*$, $L \in \mathcal{P}$, это означает, что существует полином p такой, что $w \in L$ может быть проверено не более, чем за $p(|w|)$ шагов на 1-МТ, для всякого $w \in \Sigma^*$.

Рассмотрим конкретную такую задачу. Это — задача выполнимости булевых формул.

Напомним, что называется булевой формулой. Пусть фиксирован список $x_0, x_1, x_2, \ldots$ переменных. Переменные могут принимать значения 0 и 1.

Определение 6.3.5 (булевой формулы).

1. Всякая переменная x_i является булевой формулой.

2. Если φ, ψ — булевы формулы, то $(\varphi + \psi)$, $(-\varphi)$, $(\varphi \odot \psi)$ — булевы формулы.

3. Других булевых формул нет.

Каждый булевой формуле φ приписывается значение 0 или 1, в зависимости от значений ее переменных, индуктивно по следующим правилам:

$$\text{Зн}((\varphi + \psi)) = \begin{cases} 0, & \text{если } \text{Зн}(\varphi) = \text{Зн}(\psi) = 0, \\ 1 & \text{в остальных случаях.} \end{cases}$$

$$\text{Зн}((-\varphi)) = \begin{cases} 0, & \text{если } \text{Зн}(\varphi) = 1, \\ 1 & \text{в остальных случаях.} \end{cases}$$

$$\text{Зн}((\varphi \odot \psi)) = \begin{cases} 1, & \text{если } \text{Зн}(\varphi) = \text{Зн}(\psi) = 1, \\ 0 & \text{в остальных случаях.} \end{cases}$$

Пусть φ — некоторая булева формула. Будем записывать ее в алфавите $\Sigma = \{0, 1,), (, +, \odot, -\}$ в виде код(φ), где:

код$(x_i) = i$ (в двоичной записи из 0 и 1),

код$((\varphi + \psi)) =$ (код$(\varphi)+$ код(ψ)),

код$((-\varphi)) = (-$ код$(\varphi))$,
код$((\psi \odot \varphi)) = ($ код$(\psi)\odot$ код$(\varphi))$.

Таким образом,

код$((x_0+((x_2\odot x_3)+((-x_5)\odot(x_1+x_2))))) = (0+((10\odot 11)+((-101)\odot (1+10))))$.

Проблема выполнимости.

Дана булева формула φ. Пусть $x_0, x_1, \dots x_n$ — ее переменные. Определить, существует ли такая функция Зн:$\{x_0, x_1, \dots, x_n\} \to \{0,1\}$, что Зн$(\varphi) = 1$. Если такая функция Зн существует, φ называется выполнимой. Пусть Sat — множество всех выполнимых φ,

$$SAT = \text{Код}(Sat) = \{\text{Код}(\varphi) \mid \varphi \in Sat\} \subseteq \Sigma^*.$$

Лемма 6.6. $SAT \in NP$.

Доказательство. Опишем 2-НМТ - распознаватель B. Для простоты полагаем, что на вход распознавателю будут подаваться только коды корректных булевых формул (в действительности проблема распознавания того, что данное слово является кодом корректной булевой формулы, легко решается 1-МТ с полиномиальным временем работы).

1. Сначала B переписывает входную ленту на 1-ю рабочую.

2. Затем B идет слева направо по 1-й рабочей ленте и, видя последовательность 0 и 1, смотрит, есть ли уже такая последовательность на 2-й рабочей ленте. Если есть, идет дальше, а если нет, то переписывает ее на 2-ю рабочую ленту справа, и дописывает справа от этой последовательности на второй рабочей ленте три буквы

$-\odot+$

Рассмотрим приведенный выше пример с формулой φ :

$$(x_0 + ((x_2 \odot x_3) + ((-x_5) \odot (x_1 + x_2)))).$$

Итак, на входной ленте было записано:

$\cdots \wedge (0 + ((10 \odot 11) + ((-101) \odot (1 + 10)))) \wedge \dots$

Тогда после выполнения описанных двух этапов входная лента скопируется на 1-ю рабочую, а на второй рабочей будет записано:

$\wedge 0 - \odot + 10 - \odot + 11 - \odot + 101 - \odot + 1 - \odot + \wedge \dots$

3. Теперь машина B проходит по второй ленте и ставит вместо каждого знака $\odot$ либо 0, либо 1 (недетерминировано).

В описываемом примере после этого вторая рабочая лента может выглядеть, например, так:

$\cdots \wedge 0 - 0 + 10 - 0 + 11 - 1 + 101 - 0 + 1 - 0 + \wedge \dots$

а может и по другому. Описываемый проход был единственной недетерминированной частью машины .

4. Теперь B идет по первой ленте, и всякую последовательность нулей и единиц заменяет на 0 или 1 (на одну букву!) так:

Если ей нужно заменить последовательность i из 0 и 1, она находит на 2-й ленте последовательность $i-$ и меняет на первой ленте i на букву, стоящую правее $i-$ на второй ленте.

Так как при этой процедуре на 1-й ленте могут образоваться "дырки", B сдвигает слово на 1-ой ленте, чтобы закрыть эти дырки.

Продолжим рассмотрение нашего примера. В нем, после проведения последнего шага, на 1-й ленте будет записано слово

$\cdots \wedge (0 + ((0 \odot 1) + ((-0) \odot (0 + 0)))) \wedge \ldots$

5. Теперь B проходит 1-ю ленту, и меняет везде слово "-0"на "1", а "-1"на "0". При этом производится "сдвижка"ленты, чтобы опять "закрыть дыры". В нашем примере будет:

$\cdots \wedge (0 + ((0 \odot 1) + ((1) \odot (0 + 0)))) \wedge \ldots$

6. Теперь B проходит 1-ю ленту, и меняет всюду "(0) "на "0", а "(1) "на "1". При этом происходит сдвижка. В нашем примере будет:

$\cdots \wedge (0 + ((0 \odot 1) + (1 \odot (0 + 0)))) \wedge \ldots$

7. Теперь B проходит -ю ленту и меняется (со сдвижкой)

"$0 \odot 0$ "на "0",

"$0 \odot 1$ "на "0",

"$1 \odot 0$ "на "0",

"$1 \odot 1$ "на "1".

В нашем примере будет:

$\cdots \wedge (0 + ((0) + (1 \odot (0 + 0)))) \wedge \ldots$

8. Аналогично 6.

В нашем примере будет:

$\cdots \wedge (0 + (0 + (1 \odot (0 + 0)))) \wedge \ldots$

9. Теперь B проходит 1-ю ленту и меняет (со сдвижкой):

"$0 + 0$ "на "0",

"$0 + 1$ "на "1",

"$1 + 0$ "на "1",

"$1 + 1$ "на "1".

В нашем примере будет:

$\cdots \wedge (0 + (0 + (1 \odot (0)))) \wedge \ldots$

10. Аналогично 6.

В нашем примере будет:

$\cdots \wedge (0 + (0 + (1 \odot 0))) \wedge \ldots$

11. Если на 1-й ленте осталось

$\cdots \wedge 1 \wedge \ldots,$

то B пишет на выходную ленту 1 и останавливается. Если осталось

$\cdots \wedge 0 \wedge \ldots,$

то B пишет на выходную ленту 0 и останавливается. Иначе B возвращается к шагу 5.

В нашем примере B возвращается к шагу 5. Далее будет:

Шаг 5. Ничего.

Шаг 6. Ничего.

Шаг 7. $\cdots \wedge (0 + (0 + (0))) \wedge \ldots$

Шаг 8. $\cdots \wedge (0 + (0 + 0)) \wedge \ldots$

Шаг 9. $\cdots \wedge (0 + (0)) \wedge \ldots$

Шаг 10. $\cdots \wedge (0 + 0) \wedge \ldots$

Шаг 11. Переход на шаг 5. Шаг 5. Ничего.

Шаг 6. Ничего.

Шаг 7. Ничего.

Шаг 8. Ничего.

Шаг 9. $\cdots \wedge (0) \wedge \ldots$

Шаг 10. $\cdots \wedge 0 \wedge \ldots$

Шаг 11. B напишет 0 на выходную ленту и остановится.

Однако в действительности наш пример принадлежал SAT. Чтобы убедиться в этом читателям достаточно рассмотреть выполнение B в том случае, когда на шаге 3 на 2-й ленте было порождено:

$\cdots \wedge 0 - 1 + 10 - 0 + 11 - 1 + 101 - 0 + 1 - 0 + \wedge \ldots$

Ясно, что при этом B остановится и напишет 1 на входную ленту.

Оценим теперь время работы B на слове w длины n.

Шаг 1. Длится n тактов ровно.

Шаг 2. Длится не более $2n^4 + 6n^3 + n^2 + 3$ шагов (всего вхождений переменных $\leq n$, для каждой нужно прочесть 2-ю ленту ($\leq 2n(n+3) = 2n^2 + 6n$ шагов) и, может быть, записать $\leq n + 3$ букв, так как длина любой переменной $\leq n$). При этом могут быть возвраты ($\leq n$) на 1-й ленте.

Шаг 3. $\leq 2n^2 + 3n$ (недетерминированно!).

Шаг 4. $\leq 2n^4 + 6n^3 + 3n^2 + n + 4$.

Шаг 5. $\leq 2n^2 + 2n$.

Шаг 6. $\leq 2n^2 + 2n$.

Шаги 7-10. $\leq 2n^2 + 2n$.

Шаг 11. $\leq n + 2$.

Поскольку каждое повторение шагов 5-10 сокращает слово на 1-й ленте хотя бы на 1, то число этих повторов не превосходит n. Итак:

$t_B(n) \leq n + 2n^4 + 6n^3 + n^2 + 3 + 2n^2 + 3n + 2n^4 + 6n^3 + 3n^2 + n + 4 + 6n(2n_n^2 + 2n) + n + 2$.

Ясно, что $t_B \ll n^5$.

Мы знаем, что можно подобрать такую 1-НМТ B', что $L_B = L_{B'}$ и $t_{B'} \ll k(n^5)^2$ при подходящем k. Значит, $t_{B'} \ll n^{11}$.

Однако легко видеть, что $L_B = SAT$, значит, $L_{B'} = SAT$. ∎

Мы будем использовать для записи булевых формул сокращения, состоящие в опускании скобок, считая при этом, что сильнее всего связывает $-$, затем $\odot$ и затем $+$. В силу коммутативности операций $\odot$ и $+$ несущественно, выполняются ли они слева направо или справа налево.

Итак, мы показали, что $SAT \in NP$.

До сих пор никому не удалось доказать или опровергнуть принадлежность языка SAT к классу $\mathcal{P}$. Между тем этой проблемой занимаются весьма активно в течение по меньшей мере 30 лет. Причина столь пристального интереса к проблеме $SAT \in \mathcal{P}$ в том, что SAT оказывается универсальной задачей в классе NP относительно $\mathcal{P}$-сводимости (смотри теорему 6.7 ниже).

Другими словами, если $SAT \in \mathcal{P}$, то $\mathcal{P} \equiv NP$ и, конечно, наоборот. За время, прошедшее со времени доказательства этих результатов, была показана принадлежность к классу NP большого числа задач, в том числе практически очень важных. Видимо, можно сказать, что проблема $\mathcal{P} \equiv NP$? (а, значит, и $SAT \in \mathcal{P}$?) является главной открытой проблемой теории сложности вычислений.

На примере языка SAT хорошо видно соотношение между $\mathcal{P}$ и NP. Так, если нам уже предложена некоторая функция Зн, определенная на переменных булевой формулы φ, то проверить, что Зн$(\varphi) = 1$, то есть что Зн является решением φ, можно в классе $\mathcal{P}$. Таким образом, NP в некотором смысле является классом задач, проверить корректность решения которых можно в классе $\mathcal{P}$.

Но вернемся к языку SAT.

Теорема 6.7. *Язык SAT является универсальным в классе NP относительно $\mathcal{P}$-сводимости.*

Доказательство. Мы должны показать, что всякий язык $L \in NP$ во всяком таком алфавите Σ, что $\{0, 1,), (, +, \odot, -\} \subseteq \Sigma$, $\mathcal{P}$-сводим к языку SAT.

Рассмотрим для простоты некоторый язык $L \in NP$, $L \subseteq \{0, 1\}^*$.

Общий случай рассматривается аналогично, но возникают дополнительные сложности с кодированием, несущественные для идеи доказательства.

Поскольку $L \in NP$, то существуют такие 1-НМТ распознаватель M и полином p, что $L_M = L$ и $(\forall n) t_M(n) \leq p(n)$. Последнее следует из того, что t_M асимптотически мажорируется некоторой степенью n, скажем, $t_M \ll n^k$; но тогда очевидно, что $(\exists l)(\forall n)(t_M(n) \leq n^k + l)$. Следовательно, можно в качестве $p(n)$ взять $n^k + l$.

Мы хотим построить 1-МТ R такую, что $(\exists r)(t_R \ll n^r)$ и R вычисляет такую функцию $r : \{0,1\}^* \to \{0,1,),(,+,\odot,-\}^*$, что если $w \in \{0,1\}^*$, то $w \in L \Leftrightarrow r(w) \in SAT$.

Фактически такая 1-МТ R будет строиться по двум объектам: по 1-НМТ M и по полиному p (смотри выше).

Начнем с построения, для каждого $w \in \{0,1\}^*$, такой булевой формулы φ_w, что $w \in L \Leftrightarrow \varphi_w \in Sat$.

Итак, пусть

$$M = \langle \{0,1\}, Q, \pi, q_0 \rangle,$$

где:

Q — алфавит состояний,

$$Q = \{q_0, q_1, \ldots, q_m\},$$

$q_0 \in Q$ — начальное состояние,

$$\pi : (\{\wedge, 0, 1\} \times \{\wedge, 0, 1\} \times Q) \to$$

$$\mathfrak{P}(\{\wedge, 0, 1\} \times Q \times \{L, R, N, S\} \times \{L, R, N\} \times \{\wedge, 0, 1\})$$

есть функция переходов, где $\mathfrak{P}(Z)$ — множество всех подмножеств Z.

Пусть $w \in \{0,1\}^*$, $w = \alpha_1 \alpha_2 \ldots \alpha_n$, где $\alpha_i \in \{0,1\}$.

Булева формула φ_w будет иметь переменные (мы просто поясняем сперва "смысл"переменных):

$a_1, \ldots, a_n$ для запоминания входной ленты;

$\{r^j_{i,k} \mid i,j \in \{0,1,\ldots,p(n)\}, k \in \{0,1\}\}$ для запоминания рабочей ленты в различные моменты времени;

$\{p^j_i \mid i \in \{0,1,\ldots,n+1\}, j \in \{0,1,\ldots,p(n)\}\}$ для запоминания позиции на входной ленте в различные моменты времени;

$\{v^j_i \mid i,j \in \{0,1,\ldots,p(n)\}\}$ для запоминания позиции на рабочей ленте в различные моменты времени;

$\{z^j \mid j \in \{0,1,\ldots,p(n)\}\}$ для запоминания остановки;

$\{\sigma^j_0, \sigma^j_1 \mid j \in \{0,1,\ldots,p(n)\}\}$ для запоминания выходной ленты;

$\{q^j_0, \ldots, q^j_m \mid j \in \{0,1,\ldots,p(n)\}\}$ для запоминания состояний в различные моменты времени.

Начнем со словесных пояснений. Машина M работает на слове w такт за тактом. После каждого такта дальнейшее поведение M полностью определяется ее конфигурацией, то есть пятеркой $\langle w, i, u, j, q\rangle$:

1) содержимым входной ленты (w);

2) положением головки на входной ленте (то есть числом i от 0 до $n+1$);

3) содержимым рабочей ленты (некоторое слово $u = \beta_1\beta_2\ldots\beta_l$, где $l \leq p(n)$);

4) положением головки на рабочей ленте (то есть числом j от 0 до $p(n)$);

5) состоянием q машины M ($q \in \{q_0, q_1, \ldots, q_m\}$).

Всякое останавливающееся вычисление M на w (которое нас интересует) есть последовательность конфигураций:

$$\langle w, i_0, u_0, j_0, q_0\rangle, \langle w, i_1, u_1, j_1, q_{s_1}\rangle, \ldots, \langle w, i_\theta, u_\theta, j_\theta, q_{s_\theta}\rangle,$$

где $\theta \leq p(n)$.

Эти конфигурации связаны естественными условиями, попарно в последовательности; условия определяются функцией переходов π. Есть также условия на начальную конфигурацию и на конечную.

Конфигурация $\langle w, i_\gamma, u_\gamma, j_\gamma, q_{s_\gamma}\rangle$ будет кодироваться при помощи значений переменных с верхним индексом γ или вообще без верхнего индекса следующим образом.

Пусть $w = \alpha_1\ldots\alpha_n$, $u_\gamma = \beta_1\ldots\beta_l$, где $l \leq p(n)$. Тогда:

1) Зн$(a_k) = \alpha_k$;

2)

$$\text{Зн}(r^\gamma_{k,0}) = \begin{cases} 0, \text{ если } k > l \text{ или } \beta_k = \wedge, \\ 1 \text{ в остальных случаях;} \end{cases}$$

$$\text{Зн}(r^\gamma_{k,1}) = \begin{cases} 0, \text{ если Зн}(r^\gamma_{k,0}) = 0, \\ \beta_k \text{ в остальных случаях;} \end{cases}$$

3)

$$\text{Зн}(p^\gamma_k) = \begin{cases} 1, \text{ если } k = i_\gamma, \\ 0 \text{ в остальных случаях;} \end{cases}$$

4)

$$\text{Зн}(v^\gamma_k) = \begin{cases} 1, \text{ если } k = j_\gamma, \\ 0 \text{ в остальных случаях;} \end{cases}$$

5)

$$\text{Зн}(q_k^\gamma) = \begin{cases} 1, \text{ если } q_{s_\gamma} = q_k, \\ 0 \text{ в остальных случаях.} \end{cases}$$

Рассмотрим формулу φ_0 :

$$(a_1 = \alpha_1 \odot a_2 = \alpha_2 \odot \cdots \odot a_n = \alpha_n) \odot$$

$$(-r^0_{1,0} \odot -r^0_{1,1} \odot \cdots \odot -r^0_{p(n),0} \odot -r^0_{p(n),1}) \odot$$

$$(-p^0_0 \odot p^0_1 \odot -p^0_2 \odot \cdots \odot -p^0_n \odot -p^0_{n+1}) \odot (v^0_1 \odot -v^0_2 \odot \ldots . \odot -v^0_{p(n)}) \odot$$

$$(q^0_0 \odot -q^0_1 \odot -q^0_2 \odot \cdots \odot -q^0_m).$$

Ее легко превратить в булеву формулу, заменив везде $a_k = 1$ на a_k, а $a_k = 0$ на $-a_k$.

Полученную булеву формулу обозначим φ^0_w.

Ясно, что

Зн$(\varphi^0_w) = 1 \Leftrightarrow$ функция Зн описывает начальную конфигурацию M на слове w.

Напишем теперь для каждого $\gamma \leq p(n)$, такую формулу φ^γ_w, что

Зн$(\varphi^\gamma_w) = 1 \Leftrightarrow$ Зн описывает конфигурацию после γ-го шага "правдоподобно", то есть что Зн однозначно указывает состояние $q \in Q$ и однозначно же — положение головок на входной и рабочей лентах.

Итак, φ^γ_w есть:

$$((q^\gamma_0 \odot -q^\gamma_1 \odot -q^\gamma_2 \odot \cdots \odot -q^\gamma_m) + (-q^\gamma_0 \odot q^\gamma_1 \odot -q^\gamma_2 \odot \cdots \odot -q^\gamma_m) +$$

$$(-q^\gamma_0 \odot -q^\gamma_1 \odot q^\gamma_2 \odot -q^\gamma_3 \odot \cdots \odot -q^\gamma_m) + \cdots + (-q^\gamma_0 \odot -q^\gamma_1 \odot \cdots \odot -q^\gamma_{m-1} \odot q^\gamma_m)) \odot$$

$$((p^\gamma_0 \odot -p^\gamma_1 \odot \cdots \odot -p^\gamma_n \odot -p^\gamma_{n+1}) + (-p^\gamma_0 \odot p^\gamma_1 \odot -p^\gamma_2 \cdots \odot -p^\gamma_{n+1}) +$$

$$(-p^\gamma_0 \odot -p^\gamma_1 \odot p^\gamma_2 \odot -p^\gamma_3 \odot \cdots \odot -p^\gamma_{n+1}) + \cdots + (-p^\gamma_0 \odot -p^\gamma_1 \odot \cdots \odot -p^\gamma_n \odot p^\gamma_{n+1})) \odot$$

$$((v^\gamma_1 \odot -v^\gamma_1 \odot \cdots \odot -v^\gamma_{p(n)}) + (-v^\gamma_1 \odot v^\gamma_2 \odot -v^\gamma_3 \odot \cdots \odot -v^\gamma_{p(n)}) +$$

$$(-v^\gamma_1 \odot -v^\gamma_2 \odot v^\gamma_3 \odot -v^\gamma_4 \cdots \odot -v^\gamma_{p(n)}) + \cdots + (-v^\gamma_1 \odot -v^\gamma_2 \odot \cdots \odot -v^\gamma_{p(n)-1} \odot v^\gamma_{p(n)})).$$

Запишем теперь для каждого такого γ, что $1 \leq \gamma \leq p(n)$, булеву формулу ψ^γ_w, которая будет "говорить", что γ-я конфигурация может быть получена из $(\gamma - 1)$-й при помощи функции π.

Сначала для каждой тройки $\langle \delta, \rho, q_i \rangle$ и каждой пятерки $\langle \alpha, q_j, f, g, \beta \rangle$, где

$$\alpha, \beta, \delta, \rho \in \{\wedge, 0, 1\}, \quad q_i, q_j \in Q, \quad f \in \{L, R, N, S\}, \quad g \in \{L, R, N\},$$

$$\langle \alpha, q_j, f, g, \beta \rangle \in \pi(\delta, \rho, q_i),$$

напишем формулу

$$\psi_w^\gamma(\delta, \rho, q_i, \alpha, q_j, f, g, \beta),$$

говорящую, что γ-я конфигурация получена из предыдущей в точности по указанному правилу. Написание этой формулы рутинно и предоставляется читателю. Напишем, к примеру, что в $(\gamma-1)$-й конфигурации на рабочей ленте обозревался символ 0:

$$(v_1^{\gamma-1} \odot r_{1,0}^{\gamma-1} \odot -r_{1,1}^{\gamma-1} + v_2^{\gamma-1} \odot r_{2,0}^{\gamma-1} \odot -r_{2,1}^{\gamma-1} + \cdots + v_{p(n)}^{\gamma-1} \odot r_{p(n),0}^{\gamma-1} \odot -r_{p(n),1}^{\gamma-1}).$$

Аналогичным способом можно записать все условия, обеспечивающие, что γ-я конфигурация получена из $(\gamma-1)$-й по правилу:

$$\langle \delta, \rho, q_i \rangle \to \langle \alpha, q_j, f, g, \beta \rangle.$$

Дополнительно припишем к формуле

$$\psi_w^\gamma(\delta, \rho, q_i, \alpha, q_j, f, g, \beta),$$

используя знак $\odot$ в качестве разделителя:

1. Если $\beta = \wedge$, то $-o_0^\gamma$.
2. Если $\beta = 0$, то $(o_0^\gamma \odot -o_1^\gamma)$.
3. Если $\beta = 1$, то $(o_0^\gamma \odot o_1^\gamma)$.
4. Если $f = S$, то z^γ, иначе $-z^\gamma$.

Другими словами, рассмотрим взятое в скобки произведение формулы

$$\psi_w^\gamma(\delta, \rho, q_i, \alpha, q_j, f, g, \beta)$$

и формул, указанных в пунктах 1—4. Полученную формулу обозначим через

$$\psi_{w,1}^\gamma(\delta, \rho, q_i, \alpha, q_j, f, g, \beta).$$

Таким образом, будет проверено, если это необходимо, что M писала что-либо на рабочую ленту, и что M остановилась.

Теперь в качестве ψ_w^γ рассмотрим взятую в скобки сумму всех формул

$$\psi_{w,1}^\gamma(\delta, \rho, q_i, \alpha, q_j, f, g, \beta)$$

для всевозможных $\delta, \rho, q_i, \alpha, q_j, f, g, \beta$ как выше.

Рассмотрим теперь для всех таких γ, что $1 \le \gamma \le p(n)$, формулы τ_w^γ, где τ_w^γ есть:

$$(\varphi_w^0 \odot \varphi_w^1 \odot \cdots \odot \varphi_w^\gamma) \odot$$

$$(\psi_w^1 \odot \psi_w^2 \odot \cdots \odot \psi_w^\gamma)\odot$$
$$(-z^1 \odot -z^2 \odot \cdots \odot -z^{\gamma-1} \odot z^\gamma)\odot$$
$$(o_0^1 \odot o_1^1 + o_0^2 \odot o_1^2 + \cdots + o_0^\gamma \odot o_1^\gamma).$$

Первая строчка τ_w^γ "говорит", что значениями переменных корректно описывается последовательность длины $(\gamma+1)$ последовательных конфигураций машины M на слове w, причем первая конфигурация — начальная.

Вторая строчка говорит, что каждая из описанных последовательностей действительно может быть получена M при работе на слове w.

Третья говорит, что M при порождении этих последовательностей останавливается точно через γ тактов.

Четвертая говорит, что M при прохождении этих последовательностей пишет 1 на выходную ленту.

Итак, ясно, что сумма

$$(\tau_w^1 + \tau_w^2 + \tau_w^3 + \cdots + \tau_w^{p(n)})$$

может быть взята в качестве искомой формулы φ_w с требуемыми свойствами.

Из построения φ_w видно, что φ_w может быть построена за полином от $|w| = n$ шагов (конечно, от размера M φ_w зависит более, чем полиномиально).

В проводимом построении формул мы опускали некоторые скобки, руководствуясь соглашением об очередности применения операций. Это, конечно, делалось только для увеличения наглядности формул и можно все вышеприведенные формулы выписать со всеми скобками. При этом, конечно, сохранится полиномиальность зависимости длины окончательной формулы φ_w от n.

Запись описанного выше алгоритма построения φ_w по w на 1-МТ, то есть построение 1-МТ, строящей по w слово код(φ_w), представляет собой технически трудоемкую, но рутинную процедуру, которую читатель потенциально может осуществить. В результате можно убедиться, что такая 1-МТ R будет построена и что $t_M \ll n^k$ для некоторого k, зависящего от M и $p(n)$ (от $p(n)$ полиномиально, а от M — не полиномиально). ∎

SAT является далеко не единственным языком, универсальным в NP относительно $\mathcal{P}$-сводимости. К настоящему времени известны сотни таких языков. Некоторые наиболее часто упоминаемые из этих языков приведены в упражнениях. Коль скоро нами доказана универсальность SAT в NP относительно $\mathcal{P}$-сводимости, нам для доказательства универсальности другого языка U в NP относительно $\mathcal{P}$-сводимости достаточно, очевидно, показать, что SAT $\mathcal{P}$-сводим к U. Итак, теорема 3 дает нам в руки инструмент для доказательства универсальности языков в классе NP.

В литературе обычно языки, универсальные в NP относительно $\mathcal{P}$-сводимости, называют NP-полными.

Упражнения

Упражнение 6.3.1. *Булеву формулу φ будем называть:*

атомной, если она есть либо x_i, либо $-x_i$ для некоторой переменной i;

суммой, если φ есть $\psi_1 + \psi_2 + \cdots + \psi_n$ для некоторого $n \geq 1$ и некоторых атомных формул $\psi_1, \psi_2, \ldots, \psi_n$;

k-суммой, если φ есть $\psi_1 + \psi_2 + \cdots + \psi_n$ для некоторого $n \geq 1$, $n \leq k$ и некоторых атомных формул $\psi_1, \psi_2, \ldots, \psi_n$;

конъюнктивной нормальной формой (кнф), если φ есть $(\psi_1) \odot (\psi_2) \odot \cdots \odot (\psi_n)$ для некоторого $n \geq 1$ и некоторых сумм $\psi_1, \psi_2, \ldots, \psi_n$;

k-конъюнктивной нормальной формой (k-кнф), если φ есть $(\psi_1) \odot (\psi_2) \odot \cdots \odot (\psi_k)$ для некоторых сумм $\psi_1, \psi_2, \ldots, \psi_k$.

Покажите, что подъязык $CNF \subseteq SAT$, составленный из кодов конъюнктивных нормальных форм, универсален в NP относительно $\mathcal{P}$-сводимости, то есть NP-полон.

Упражнение 6.3.2 (*). *Пусть $CNF(k) \subseteq SAT$, $CNF(k)$ состоит из кодов k-конъюнктивных нормальных форм. Покажите, что $CNF(2) \in \mathcal{P}$.*

Упражнение 6.3.3 (*). *Покажите, что $CNF(3)$ NP-полон.*

Упражнение 6.3.4 (*). *Ориентированным (конечным) графом называется конечное множество G вместе с некоторым подмножеством E декартова квадрата G^2.*

Циклом в графе $\mathfrak{G} = \langle G, E \rangle$ называется такая последовательность $g_1, g_2, \ldots, g_n \in G$, что все $g_1, \ldots, g_n$ попарно различны, для каждого $i = 1, 2, \ldots, n-1$ пара $\langle g_i, g_{i+1} \rangle \in E$ и пара $\langle g_n, g_1 \rangle \in E$. Цикл называется гамильтоновым, если $G = \{g_1, \ldots, g_n\}$.

Предложите такую кодировку всех графов словами в некотором конечном алфавите, что если Γ — язык, составленный из кодов тех графов, в которых есть гамильтонов цикл, то Γ NP-полон.

Упражнение 6.3.5. *Постройте упоминаемый в доказательстве леммы 6.6 1-ДМТ распознаватель с полиномиальным временем работы, распознающий коды корректных булевых формул.*

6.4 Характеризация классов временной сложности в терминах стековых машин

Результаты данного параграфа носят более специальный характер. Фактически здесь будет установлена тонкая взаимосвязь между временной и емкостной сложностями. Эта связь гораздо точнее той, что была установлена в §6.1. Однако характеризация здесь дается не с помощью машин Тьюринга, а в терминах стековых машин.

Стековая машина — это гибрид машины Тьюринга и автомата с магазинной памятью. Оба эти объекта нам хорошо знакомы. Конечно, стек (магазин) может быть промоделирован на рабочей ленте МТ, но это требует времени и памяти. Таким образом, если, например, память МТ ограничена, то использование магазина дает ей дополнительные возможности.

Но перейдем к формальным определениям.

Детерминированная стековая k-машина, сокращенно k-DAPDA (от английского Deterministic Auxiliary Push-Down Automaton) — это четверка $\langle \Sigma, Q, \pi, q_0 \rangle$, где:

Σ — (конечный) алфавит;

Q — (конечный) алфавит состояний;

$q_0 \in Q$ — начальное состояние;

π — программа, а именно отображение:

$$\pi : (\Sigma \times \Sigma \times (\Sigma \bigcup \{\dagger\}) \times Q) \to$$

$$(\Sigma \times Q \times \{L, R, N, S\} \times \{L, R, N\} \times \{1, 2, \dots, k\} \times (\Sigma \bigcup \{\dagger\})).$$

Алфавит Σ содержит букву $\wedge$, называемую "пусто", и еще по крайней мере две буквы, для определенности 0 и 1.

Содержательно k-DAPDA представляет собой машину с одной входной, k рабочими лентами и одним стеком (магазином). Каждая рабочая лента снабжена головкой. В начальный момент времени на входной ленте записано некоторое слово $w \in (\Sigma \setminus \{\wedge\})^*$, ограниченное справа и слева буквами $\wedge$, и головка на входной ленте стоит на самой левой букве слова w. На всех рабочих лентах записаны буквы $\wedge$ во всех ячейках (все рабочие ленты пусты) и первая рабочая лента является активной. Стек в начальный момент времени пуст, и машина находится в состоянии q_0.

Рассмотрим такт работы k-DAPDA $D = \langle \Sigma, Q, \pi, q_0 \rangle$.

Пусть на входной ленте обозревается клетка с буквой $a \in \Sigma$, активна i-я рабочая лента, на ней головка обозревает клетку с буквой $b \in \Sigma$ и машина находится в состоянии $q \in Q$. Пусть α есть верхний символ стека, $\alpha \in \Sigma$ или, если стек пуст, α есть $\dagger$. Пусть

$$\pi(a, b, \alpha, q) = \langle b', q', \mu_1, \mu_2, j, \beta \rangle,$$

где $b' \in \Sigma$, $q' \in Q$, $\mu_1 \in \{L, R, N, S\}$, $\mu_2 \in \{L, R, N\}$, $j \in \{1, 2, \ldots, k\}$, $\beta \in (\Sigma \bigcup \{\dagger\})$.

Тогда, на рассматриваемом такте, D пишет на i-ой рабочей ленте в обозреваемой клетке букву b' и переходит в состояние q'. Кроме того:

$$\text{Если } \mu_1 = \begin{cases} S, & \text{то } D \text{ останавливается;} \\ L, & \text{то } D \text{ сдвигает головку на } i\text{-ой рабочей ленте} \\ & \quad \text{на одну клетку влево;} \\ R, & \text{то } D \text{ сдвигает головку на } i\text{-ой рабочей ленте} \\ & \quad \text{на одну клетку вправо;} \\ N, & \text{то } D \text{ оставляет головку на } i\text{-ой рабочей ленте} \\ & \quad \text{неподвижной.} \end{cases}$$

$$\text{Если } \mu_2 = \begin{cases} L, & \text{то } D \text{ сдвигает головку на входной ленте} \\ & \quad \text{на одну клетку влево;} \\ R, & \text{то } D \text{ сдвигает головку на входной ленте} \\ & \quad \text{на одну клетку вправо;} \\ N, & \text{то } D \text{ оставляет головку на входной ленте} \\ & \quad \text{неподвижной.} \end{cases}$$

Затем D делает i-ю ленту неактивной, а j-ю ленту — активной.

Наконец, если $\beta \in \Sigma$, то D записывает β в вершину стека (тем самым стек углубляется на 1, если $\beta \neq \wedge$, а в случае $\beta = \wedge$ стек не изменяется). Если же $\beta = \dagger$, то D выталкивает верхний элемент из стека (и укорачивает стек на 1), если стек не пуст. Пустой стек в этом случае не изменяется.

После этого, если D еще не остановилась, она переходит к следующему такту.

Говорят, что D принимает слово $w \in (\Sigma \backslash \{\wedge\})^*$, если, начиная работу на слове w на входной ленте, D останавливается.

По аналогии с недетерминированными машинами Тьюринга, вводятся в рассмотрение недетерминированные стековые k-машины, сокращенно k-NAPDA. Детали их определения мы оставляем читателю.

Если D есть k-DAPDA (k-NAPDA) для некоторого k, будем называть D также детерминированной (недетерминированной) стековой машиной, сокращенно DAPDA (NAPDA), без упоминания числа лент k.

Пусть D — DAPDA (NAPDA). Языком D, сокращенно L_D или $L(D)$, будем называть множество слов, принимаемых D (для недетерминированного случая, как обычно, считаем, что слово воспринимается D тогда и только тогда, когда существует путь с данным словом на входной ленте, по которому D останавливается).

Ясно, что стек может быть промоделирован еще на одной рабочей ленте, а, как нам известно, многоленточная МТ (НМТ) сводится к одноленточной. Поэтому очевидна

Теорема 6.8. *Пусть $L \subseteq (\Sigma \setminus \{\wedge\})^*$ — некоторый язык. Следующие условия эквивалентны:*

1. Существует DAPDA (NAPDA) D, что $L(D) = L$;

2. Существует 1-МТ (1-НМТ) M, что $L(M) = L$.

Однако ситуация меняется при ограничениях на ленточную сложность.

Определение 6.4.1. *Пусть s — некоторая арифметическая функция. Рассмотрим модификацию DAPDA, в которой рабочие ленты конечны, а именно, каждая рабочая лента состоит ровно из $s(|w|)$ клеток при работе на произвольном слове $w \in (\Sigma \setminus \{\wedge\})^*$. При этом определение DAPDA модифицируется так. Пусть $D = \langle \Sigma, Q, \pi, q_0 \rangle$, где:*

Σ — (конечный) алфавит;

Q — (конечный) алфавит состояний;

$q_0 \in Q$ — начальное состояние.

Функция π порождает не 6-ку, а 7-ку:

$$\pi : (\Sigma \times \Sigma \times (\Sigma \bigcup \{\dagger\}) \times Q) \to$$

$$(\Sigma \times Q \times \{L, R, N, S\} \times \{L, R, N\} \times \{1, 2, \ldots, k\} \times (\Sigma \bigcup \{\dagger\}) \times Q).$$

Таким образом, π выдает дополнительно еще одно состояние. Вначале головки на всех рабочих лентах находятся в самых левых клетках. Неформально, работа машины происходит так же, как в неограниченном случае, если не делается попытки выйти за пределы рабочей *ленты. Если же такая попытка делается, то переход не производится, и машина переходит в состояние, определенное 7-м (а не 2-м) элементом 7-ки.*

Формально, пусть на входной ленте обозревается клетка с буквой $a \in \Sigma$, *активна* i*-я рабочая лента, на ней головка обозревает клетку с буквой* $b \in \Sigma$ *и машина находится в состоянии* $q \in Q$. *Пусть* α *есть верхний символ стека,* $\alpha \in \Sigma$ *или, если стек пуст,* α *есть* $\dagger$. *Пусть*

$$\pi(a, b\alpha, 0, q) = \langle b', q', \mu_1, \mu_2, j, \beta, q'' \rangle,$$

где $b' \in \Sigma$; $q', q'' \in Q$; $\mu_1 \in \{L, R, N, S\}$; $\mu_2 \in \{L, R, N\}$; $j \in \{1, 2, \ldots, k\}$; $\beta \in \Sigma \bigcup \{\dagger\}$.

Тогда, на рассматриваемом такте, D *пишет на* i*-ой рабочей ленте в обозреваемой клетке букву* b'. *Кроме того:*

$$\text{Если } \mu_1 = \begin{cases} S, & \text{то } D \text{ останавливается;} \\ L, & \text{то } D \text{ пытается сдвинуть головку на } i\text{-ой рабочей} \\ & \quad \text{ленте на одну клетку влево;} \\ R, & \text{то } D \text{ пытается сдвинуть головку на } i\text{-ой рабочей} \\ & \quad \text{ленте на одну клетку вправо;} \\ N, & \text{то } D \text{ оставляет головку на } i\text{-ой рабочей ленте} \\ & \quad \text{неподвижной.} \end{cases}$$

Если при этом машина пыталась сдвинуть головку на рабочей ленте, и эта попытка была неуспешной из-за того, что был достигнут край ленты, то машина переходит в состояние q''. *В противном случае, то есть если машина либо не пыталась сдвинуть головку на рабочей ленте либо пыталась и эта попытка была успешной, машина переходит в состояние* q'. *Далее:*

$$\text{Если } \mu_2 = \begin{cases} L, & \text{то } D \text{ сдвигает головку на входной ленте} \\ & \quad \text{на одну клетку влево;} \\ R, & \text{то } D \text{ сдвигает головку на входной ленте} \\ & \quad \text{на одну клетку вправо;} \\ N, & \text{то } D \text{ оставляет головку на входной ленте} \\ & \quad \text{неподвижной.} \end{cases}$$

Затем D *делает* i*-ю ленту неактивной, а* j*-ю ленту — активной.*

Наконец, если $\beta \in \Sigma$, *то* D *записывает* β *в вершину стека (тем самым стек углубляется на 1, если* $\beta \neq \wedge$, *а в случае* $\beta = \wedge$ *стек* *не изменяется). Если же $\beta = \dagger$ и стек не является пустым, то D выталкивает верхний элемент из стека (и укорачивает стек на 1).*

При таком варианте определения D называется $s-k$-DAPDA или просто s-DAPDA. Аналогично определяется s-NAPDA.

Целью данного параграфа является доказательство следующей теоремы, принадлежащей С.Куку.

Теорема 6.9. *Пусть s — некоторая арифметическая функция, асимптотически мажорирующая функцию $log(n)$ (то есть $log(n) \ll s$). Пусть Σ — алфавит, содержащий буквы 0, 1, $\wedge$ и, возможно, другие, $L \subseteq (\Sigma \setminus \{\wedge\})^*$, M — 1-МТ распознаватель, $L = L_M$ и $t_M \ll 2^{cs}$ для некоторой константы c.*

Тогда существует s-DAPDA D, что $L_D = L$.

Начнем с комментария. Теорема представляется удивительной, потому что устанавливает тонкую взаимосвязь между временной сложностью языка и "почти"ленточной сложностью. В действительности неизвестно, можно ли в формулировке теоремы 6.9 заменить s-DAPDA D на 1-МТ M' из класса $DSPACE(s)$. Это одна из самых интригующих проблем в общей теории сложности. Современная гипотеза: нет, нельзя.

Еще более удивительно, что верно и обращение теоремы 6.9 (собственно, обычно теорема Кука формулируется и доказывается в обе стороны). Мы приводим обращение теоремы Кука в качестве упражнения к настоящему параграфу, но фактически докажем это обращение в важном частном случае в следующей главе, по существу методами теории КС-языков.

Доказательство теоремы 6.9. Итак, пусть M — 1-МТ из теоремы, $M = \langle \Sigma, Q, \pi, q_0 \rangle$, $w \in (\Sigma \setminus \{\wedge\})^*$.

Рассмотрим работу M на w.

Конфигурацией M на w будем называть кортеж из 13-ти элементов:

$$\langle w_1, \alpha, w_2, v, m_1, m_2, l_1, l_2, \gamma, \beta, q, r, s \rangle,$$

где $w_1, w_2, v \in \Sigma^*$; $\alpha, \gamma, \beta \in \Sigma$; $m_1, m_2, l_1, l_2 \in \{0, 1, 2, \dots\}$; $q \in Q$; $r, s \in \{0, 1\}$.

Содержательно, конфигурация описывает "мгновенное состояние"M на каждом такте. Здесь w_1 — часть слова на входной ленте слева от головки, w_2 — часть слова на входной ленте справа от головки (без букв $\wedge$ в конце), α — обозреваемая буква на входной ленте, γ — просматриваемая на такте буква на рабочей ленте, β — записываемая на такте на рабочую ленту буква, v — слово на рабочей ленте до выполнения такта (без букв $\wedge$ в начале и в конце), q — состояние машины до выполнения такта.

Числа l_1 и l_2 характеризуют положение головки на рабочей ленте по отношению к той клетке, которая обозревалась в начальном состоянии, а именно:

если головка на рабочей ленте находится левее этой начальной клетки на l клеток, то $l_1 = l$, $l_2 = 0$;

если головка на рабочей ленте находится правее этой начальной клетки на l клеток, то $l_1 = 0$, $l_2 = l$;

если головка на рабочей ленте находится на той же начальной клетке, то $l_1 = 0$, $l_2 = 0$.

Числа m_1 и m_2 аналогичным образом характеризуют положение самой левой клетки на рабочей ленте, в которой записана буква, отличная от $\wedge$, а именно первая буква слова v, относительно клетки, которая обозревалась в начальный момент времени.

r становится равным 1, как только M записала 1 на выходную ленту, и потом всегда остается равным 1. До этого $r = 0$.

s становится равным 1, как только M останавливается. До этого $s = 0$.

Начальная конфигурация имеет вид:

$$\langle e, a, w', e, 0, 0, 0, 0, \wedge, \beta_0, q_0, 0, 0\rangle,$$

где $w = aw'$, e — пустое слово,

$$\pi(a, \wedge, q_0) = \langle \beta_0, q', \mu_1, \mu_2, c\rangle.$$

Фактически работу M на w можно описать последовательностью однозначно определяемых конфигураций.

Дадим теперь определение точки. Точка вычисления M на w — это 7-ка

$$\langle l_1, l_2, \gamma, \beta, q, r, s\rangle,$$

где $\gamma, \beta \in \Sigma$; $l_1, l_2 \in \{0, 1, 2, \dots\}$; $q \in Q$; $r, s \in \{0, 1\}$, которая является подкортежом конфигурации с теми же значениями элементов.

То есть γ — просматриваемая на такте буква на рабочей ленте, буква β записывается на такте на рабочую ленту, числа l_1 и l_2 характеризуют положение головки на рабочей ленте по отношению к той клетке, которая обозревалась в начальном состоянии, q — состояние машины до выполнения такта, r становится равным 1, как только M записала 1 на выходную ленту, и потом всегда остается равным 1. До этого $r = 0$. s становится равным 1, как только M останавливается. До этого $s = 0$.

Рассмотрим последовательность конфигураций $C_1, C_2, \ldots, C_n$, порождаемую при работе M на слове w, где C_1 — начальная конфигурация, и полученную проецированием этой последовательности конфигураций соответствующую последовательность точек вычисления $P_1, P_2, \ldots, P_n$.

Если C_n — заключительная конфигурация, то есть 13-й элемент C_n равен 1, то следующий элемент в этой последовательности конфигураций породить нельзя.

В противном случае можно породить C_{n+1} и, значит, P_{n+1}, получаемый его проекцией.

Зная машину M и C_n, можно явно выписать C_{n+1} и P_{n+1}.

Зададимся вопросом, какой информации не хватает в последовательности $P_1, P_2, \ldots, P_n$ для выписывания P_{n+1}?

Естественно, мы должны знать, какой символ встречается на входной ленте, чтобы вычислить π для конфигурации C_n (мы из P_n знаем символ, обозреваемый на рабочей ленте, и состояние машины). Зная этот символ, мы вычислим π и определим l'_1, l'_2, q', r', s', где

$$P_{n+1} = \langle l'_1, l'_2, \gamma', \beta', q', r', s' \rangle.$$

Остается определить γ', β'. Если, согласно вычисленному в конфигурации C_n значению n, головка на рабочей ленте остается неподвижной, то γ' совпадает с четвертым элементом P_n. Однако, если головка на рабочей ленте двигается вправо или влево, то мы должны, для определения γ', знать, какой символ находится в соответствующей клетке.

Выясняется, что эта последняя информация уже содержится в последовательности $P_1, P_2, \ldots, P_{n-1}$. Чтобы установить это, найдем такое максимальное m, что $1 \leq m \leq n-1$ и P_m имеет вид

$$\langle l'_1, l'_2, \gamma'', \beta'', q'', r'', s'' \rangle.$$

Ясно, что если такое m есть, то $\gamma' = \beta''$. Если же такого m не сыщется, то очевидно, что $\gamma' = \wedge$.

Теперь, зная обозреваемый в конфигурации C_{n+1} символ на входной ленте, состояние q' и символ γ', мы в состоянии вычислить вновь функцию π и определить тем самым символ β'.

Это рассуждение является ключевым в доказательстве теоремы 6.9.

Рассмотрим процесс порождения последовательности

$$P_1, P_2, P_3, \ldots,$$

при котором мы, порождая элементы последовательности, по одному погружаем их в стек. При этом мы "позволяем"себе смотреть, что делается на входной ленте машины, но "закрываем"от себя рабочую. Сначала в этом процессе мы запишем в вершину стека P_1. Далее будем порождать P_2, $P_3, \dots$ и записывать их в стек до тех пор, пока головка M на рабочей ленте стоит на месте. Тогда для порождения каждого элемента P_i нам нужен только элемент P_{i-1}, в силу приведенного выше рассуждения.

Предположим, что в какой-то момент времени головка M на рабочей ленте сдвигается. Допустим, что к этому моменту мы породили уже последовательность точек вычисления P_1, $P_2, \dots, P_m$ и записали их в стек.

Чтобы породить P_{m+1}, мы должны знать, какая буква $\gamma^{(m+1)}$ записана на рабочей ленте M в клетке, на которую M переходит. При этом, если

$$P_i = \langle l_1^{(i)}, l_2^{(i)}, \gamma^{(i)}, \beta^{(i)}, q^{(i)}, r^{(i)}, s^{(i)} \rangle$$

для $i = 1, 2, \dots, m+1$, то $l_1^{(m+1}, l_2^{(m+1)}$ легко вычисляются по $l_1^{(m)}, l_2^{(m)}$ и значению функции π, q^{m+1} содержится в значении π, $r^{(m+1)}$ и $s^{(m+1)}$ вычисляются по $r^{(m)}$, $s^{(m)}$ и по π.

Легко видеть, что $\gamma^{(m+1)}$ может быть извлечена из последовательности P_1, $P_2, \dots, P_m$, а именно, $\gamma^{(m+1)} = \beta^{(j)}$, где

$$j = \max{}_{1 \le k \le m} \{ k \mid l_1^{(k)} = l_1^{(m+1)}, l_2^{(k)} = l_2^{(m+1)} \},$$

если такое j существует, и $\gamma^{(m+1)} = \wedge$ в остальных случаях. Теперь $\beta^{(m+1)}$ может быть вычислена при помощи функции π, а именно $\beta^{(m+1)}$ равно первому элементу значения

$$\pi(\alpha, \gamma^{(m+1)}, q^{(m+1)}).$$

Однако мы не можем для порождения P_{m+1} взять искомый P_j прямо из стека, так как при этом, по правилам работы со стеком, мы потеряем в стеке все элементы $P_m, P_{m-1}, \dots, P_j$, а мы хотим иметь в стеке элементы P_1, $P_2, \dots, P_m$, P_{m+1} после выполнения данного шага.

Чтобы выйти из данной ситуации, мы вначале вновь породим в стеке все элементы P_1, $P_2, \dots, P_m$, а потом уже будем их уничтожать по одному и в процессе этого получим $\gamma^{(m+1)} = \beta^{(j)}$ (или же исчерпаем их все и, убедившись, что подходящего элемента нет, положим $\gamma^{(m+1)} = \wedge$). Итак, стек сначала будет преобразован к виду

$$P_1, P_2, \dots, P_{m-1}, P_m, P_1, P_2, \dots, P_{m-1}, P_m,$$

а потом уже к искомому виду

$$P_1, P_2, \dots, P_{m-1}, P_m, P_{m+1}.$$

Этот процесс мы будем называть *реконструкцией.*

При этом мы будем говорить, что

$$P_1, P_2, \dots, P_{m-1}, P_m$$

получены без реконструкций, а P_{m+1} — с одной реконструкцией.

Продолжив процесс порождения последовательности

$$P_1, P_2, \dots, P_{m-1}, P_m, P_{m+1}, \dots$$

дальше, мы, через некоторое время, вновь столкнемся с ситуацией, когда для получения некоторого P_{m+k}, где $k > 1$, нам нужен будет элемент P_{j_1}, для некоторого $1 \le j_1 \le m + k - 1$. Для получения этого элемента мы вновь преобразуем стек сначала к виду

$$P_1, P_2, \dots, P_{m+k-1}, P_1, P_2, \dots, P_{m+k-1},$$

а потом к виду

$$P_1, P_2, \dots, P_{m+k-1}, P_{m+k}.$$

При этом P_{m+k} будет получен с двумя реконструкциями: одна для получения P_m, а другая — для получения P_{m+k}.

И так далее ...

Неформально процесс порождения P_1, $P_2, \dots$ и записи их в стек описан.

Опишем процесс порождения более формально.

Четверку $\langle z, u, P, v\rangle$ будем называть *маркированной точкой*, если $u \in \{0, 1, \dots\}$, $v \in \{0, 1, 2, \dots, |w| + 1\}$, $z \in \{\infty, 0, 1, \dots\}$, P — точка вычисления M. Содержательно, u будет обозначать число реконструкций, с которым получена P, z — максимально допустимое число реконструкций (точный смысл поясним позже), а v — положение головки на входной ленте.

Рассмотрим абстрактную машину $\mathfrak{A}$, моделирующую работу M описанным выше неформально способом.

$\mathfrak{A}$ снабжена входной лентой, аналогичной ленте M, с головкой на ней, и стеком для запоминания маркированных точек для машины M. Дополнительно $\mathfrak{A}$ имеет память, достаточную для запоминания двух маркированных точек и функции π машины M.

Рассмотрим работу $\mathfrak{A}$ на входном слове w.

Шаг 1. $\mathfrak{A}$ делает маркированную точку

$$\langle \infty, 0, \langle 0, 0, \wedge, \beta, q_0, 0, 0\rangle, 1\rangle$$

текущей (то есть формирует такую маркированную точку и запоминает ее в месте в памяти, предназначенном для текущей точки). Здесь

$$\pi(a, \wedge, q_0) = \langle \beta, q'', \mu_1'', \mu_2'', c\rangle,$$

где a — первая буква слова w.

Шаг 2. Пусть

$$\langle i, j, \langle l_1, l_2, b, d, q, r, s\rangle, v\rangle$$

есть текущая маркированная точка. Запишем ее в стек.

Пусть a — v-я буква на входной ленте (в частности, если $v = 0$ или $v = |w| + 1$, то $a = \wedge$). Вычислим

$$\pi(a, b, q) = \langle d, q', \mu_1, \mu_2, c\rangle.$$

Шаг 3. Теперь наша цель — вычислить следующую текущую маркированную точку.

Вычислим v'. v' определяется в зависимости от v и от μ_2 :

если $\mu_2 = N$, то $v' = v$;

если $\mu_2 = L$, то

$$v' = \begin{cases} 0, \text{ если } v = 0, \\ v - 1 \text{ в остальных случаях;} \end{cases}$$

если $\mu_2 = R$, то

$$v' = \begin{cases} |w| + 1, \text{ если } v = |w| + 1, \\ v + 1 \text{ в остальных случаях.} \end{cases}$$

Шаг 4. Вычислим r'.

Если $c = 1$, то положим $r' = 1$, иначе положим $r' = r$.

Теперь, если $\mu_1 = S$ или $c = 0$, то перейдем к выполнению шага 10. Это соответствует завершению работы машины M.

Чтобы завершить вычисление очередной текущей точки, нам теперь осталось только определить новые координаты головки на рабочей ленте и букву, записанную под этими координатами.

Если $\mu_1 = L$ или $\mu_1 = R$, то перейдем к выполнению шага 6, в противном случае — к выполнению шага 5.

Шаг 5. Соответствует тому простому случаю, когда $\mu_1 = N$, то есть головка на рабочей ленте не сдвинулась.

Положим в качестве текущей маркированную точку

$$\langle i, j, \langle l_1, l_2, d, d', q', r', s \rangle, v' \rangle,$$

где

$$\pi(a', d, q') = \langle d', q'', \mu_1'', \mu_2'', c'' \rangle,$$

a' — v'-я буква слова w, и перейдем к шагу 2.

Шаг 6. Соответствует случаю, когда головка на входной ленте сдвинулась.

Положим

$$l_1' = \begin{cases} l_1 - 1, \text{ если } l_1 > 0, \text{ и } \mu_1 = R; \\ l_1 + 1, \text{ если } l_2 = 0, \text{ и } \mu_1 = L; \\ l_1 \text{ в остальных случаях.} \end{cases}$$

Положим

$$l_2' = \begin{cases} l_2 - 1, \text{ если } l_2 > 0, \text{ и } \mu_1 = L; \\ l_2 + 1, \text{ если } l_1 = 0, \text{ и } \mu_1 = R; \\ l_2 \text{ в остальных случаях.} \end{cases}$$

Таким образом, мы вычислили координаты новой клетки на рабочей ленте. Осталось найти, какая буква записана в клетке с этими координатами.

Если $i \neq j$, то перейдем к шагу 9 (где начинается очередная реконструкция).

Иначе будем вынимать из стека маркированные точки до тех пор, пока не окажется вынутой

а) либо маркированная точка вида

$$\langle i', j, \langle l_1'', l_2'', b'', d'', q'', r'', s'' \rangle, v'' \rangle$$

для некоторых $i' > i$ $(= j)$, l_1'', l_2'', b'', d'', q'', r'', s'', v'';

б) либо маркированная точка вида

$$\langle i, j', \langle l_1', l_2', b'', d'', q'', r'', s'' \rangle, v'' \rangle$$

для некоторых j', b'', d'', q'', r'', s'', v''.

а) соответствует случаю, когда при реконструкции не удалось найти нужной клетки на рабочей ленте. В случае а) идем на шаг 7.

б) соответствует случаю, когда при реконструкции удалось найти нужную клетку на рабочей ленте. В случае б) идем на шаг 8.

Шаг 7. Соответствует случаю, когда при реконструкции не удалось найти нужной клетки на рабочей ленте. Тогда искомая буква есть $\wedge$.

Итак, последней мы извлекли маркированную точку

$$\langle i', j, \langle l_1'', l_2'', b'', d'', q'', r'', s'' \rangle, v'' \rangle.$$

Из построения видно, что это та самая точка, которую мы загружали в стек при выполнении шага 9 в начале реконструкции, и что $b'' = d'' = \wedge$.

Сделаем текущей маркированную точку

$$\langle i', j+1, \langle l_1'', l_2'', \wedge, d''', q'', r'', s'' \rangle, v'' \rangle,$$

где

$$\pi(a'', \wedge, q'') = \langle d''', q''', \mu_1''', \mu_2''', c''' \rangle,$$

a'' — v''-я буква слова w, и перейдем к шагу 2.

Шаг 8. Соответствует случаю, когда при реконструкции удалось найти нужную клетку на рабочей ленте. Запомним d'' и будем вынимать из стека маркированные точки до тех пор, пока не вынем маркированную точку

$$\langle i', j, \langle l_1''', l_2''', b''', d''', q''', r''', s''' \rangle, v''' \rangle$$

для некоторых $i' > i$, l_1''', l_2''', b''', d''', q''', r''', s''', v'''.

Из построения видно, что это та самая точка, которую мы загружали в стек при выполнении шага 9 в начале реконструкции.

Сделаем текущей маркированную точку

$$\langle i', j+1, \langle l_1''', l_2''', d'', d_*, q''', r''', s''' \rangle, v''' \rangle,$$

где

$$\pi(a''', d'', q''') = \langle d_*, q_*, \mu_1'''', \mu_2'''', c_* \rangle,$$

a''' — v'''-я буква слова w, и перейдем к шагу 2.

Шаг 9. Отметим, что, раз мы пришли к данному шагу, то $j < i$.

Запишем в стек маркированную точку

$$\langle i, j, \langle l_1', l_2', \wedge, \wedge, q', r', s' \rangle, v' \rangle.$$

Фактически это была бы следующая текущая точка, если бы мы могли поставить нужные буквы вместо $\wedge$.

Сделаем текущей маркированную точку

$$\langle j, 0, \langle 0, 0, \wedge, \beta, q_0, 0, 0\rangle, 1\rangle,$$

где

$$\pi(a, \wedge, q_0) = \langle \beta, q'', \mu_1'', \mu_2'', c\rangle,$$

a — первая буква слова w, и перейдем к шагу 2.

Шаг 10. Если $r' = 1$, то остановиться. Иначе перейти к шагу 10. Таким образом, на этом шаге организуется вечный цикл, если только r' не было равно 1.

Конец описания.

Приведенное выше описание $\mathfrak{A}$ формально описывает процесс порождения точек вычисления P_1, P_2, $P_3, \ldots$

При этом $\mathfrak{A}$ останавливается на входном слове w тогда и только тогда, когда $w \in L_M$.

Для завершения доказательства теоремы 6.9 достаточно только показать, что описанная выше машина $\mathfrak{A}$ может быть промоделирована на подходящей DAPDA M'.

Для этого оценим максимальный размер маркированной точки при работе $\mathfrak{A}$ на слове w, $|w| = n$. Мы знаем, что M на слове w делает не более $2^{cs(n)}$ шагов при некоторой константе c (и при достаточно больших n, то есть почти для всех слов $w \in L_M$).

Мы знаем, что тогда и число клеток, которые M в состоянии посетить на входной и рабочей лентах, не превосходит для почти всех w как выше $2^{cs(n)}$.

В дальнейшем мы без ограничения общности будем считать, что приведенные выше ограничения времени работы верны для всех $w \in L_M$
.

Ясно, что всякая маркированная точка может быть закодирована словом длины $\le klog_2 2^{cs(n)}$, где k — некоторая константа. Но

$$klog_2 2^{cs(n)} = kcs(n) = k's(n).$$

Поясним это. Пусть

$$\langle i, j, \langle l_1, l_2, b, d, q, r, s\rangle, v\rangle$$

является маркированной точкой, где, напомним, i — допустимое число реконструкций, j — число реконструкций, с которым получена данная точка, l_1, l_2 — координаты головки на рабочей ленте, b — просматриваемая на такте буква на рабочей ленте, d — записываемая на такте на рабочую ленту буква, q — состояние машины M, r, s — соответственно флажки записи на выходную ленту и остановки машины M, v — координата головки на входной ленте .

Ясно, что b, d, q, r, s — элементы конечных алфавитов, и для их запоминания достаточно некоторого константного числа клеток.

В то же время i, j, l_1, l_2, v могут быть закодированы натуральными числами. При этом l_1, l_2 не превосходят числа просмотренных машиной M клеток на рабочей ленте и, следовательно, не превосходят числа $2^{cs(n)}$. Не превосходят числа $2^{cs(n)}$ и числа i, j, так как $j < t_M \leq 2^{cs(n)}$, $i \in \{\infty, 1.2, \ldots, t_M = 2^{cs(n)}\}$. Число же v принадлежит множеству $\{0, 1, 2, \ldots, n+1\}$. Таким образом, для кодирования каждого из этих чисел достаточно $log_2 2^{cs(n)} = cs(n)$ нулей и единиц.

Легко видеть, что запись машиной $\mathfrak{A}$ маркированной точки в стек может быть легко промоделирована записью ее посимвольно в стек DAPDA M' с разделителями между маркированными точками.

Значит, для моделирования $\mathfrak{A}$ можно построить $(2k' + m)$-DAPDA (k' лент требуется для хранения одной маркированной точки, нам же нужно, если читатель внимательно просмотрит алгоритм, работать одновременно с двумя маркированными точками: с одной текущей, другой для поиска в стеке; а также нам требуется фиксированное число лент — обозначенное выше через m — для манипулирования с числами и элементами конечных алфавитов).

Итак, мы показали, что можно построить такую s-DAPDA M', что $L_{M'} \bigcup \mathfrak{F} = L_M$, где $\mathfrak{F}$ — некоторый конечный язык (слов w, для которых еще не действует асимптотическая оценка $t^*_M(w) \leq 2^{cs(|w|)}$).

Очевидно, что M' можно преобразовать в такую s-DAPDA D, что $L_D = L_M$, ибо всякий конечный язык распознается конечным автоматом и D может сначала моделировать ДКА для $\mathfrak{F}$, а, в случае неудачи, затем моделировать DAPDA M'. ∎

Замечание 6.4.1. *Построенная в теореме 6.9 s-DAPDA D отличается тем свойством, что в процессе ее работы головка на входной ленте никогда не выходит за пределы входного слова более, чем на одну клетку.*

Упражнения

Упражнение 6.4.1. *Определение временной сложности можно естественным образом обобщить на DAPDA, то есть определить как число тактов при работе DAPDA на всех словах данной длины, на которых она останавливается. Покажите, что в условиях теоремы 6.9 су-*

ществует константа k'' такая, что временная сложность s-DAPDA D асимптотически мажорируется функцией $2^{k''s}$.

Упражнение 6.4.2 (*; сильное обращение теоремы Кука).**

Пусть D — некоторая s-NAPDA, $L = L_D$. Покажите, что существуют такие 1-МТ распознаватель M и константа c, что:

а) $L_M = L_D$;

б) $t_M \ll 2^{cs}$.

6.5 Сложность некоторых конкретных проблем из теории языков

В этом параграфе нас будет в основном интересовать необходимая для главы 7 теорема о временной сложности проблемы пустоты пересечения КС-языка и регулярного множества. В параграфе приведены точные определения, в то время как доказательства теорем сведены к изложению идей, а рутинная техническая часть, связанная с очевидным кодированием, опущена.

Пусть $J = \langle \Sigma, Q, \Gamma, q_0, F, P \rangle$ — МП-автомат (см. § 2.6). Без ограничения общности можно считать, что Σ, Q, Γ, $\{,\ ;\ \to\}$ попарно не пересекаются. Рассмотрим кодировку автомата J словами алфавита

$$\Sigma \bigcup Q \bigcup \Gamma \bigcup \{,\ ;\ \to\}.$$

Именно, пусть

$$\Sigma = \{a_1, a_2, \ldots, a_m\},$$

$$Q = \{q_0, q_1, q_2, \ldots, q_n\},$$

$$\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_k\},$$

$$F = \{q_{i_1}, q_{i_2}, \ldots, q_{i_l}\},$$

$$P_* = \{p_1, p_2, \ldots, p_r\},$$

где P_* — график отображения P, то есть для произвольных $u \in (\Sigma)^*$, $q \in Q$, $\delta \in (\Gamma)^*$, $v \in \{0, 1, \ldots\}$,

$$I = \{\langle q_{j_1}, \epsilon_{j_1}\rangle, \langle q_{j_2}, \epsilon_{j_2}\rangle, \ldots, \langle q_{j_v}, \epsilon_{j_v}\rangle\},$$

где

$$q_{j_1}, q_{j_2}, \ldots, q_{j_v} \in Q,$$

$$\epsilon_{j_1}, \epsilon_{j_2}, \ldots, \epsilon_{j_v} \in \Gamma^*,$$

имеет место
$$P(\langle u,q,\delta\rangle) = I \Leftrightarrow \langle\langle u,q,\delta\rangle, I\rangle \in P_*.$$

Тогда кодом J будем называть слово:

$$\#(J) = a_1a_2\dots a_m; q_0q_1q_2\dots q_n; \gamma_1\gamma_2\dots\gamma_k; q_0; q_{i_1}q_{i_2}\dots q_{i_l};$$

$$\#(p_1); \#(p_2); \dots; \#(p_r)$$

где
$$\#(\langle\langle u,q,\delta\rangle, \{\langle q_{j_1},\epsilon_{j_1}\rangle, \langle q_{j_2},\epsilon_{j_2}\rangle, \dots, \langle q_{j_v},\epsilon_{j_v}\rangle\}\rangle) =$$
$$uq\delta \to q_{j_1}\epsilon_{j_1}, q_{j_2}\epsilon_{j_2}, \dots, q_{j_v}\epsilon_{j_v}$$

В аналогичном стиле проведем кодировку конечных автоматов.

Пусть $J = \langle\Sigma, Q, q_0, F, P\rangle$ — конечный автомат (см. § 1.4). Без ограничения общности можно считать, что Σ, Q, $\{,\ ;\ \to\}$ попарно не пересекаются. Рассмотрим кодировку автомата J словами алфавита

$$\Sigma \bigcup Q \bigcup \{,\ ;\ \to\}.$$

Именно, пусть
$$\Sigma = \{a_1, a_2, \dots, a_m\},$$
$$Q = \{q_0, q_1, q_2, \dots, q_n\},$$
$$F = \{q_{i_1}, q_{i_2}, \dots, q_{i_l}\},$$
$$P = \{\langle q_{j_1}, a_{k_1}, q_{r_1}\rangle, \langle q_{j_2}, a_{k_2}, q_{r_2}\rangle, \dots, \langle q_{j_h}, a_{k_h}, q_{r_h}\rangle\},$$

где, напомним, P — конечное множество правил автомата J. Тогда кодом J будем называть слово:

$$\#(J) = a_1a_2\dots a_m; q_0q_1q_2\dots q_n; q_{j_1}a_{k_1}q_{r_1}, q_{j_2}a_{k_2}q_{r_2}, \dots, q_{j_h}a_{k_h}q_{r_h}$$

Фактически символ $\to$ в кодах конечных автоматов не встречается.

Подобным же образом будут кодироваться КС-грамматики. Так, пусть $G = \langle\Sigma, Q, S_0, P\rangle$ — КС-грамматика (см. § 2.1). Без ограничения общности можно считать, что Σ, Q, $\{,\ ;\ \to\}$ попарно не пересекаются. Рассмотрим кодировку грамматики G словами алфавита

$$\Sigma \bigcup Q \bigcup \{,\ ;\ \to\}.$$

Именно, пусть
$$\Sigma = \{a_1, a_2, \dots, a_m\},$$

$$Q = \{S_0, S_1, S_2, \dots, S_n\},$$

$$P = \{S_{k_1} \to u_{k_1}, S_{k_2} \to u_{k_2}, \dots, S_{k_r} \to u_{k_r}\},$$

где $u_{k_1}, u_{k_2}, \dots, u_{k_r} \in (\Sigma \bigcup Q)^*$.

Тогда кодом G будем называть слово:

$$\#(G) = a_1 a_2 \dots a_m; S_0 S_1 S_2 \dots S_n; S_0; S_{k_1} \to u_{k_1}, S_{k_2} \to u_{k_2}, \dots, S_{k_r} \to u_{k_r}$$

Лемма 6.10. *Пусть Σ, Q, Γ — соответственно входной, нетерминальный и магазинный алфавиты. Пусть*

$$L_+ \subseteq (\Sigma \bigcup Q \bigcup \Gamma \bigcup \{,\ ;\ \to\})^*,$$

$$w \in L_+ \Leftrightarrow w = \#(J)$$

для некоторого такого МП-автомата $J = \langle \Sigma, Q, \Gamma, q_0, F, P \rangle$, что $L(J) \neq \emptyset$.

Тогда $L_+ \in \mathcal{P}$.

Доказательство. Для доказательства необходимо построить МТ распознаватель, распознающий L_+, и оценить время его работы.

Такое построение рутинно и состоит из четырех этапов.

1 этап. В соответствии с методом леммы 2.33, по автомату J строится эквивалентный ему полуканонический МП-автомат J' (точнее, по коду $\#(J)$ строится код $\#(J')$).

2 этап. В соответствии с методом теоремы 2.34, по J' (точнее, по коду $\#(J')$) строится эквивалентная J' КС-грамматика G (точнее, код КС-грамматики $\#(G)$).

3 этап. В соответствии с методом теоремы 2.10, можно G эффективно преобразовать к приведенной КС-грамматике G' (то есть по коду $\#(G)$ построить код $\#(G')$).

4 этап. Ясно, что

$$L(G') = \emptyset \Leftrightarrow L(J) = \emptyset.$$

Но $L(G') = \emptyset \Leftrightarrow G'$ имеет непустое множество правил. Это можно легко проверить на кодах.

Все указанные выше преобразования могут быть проведены на детерминированной машине Тьюринга, работающей с кодами. Это предоставляется доказать читателю в качестве упражнения. Полиномиальность каждого шага очевидна.

Кроме того, за полиномиальное время можно проверить, до начала работы, что на входной ленте записан именно код некоторого МП-автомата.

Построенная детерминированная машина Тьюринга является, при прямом построении, многоленточной, но может быть, согласно теореме 6.2, промоделирована 1-МТ с сохранением полиномиальности времени работы. ∎

Теорема 6.11. *Пусть Σ, Q, Γ — соответственно входной, нетерминальный и магазинный алфавиты. Пусть*

$$L_{++} \subseteq (\Sigma \bigcup Q \bigcup \Gamma \bigcup \{,\ ;\ \rightarrow\})^*,$$

$$w \in L_{++} \Leftrightarrow w = u;;v$$

для кода $u = \#(J)$ некоторого такого МП-автомата

$$J = \langle \Sigma, Q, \Gamma, q_0, F, P \rangle$$

и кода $v = \#(J')$ некоторого такого конечного автомата

$$J' = \langle \Sigma, Q', q_0', F', P' \rangle,$$

что $L(J) \bigcap L(J') \neq \emptyset$.

Тогда $L_{++} \in \mathcal{P}$.

Доказательство. Доказательство состоит в построении 1-МТ распознавателя, который распознает L_{++}. Он работает в два этапа.

1 этап. Строит по $\#(J)$ и $\#(J')$ код $\#(J'')$ МП-автомата J'', распознающего в точности язык $L(J) \bigcap L(J')$ (очевидно, построение кода такого автомата может быть проведено в полиномиальное время так, как указано в доказательстве теоремы 2.41).

2 этап. Использует метод леммы 6.10.

Как и в лемме, необходимо сначала проверить, что входное слово имеет вид $\#(J);;\#(J')$, то есть что оно представляет собой в точности разделенные коды МП-автомата и конечного автомата. ∎

Упражнения

Упражнение 6.5.1. *Проведите формальное доказательство леммы 6.10 и теоремы 6.11.*

Упражнение 6.5.2 (**). *Пусть L — КС-язык.*

Покажите, что $L \in \mathcal{P}$.

Упражнение 6.5.3. *Пусть L — регулярный язык.*
Покажите, что $L \in DLINEARTIME$.

Упражнение 6.5.4 (*; теорема Майера).** *В связи с теоремами настоящего параграфа интересен следующий результат Альберта Майера. Мы знаем, что регулярные языки замкнуты относительно пересечения. Рассмотрим 1-МТ M, которая по записи двух регулярных выражений r и r' выдает запись регулярного выражения r'' такого, что*

$$L(r'') = L(r) \bigcap L(r').$$

Существование такой машины очевидно. Покажите, что 1-МТ M работает экспоненциальное время, в частности, что $M \notin \mathcal{P}$.

7 ПОЛИНОМИАЛЬНО ВЫЧИСЛИМЫЕ ГЛОБАЛЬНЫЕ ПРЕДИКАТЫ

7.1 Реляционные базы данных. Языки запросов. Глобальные предикаты

Под базой данных понимают программную систему, обеспечивающую ввод, долговременное хранение и поиск данных. С течением времени решение задач такого рода становится одной из основных областей применения компьютеров.

Один из путей решения таких задач — написание для каждой конкретной задачи своей собственной программы обработки. Таким образом и до сих пор устроены многие системы, например, некоторые системы резервирования авиабилетов и т.п. Преимуществом такого пути является высокая эффективность программной системы.

Однако этот подход имеет ряд недостатков. Главный из них — неуниверсальность. Дело в том, что при этом подходе малейшие изменения спецификаций системы или ее данных приводят к необходимости перепрограммирования ее частей.

Рассмотрим, к примеру, гипотетическую кадровую систему, в которой хранятся сведения о работниках некоторого предприятия. Невозможно предвидеть заранее, вопросы какого рода будут задаваться системе пользователем. Писать для каждого запроса программу на универсальном языке программирования нельзя: это времяемкое занятие, к тому же требующее квалифицированного труда, в то время как пользователь такой системы — это менеджер.

В результате осмысления этих проблем были предложены различные концепции баз данных, из которых все более популярной становится концепция реляционных баз данных, предложенная Е.Ф.Коддом. Ниже мы будем иметь дело с различными модификациями этой концепции.

В реляционных базах данных предполагается, что данные структурированы в виде двумерных таблиц, каждая с фиксированным числом столбцов и неизвестным заранее числом строк. Скажем, в случае простой кадровой системы таблица может выглядеть так:

Ф.И.О.	Год рожд.	Специальность	Год пост.

Строками данной таблицы будут записи о конкретных работниках, например

Ф.И.О.	Год рожд.	Специальность	Год пост.
Иванов И.И.	1950	рабочий	1985
Петров С.С.	1940	бухгалтер	1985
Сидорова С.Ю.	1945	директор	1980

В целом реляционная база данных может содержать не одну, а несколько различных таблиц, описание структуры которых (т.е. числа столбцов и заголовков) называется схемой базы данных.

Состоянием базы данных (с некоторой схемой) называется некоторое конкретное заполнение строк ее таблиц.

При этом ввод в эту базу данных мыслится как добавление некоторой строки в некоторую таблицу, а поиск — как ответ на некоторый вопрос о содержимом таблиц.

Для формулирования запросов разрабатываются специальные языки запросов, о которых будет идти речь ниже.

В наших рассмотрениях мы без потери общности будем пренебрегать разносортностью столбцов таблиц, то есть считать, что во всех столбцах могут быть записаны одни и те же данные. Другими словами, мы оставляем в схеме базы данных только число столбцов у каждой из таблиц, и пренебрегаем их заголовками.

В этом случае очевидно, что фактически схема базы данных представляет собой некоторую реляционную (то есть состоящую только из символов отношений) конечную сигнатуру, в которой каждой таблице с n столбцами поставлен в соответствие предикатный n-местный символ.

Состоянию же базы данных соответствует некоторая конечная алгебраическая система этой сигнатуры.

Итак в дальнейшем под *схемой реляционной базы данных* мы будем понимать реляционную сигнатуру, а под *состоянием этой базы данных* будем понимать конечную алгебраическую систему этой сигнатуры.

Так, в описанном выше примере упрощенной кадровой системы сигнатура будет состоять из единственного 4-х местного предикатного символа, скажем, R, а описанному состоянию:

Ф.И.О.	Год рожд.	Специальность	Год пост.
Иванов И.И.	1950	рабочий	1985
Петров С.С.	1940	бухгалтер	1985
Сидорова С.Ю.	1945	директор	1980

будет соответствовать алгебраическая система A с основным множеством

$$|A| = \{\text{Иванов И.И., Петров С.С., Сидорова С.Ю.,}$$
$$1940, 1945, 1950, 1980, 1985,$$
$$\text{рабочий, бухгалтер, директор}\}$$

и предикатом

$$R^A = \{\langle \text{Иванов И.И.}, 1950, \text{ рабочий }, 1985\rangle,$$
$$\langle \text{Петров С.С.}, 1940, \text{ бухгалтер }, 1985\rangle,$$
$$\langle \text{Сидорова С.Ю.}, 1945, \text{ директор }, 1980\rangle\}.$$

В реляционных базах данных ответ на запрос пользователя также представляет собой таблицу (отношение), которая является производной от состояния базы данных.

Сам Кодд в качестве языка запросов предложил использовать так называемую реляционную алгебру. В реляционной алгебре имеется фиксированный конечный набор операций разной местности, определенных на таблицах и дающих таблицу в качестве своего результата. Для некоторых операций требуется также согласование аргументов. Например, в операции объединения двух таблиц требуется, чтобы таблицы имели одинаковое число столбцов.

Таким образом, чтобы сформулировать запрос, пользователь должен лишь указать выражение, составленное из таблиц базы данных и операций реляционной алгебры, и в качестве ответа получить результирующую таблицу.

Оказалось, что по существу эта реляционная алгебра эквивалентна языку логики предикатов (в смысле определяемых глобальных предикатов, см. ниже), поэтому мы не будем подробно останавливаться собственно на реляционной алгебре.

Итак, пусть Ω — реляционная сигнатура, L_Ω — язык логики предикатов в сигнатуре Ω, $\varphi \in L_\Omega$ — формула этого языка, а список x_1, $x_2, \ldots, x_n$ задает все свободные переменные формулы φ (мы предполагаем, что множество всех переменных языка L_Ω линейно упорядочено и перечисляем переменные в этом порядке). Всякая такая формула φ называется *запросом* (к базе данных со схемой Ω).

Пусть теперь $\mathcal{A}$ — некоторая конечная алгебраическая система сигнатуры Ω, $|\mathcal{A}|$ — ее основное множество (носитель).

Напомним, что оценкой в системе $\mathcal{A}$ или состоянием системы $\mathcal{A}$ называется произвольное отображение σ множества переменных в носитель системы $\mathcal{A}$.

Ответом на запрос φ при состоянии σ системы $\mathcal{A}$ называется n-местный предикат $\varphi_\mathcal{A}$, определенный так:

$$\langle a_1, a_2, \ldots, a_n \rangle \in \varphi_\mathcal{A} \Leftrightarrow \sigma_{a_1, a_2, \ldots, a_n} \models \varphi$$

для произвольных $a_1, a_2, \ldots, a_n \in |\mathcal{A}|$, где

$$\sigma_{a_1, a_2, \ldots, a_n}(x) = \begin{cases} a_i, \text{ если } x \text{ есть } x_i \text{ и } 0 < i \leq n; \\ a_1 \text{ в остальных случаях.} \end{cases}$$

Другими словами, предикат $\varphi_\mathcal{A}$ определяется как множество точек истинности формулы φ в системе $\mathcal{A}$.

Ясно, что формула φ определяет предикат во всякой конечной системе сигнатуры Ω, то есть глобальный предикат.

Определение 7.1.1 (глобального предиката). *Пусть Ω — произвольная сигнатура, $\mathfrak{K}$ — некоторый класс ее конечных систем.*

$\mathfrak{K}$-глобальным n-местным предикатом называется отображение, которое каждой системе $\mathcal{A} \in \mathfrak{K}$ ставит в соответствие некоторый n-местный предикат в системе $\mathcal{A}$. Мы будем опускать $\mathfrak{K}$, когда ясно, о каком классе систем идет речь.

По существу языки запросов представляют собой языки описания глобальных предикатов. Ясно в частности, что каждая формула φ языка логики предикатов определяет глобальный предикат в классе всех конечных систем соответствующей сигнатуры,

$$\mathcal{A} \longmapsto \varphi_\mathcal{A}.$$

Некоторое время язык логики предикатов был в центре изучения как язык запросов (то есть как язык описания глобальных предикатов).

Однако со временем в качестве языков запросов стали рассматриваться и другие логические языки. Это делалось как из практических, так и из теоретических соображений.

Нас, безусловно, интересуют только вычислимые глобальные предикаты. Неформально, глобальный предикат Q называется вычислимым, если существует 1-МТ распознаватель такой, что, получив на входной ленте код произвольной конечной системы $\mathcal{A}$ сигнатуры Ω и код n-ки

$$\langle a_1, a_2, \ldots, a_n \rangle \in |\mathcal{A}|,$$

M принимает вход тогда и только тогда, когда

$$\langle a_1, a_2, \ldots, a_n \rangle \in Q(\mathcal{A}).$$

Данное определение станет формальным, как только мы определим, что такое код конечной системы и код n-ки ее элементов.

Просто устанавливается, что всякая формула φ логики предикатов определяет вычислимый глобальный предикат.

Однако есть довольно простые вычислимые глобальные предикаты, не задаваемые такими формулами.

Упражнения

Упражнение 7.1.1 (*). *Формализуйте некоторым образом определение вычислимого глобального предиката и покажите, что:*

а) всякая формула логики предикатов определяет вычислимый глобальный предикат;

б) существует вычислимый глобальный предикат, который не задается никакой формулой логики предикатов.

Упражнение 7.1.2 (*). *Пусть сигнатура Ω содержит двухместный предикатный символ R и $\mathcal{A}$ — алгебраическая система сигнатуры Ω. Таким образом, в $\mathcal{A}$ определен двухместный предикат $R^{\mathcal{A}}$. Определим в $\mathcal{A}$ двухместные предикаты $R_i^{\mathcal{A}}$ для $i = 1, 2, \ldots$ индукцией по i так:*

$R_1^{\mathcal{A}} = R^{\mathcal{A}}$;

$$\langle a, b \rangle \in R_{i+1}^{\mathcal{A}} \Leftrightarrow (\langle a, b \rangle \in R_i^{\mathcal{A}} \vee (\exists c \in |\mathcal{A}|)(\langle a, c \rangle \in R_i^{\mathcal{A}} \& \langle c, b \rangle \in R^{\mathcal{A}}))$$

для любых $a, b \in |\mathcal{A}|$. Наконец, транзитивным замыканием предиката $R^{\mathcal{A}}$ будем называть предикат

$$R_*^{\mathcal{A}} = \bigcup_{i=1}^{\infty} R_i^{\mathcal{A}}.$$

Соответственно, транзитивным замыканием глобального предиката

$$\mathcal{A} \longmapsto R^{\mathcal{A}}$$

будем называть глобальный предикат

$$\mathcal{A} \longmapsto R_*^{\mathcal{A}}.$$

Покажите, что глобальный предикат

$$\mathcal{A} \longmapsto R_*^{\mathcal{A}}$$

вычислим, но не задается никакой формулой логики предикатов.

В то же время для всякого $n = 1, 2, \ldots$ глобальный предикат

$$\mathcal{A} \longmapsto R_n^{\mathcal{A}}$$

задается некоторой формулой логики предикатов.

Выпишите эту формулу.

7.2 Алгоритмическое описание глобальных предикатов

В этом параграфе мы рассмотрим алгоритмические способы задания глобальных предикатов в алгебраических системах произвольных конечных сигнатур (содержащих, возможно, некоторые функциональные символы).

В дальнейшем пусть фиксирована конечная сигнатура Ω и список переменных $x_1, x_2, \ldots$, а, тем самым, язык L_Ω логики предикатов в сигнатуре Ω. Рассмотрим также некоторую алгебраическую систему $\mathcal{A}$ сигнатуры Ω.

Определение 7.2.1 (присваивания). *Присваиванием называется конструкция вида*

$x_i := t$,

где x_i — переменная, а t — терм языка L_Ω. Присваивание $x_i := t$ определяет отображение множества оценок в системе $\mathcal{A}$ в себя следующим образом. Пусть $\sigma : \{x_1, x_2, \ldots\} \to |\mathcal{A}|$ — некоторая оценка. Тогда $(x_i := t)(\sigma)$ — оценка, определенная так:

$$((x_i := t)(\sigma))(x_j) = \begin{cases} \sigma(x_j), \text{ если } j \neq i; \\ \sigma(t), \text{ если } j = i, \end{cases}$$

где $\sigma(t)$ — значение терма t при оценке σ.

Определение 7.2.2 (простой программы). *Простой программой (в сигнатуре Ω) будем называть конечную последовательность пронумерованных числами от 1 до n n-инструкций, где n — число элементов последовательности.*

n-инструкцией называется $\langle I, m\rangle$, где m — натуральное число, $1 \leq m \leq n$, а I — одна из следующих конструкций:

а) присваивание $x_i := t$;

б) переход $if\ \varphi\ goto\ i$; здесь $1 \leq i \leq n$, φ — формула языка L_Ω;

в) останов $stop$.

Дадим сначала неформальные пояснения. Простая программа — это некоторый способ определения вычислений в произвольной алгебраической системе. По структуре простые программы напоминают программы на очень простом языке программирования типа Бэйсика. Натуральные числа, которыми пронумерованы инструкции, — это фактически метки инструкций. Число, являющееся вторым элементом инструкции, — это номер инструкции, которую следует выполнять после данной. Смысл присваивания понятен. *stop* завершает работу программы. Переход

$$if\ \ \varphi\ \ goto\ \ i$$

позволяет изменить порядок выполнения инструкций, перейдя к инструкции с номером i при условии истинности формулы φ. Фактически это соответствует условному переходу.

Для повышения наглядности мы будем при записи программы записывать каждую ее инструкцию в отдельной строке, слева писать номер инструкции, затем, через двоеточие, саму инструкцию. Например:

$1:\ \ \langle x_1 := x_2 + x_3,\ \ 2\rangle$

$2:\ \ \langle stop,\ \ 2\rangle$

является программой в сигнатуре, содержащей двухместный функциональный символ $+$.

Формально же, простая программа определяет частичное отображение множества всех оценок в данной алгебраической системе в себя. Дадим рекурсивное определение этого отображения.

Определение 7.2.3. *Пусть $P = \langle \alpha_1, \alpha_2, \ldots, \alpha_n\rangle$ — простая программа, где α_1, $\alpha_2, \ldots, \alpha_n$ — n-инструкции, Σ — множество всех оценок в системе $\mathcal{A}$, то есть отображений*

$$\sigma : \{x_1, x_2, \ldots\} \to |\mathcal{A}|.$$

Определим для всякого $i \in \{1, 2, \ldots, n\}$ частичное отображение $P_i^{\mathcal{A}} : \Sigma \to \Sigma$ так. Пусть $\alpha_i = \langle I_i, m_i\rangle$, $\sigma : \{x_1, x_2, \ldots\} \to |\mathcal{A}|$ — некоторая *оценка из* Σ. *Тогда*

$$P_i^{\mathcal{A}}(\sigma) = \begin{cases} P_{m_i}(I_i(\sigma)), \text{ если } I_i \text{ является присваиванием } x_k := t; \\ P_j(\sigma), \text{ если } I_i \text{ есть переход } if\ \varphi\ goto\ j \text{ и } \sigma \models \varphi; \\ P_{m_i}(\sigma), \text{ если } I_i \text{ есть переход } if\ \varphi\ goto\ j \text{ и } \sigma \models \neg\varphi; \\ \sigma, \text{ если } I_i \text{ есть } stop. \end{cases}$$

Таким образом, рекурсия в определении $P_i^{\mathcal{A}}(\sigma)$ *завершается только в том случае, если при вычислении мы выходим на stop.*

Определим теперь частичное отображение $P : \Sigma \to \Sigma$:

$$P^{\mathcal{A}}(\sigma) = P_1^{\mathcal{A}}(\sigma).$$

Покажем теперь, как использовать простые программы для определения глобальных предикатов.

Определение 7.2.4. *Пусть* $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ *— некоторый список переменных;* m *— натуральное число;* $0 < m \le k$; $\mathcal{A}$ *— некоторая конечная алгебраическая система сигнатуры* Ω; $a_1, a_2, \ldots, a_m \in |\mathcal{A}|$; σ *— некоторая оценка в системе* $\mathcal{A}$.

$(a_1, a_2, \ldots, a_m)$ *-ограничением оценки* σ *на переменные*

$$x_{i_1}, x_{i_2}, \ldots, x_{i_k}$$

будем называть оценку

$$\sigma[\frac{x_{i_1}, x_{i_2}, \ldots, x_{i_k}}{a_1, a_2, \ldots, a_m}] : \{x_1, x_2, \ldots\} \to |\mathcal{A}|,$$

определенную так:

$$(\sigma[\frac{x_{i_1}, x_{i_2}, \ldots, x_{i_k}}{a_1, a_2, \ldots, a_m}])(x_j) = \begin{cases} a_q, \text{ если } j = i_q, q \le m; \\ a_1 \text{ в остальных случаях.} \end{cases}$$

Определение 7.2.5. *Пусть* $P = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ *— простая программа в сигнатуре* Ω, $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ *— все использованные в программе* P *переменные, упорядоченные в порядке возрастания их номеров (переменные в простой программе могут быть использованы в правых и левых частях присваиваний и в формулах, стоящих в переходах).*

Пусть $0 < m \le k$ *— натуральное число,* $\mathcal{A}$ *— некоторая конечная алгебраическая система сигнатуры* Ω. *Определим* m*-местный предикат* $P^{\mathcal{A},m}$ *в системе* $\mathcal{A}$ *так.*

Для $a_1, a_2, \ldots, a_m \in |\mathcal{A}|$

$$\langle a_1, a_2, \ldots, a_m \rangle \in P^{\mathcal{A},m} \Leftrightarrow P^{\mathcal{A}}(\sigma[\frac{x_{i_1}, x_{i_2}, \ldots, x_{i_k}}{a_1, a_2, \ldots, a_m}]) \text{ определена.}$$

Теперь, с помощью программы P *и натурального числа* m *как выше, мы в состоянии определить глобальный предикат* $P[m] : \mathcal{A} \longmapsto P^{\mathcal{A},m}$.

Поясним определение примером. В упражнении 7.1.2 дается определение транзитивного замыкания двухместного предиката и предлагается доказать, что транзитивное замыкание глобального двухместного предиката не задается никакой формулой логики предикатов. Рассмотрим сигнатуру $\Omega = \langle f^{(1)}, R^{(2)} \rangle$, где f — одноместный функциональный символ, R — двухместный предикатный символ. Рассмотрим теперь класс $\mathfrak{A}$ таких конечных алгебраических систем сигнатуры Ω, что в каждой системе $\mathcal{A}$ класса $\mathfrak{A}$ для произвольных таких $a, b \in |\mathcal{A}|$, что $a \neq b$, существует такое натуральное n, что $(f^{\mathcal{A}})^n(a) = b$, и предикат $R^{\mathcal{A}}$ "однозначен", то есть для произвольных $a, b, c \in |\mathcal{A}|$, если $\langle a, b \rangle \in R^{\mathcal{A}}$ и $\langle a, c \rangle \in R^{\mathcal{A}}$, то $b = c$.

Известно, что $\mathfrak{A}$-глобальное транзитивное замыкание R, то есть $\mathfrak{A}$-глобальный предикат $\mathcal{A} \longmapsto R_*^{\mathcal{A}}$, не задается никакой формулой. Мы не будем здесь доказывать этот факт, а вместо этого зададим этот $\mathfrak{A}$-глобальный предикат при помощи простой программы.

Итак, рассмотрим следующую простую программу P :
1 : $\langle if \;\; R(x_1, x_2) \;\; goto \;\; 6, \;\; 2 \rangle$
2 : $\langle x_3 := x_1, \;\; 3 \rangle$
3 : $\langle x_3 := f(x_3), \;\; 4 \rangle$
4 : $\langle if \;\; R(x_1, x_3) \;\; goto \;\; 5, \;\; 3 \rangle$
5 : $\langle x_1 := x_3, \;\; 1 \rangle$
6 : $\langle stop, \;\; 6 \rangle$

Легко видеть, что предикаты $P^{\mathcal{A},2}$ и $R_*^{\mathcal{A}}$ совпадают во всякой системе $\mathcal{A}$ класса $\mathfrak{A}$, то есть что $\mathfrak{A}$-глобальный предикат $P[2] : \mathcal{A} \longmapsto P^{\mathcal{A},2}$ совпадает с искомым глобальным предикатом $\mathcal{A} \longmapsto R_*^{\mathcal{A}}$.

Замечание 7.2.1. *Пусть фиксирована произвольная сигнатура* Ω. *Для произвольной формулы* φ *с* n *переменными найдется такая простая программа* P, *что для произвольного класса* $\mathfrak{A}$ *конечных алгебраических систем сигнатуры* Ω, *глобальные предикаты*

$$\varphi : \mathcal{A} \longmapsto \varphi^{\mathcal{A}},$$

$$P[n] : \mathcal{A} \longmapsto P^{\mathcal{A},n}$$

совпадают.

Доказательство. Очевидно. Достаточно рассмотреть в качестве P программу:

1 : $\langle if \ \ \varphi \ \ goto \ \ 2, \ \ 1\rangle$
2 : $\langle stop, \ \ 2\rangle$ ∎

Мы будем рассматривать, наряду с простыми программами, также еще один класс программ, а именно рекурсивные программы. Они отличаются от простых добавлением всего одной новой инструкции *call* i. Дадим полное определение.

Определение 7.2.6 (рекурсивной программы).

Рекурсивной программой (в сигнатуре Ω*) будем называть конечную последовательность пронумерованных числами от* 1 *до* n *рекурсивных* n*-инструкций, где* n *— число элементов последовательности.*

Рекурсивной n*-инструкцией называется* $\langle I, m\rangle$*, где* m *— натуральное число,* $1 \le m \le n$*, а* I *— одна из следующих конструкций:*

а) присваивание $x_i := t$*;*
б) переход $if \ \varphi \ goto \ i$*; здесь* $1 \le i \le n$*,* φ *— формула языка* L_Ω;
в) останов stop;
г) вызов call i*; здесь* $1 \le i \le n$*.*

Опять начнем с неформальных пояснений. Рекурсивная программа отличается от простой только одной новой инструкцией *call*. Эта инструкция по смыслу близка к вызову подпрограммы в языке Бэйсик. То есть при выполнении инструкции *call* i происходит переход к выполнению i-ой инструкции (как и в случае перехода), однако с последующим возвратом после завершения "подпрограммы". Вызванная подпрограмма работает на тех же данных, что и вызвавшая.

Формально же, рекурсивная программа тоже определяет частичное отображение множества всех оценок в данной алгебраической системе в себя. Дадим рекурсивное определение этого отображения.

Определение 7.2.7. *Пусть* $P = \langle \alpha_1, \alpha_2, \dots, \alpha_n\rangle$ *— рекурсивная программа, где* $\alpha_1, \alpha_2, \dots, \alpha_n$ *—* n*-инструкции,* Σ *— множество всех оценок в системе* $\mathcal{A}$*, то есть отображений*

$$\sigma : \{x_1, x_2, \dots\} \to |\mathcal{A}|.$$

Определим для всякого $i \in \{1, 2, \dots, n\}$ *частичное отображение* $P_i^{\mathcal{A}} : \Sigma \to \Sigma$ *так. Пусть* $\alpha_i = \langle I_i, m_i\rangle$*,* $\sigma : \{x_1, x_2, \dots\} \to |\mathcal{A}|$ *— некоторая* *оценка из* Σ. *Тогда*

$$P_i^{\mathcal{A}}(\sigma) = \begin{cases} P_{m_i}(I_i(\sigma)), \text{ если } I_i \text{ является присваиванием } x_k := t; \\ P_j(\sigma), \text{ если } I_i \text{ есть переход } if\ \varphi\ goto\ j \text{ и } \sigma \models \varphi; \\ P_{m_i}(\sigma), \text{ если } I_i \text{ есть переход } if\ \varphi\ goto\ j \text{ и } \sigma \models \neg\varphi; \\ P_{m_i}(P_j(\sigma)), \text{ если } I_i \text{ есть вызов } call\ j; \\ \sigma, \text{ если } I_i \text{ есть } stop. \end{cases}$$

Таким образом, в отличие от простых программ, рекурсия в определении $P_i^{\mathcal{A}}(\sigma)$ *не всегда завершается в случае, если при вычислении мы выходим на stop. Рекурсия завершается, если мы пройдем stop на единицу больше раз, чем call.*

Определим теперь частичное отображение $P^{\mathcal{A}} : \Sigma \to \Sigma$ *так:*
$P^{\mathcal{A}}(\sigma) = P_1^{\mathcal{A}}(\sigma)$.

Для рекурсивной программы P, предикаты $P^{\mathcal{A},m}$ и глобальный предикат $P[m]$ определяются в точности по определению 7.2.5 выше.

Замечание 7.2.2. *Пусть фиксирована произвольная сигнатура* Ω. *Для произвольной простой программы* P *в сигнатуре* Ω *найдется такая рекурсивная программа* Q, *что для произвольного класса* $\mathfrak{A}$ *конечных алгебраических систем сигнатуры* Ω *и для произвольного положительного* n, *глобальные предикаты*

$$P[n] : \mathcal{A} \longmapsto P^{\mathcal{A},n},$$

$$Q[n] : \mathcal{A} \longmapsto Q^{\mathcal{A},n}$$

совпадают.

Доказательство. Очевидно из определения. ∎

Изучением свойств рекурсивных и простых программ с точки зрения определяемых с их помощью глобальных предикатов, а также значением этих формализмов для теории реляционных баз данных, мы займемся в следующих параграфах.

Упражнения

Упражнение 7.2.1 (*). *Докажите формально, что существует такие сигнатура* Ω *и простая программа* P *в этой сигнатуре, что глобальный для класса всех конечных систем сигнатуры* Ω *предикат* $P[2]$

не задается никакой формулой сигнатуры Ω. Например, покажите, что транзитивное замыкание не записывается в логике предикатов глобально для класса систем $\mathfrak{A}$, определенного в примере выше.

Упражнение 7.2.2 (***). *Докажите, что существует сигнатура Ω и рекурсивная программа P в этой сигнатуре такие, что глобальный для класса всех конечных систем сигнатуры Ω предикат $P[2]$ не задается в виде $Q[2]$ ни для какой простой программы Q в сигнатуре Ω.*

Например, рассмотрите сигнатуру $\langle f^{(1)}, g^{(1)} \rangle$, где f, g — одноместные функциональные символы, и класс конечных систем этой сигнатуры.

7.3 Полиномиальная реализуемость рекурсивных алгоритмов

Нам понадобится также некоторый эквивалентный приведенному, но технически более удобный, способ определения отображения на множестве оценок, задаваемого простой и рекурсивной программами.

Пусть фиксирована конечная сигнатура Ω и список переменных

$$x_1, x_2, \dots,$$

а тем самым язык L_Ω логики предикатов сигнатуры Ω. Рассмотрим также некоторую конечную алгебраическую систему $\mathcal{A}$ сигнатуры Ω.

Определение 7.3.1 (линейной программы).

Линейной программой (в сигнатуре Ω) называется слово, составленное из присваиваний в сигнатуре Ω (см. определение 7.2.1) и формул сигнатуры Ω. Мы уже знаем, что произвольное присваивание $x_i := t$ определяет отображение множества оценок в системе $\mathcal{A}$ в себя.

Пусть φ — формула L_Ω. φ определяет частичное отображение множества оценок в системе $\mathcal{A}$ в себя так. Пусть $\sigma : \{x_1, x_2, \dots\} \to |\mathcal{A}|$ — некоторая оценка. Тогда $\varphi(\sigma)$ — оценка, определенная так:

$$\varphi(\sigma) = \begin{cases} \sigma, & \text{если } \sigma \models \varphi; \\ & \text{не определена в остальных случаях.} \end{cases}$$

Линейная программа $\alpha = \alpha_1 \alpha_2 \dots \alpha_n$ также определяет частичное отображение множества оценок в системе $\mathcal{A}$ в себя. Пусть σ — неко-

торая оценка. Тогда $\alpha(\sigma)$ *— оценка, определенная так:*

$$\alpha(\sigma) = \begin{cases} \sigma, \text{ если } n = 0 \text{ (то есть } \alpha \text{ — пустое слово)}; \\ (\alpha_2\alpha_3 \ldots \alpha_n)(\alpha_1(\sigma)) \text{ в остальных случаях.} \end{cases}$$

Выясняется, что всякой рекурсивной программе P можно поставить в соответствие некоторый такой КС-язык L_P в некотором конечном алфавите формул и присваиваний (то есть множество линейных программ), что для произвольной оценки $\sigma : \{x_1, x_2, \ldots\} \to |\mathcal{A}|$ произвольной системы $\mathcal{A}$ оценка $P(\sigma)$ определена тогда и только тогда, когда найдется такое $\alpha \in L_P$, что $\alpha(\sigma)$ определено.

Сделаем это формально.

Определение 7.3.2 (путевой грамматики программы). *Пусть* $P = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ *— рекурсивная программа, где* $\alpha_1, \alpha_2, \ldots, \alpha_n$ *— рекурсивные* n*-инструкции,* $\mathfrak{E}$ *— множество (очевидно, конечное) всех использованных в программе* P *присваиваний и формул, а также отрицаний этих формул.*

Рассмотрим КС-грамматику

$$\mathfrak{T}_P = \langle \mathfrak{E}, \{S_1, S_2, \ldots, S_n\}, S_1, \Pi \rangle,$$

где $\mathfrak{E}$ *— алфавит терминалов,* $\{S_1, S_2, \ldots, S_n\}$ *— алфавит нетерминалов,* S_1 *— начальный нетерминал,* $\Pi = \bigcup_{i=1}^{n} \Pi_i$*, где* Π_i *есть множество всех правил из* Π *с левой частью* S_i *и определено следующим образом.*

Пусть $\alpha_i = \langle I_i, m_i \rangle$*. Тогда*

$$\Pi_i = \begin{cases} \{S_i \to I_i S_{m_i}\}, \text{ если } I_i \text{ есть присваивание}; \\ \{S_i \to \varphi S_j,\ S_i \to (\neg\varphi) S_{m_i}\}, \text{ если } I_i \text{ есть } if\ \varphi\ goto\ j; \\ \{S_i \to S_j S_{m_i}\}, \text{ если } I_i \text{ есть } call\ j; \\ \{S_i \to e\}, \text{ где } e \text{ — пустое слово, если } I_i \text{ есть } stop. \end{cases}$$

КС-грамматика $\mathfrak{T}_P$ *называется путевой грамматикой программы* P.

Очевидно, что язык грамматики $\mathfrak{T}_P$ всегда является множеством линейных программ.

Теорема 7.1. *Пусть* $P = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ *— рекурсивная программа в некоторой сигнатуре* Ω*,* $\mathfrak{E}$ *— множество всех использованных в программе* P *присваиваний, формул, а также отрицаний этих формул и*

$$\mathfrak{T}_P = \langle \mathfrak{E}, \{S_1, S_2, \ldots, S_n\}, S_1, \Pi \rangle$$

есть путевая грамматика программы P. Пусть $\mathcal{A}$ есть произвольная конечная алгебраическая система сигнатуры Ω, а $\sigma : \{x_1, x_2, \dots\} \to |\mathcal{A}|$ есть произвольная оценка для системы $\mathcal{A}$.

Тогда

(1) существует не более одной такой линейной программы $\alpha \in L(\mathfrak{T}_P)$, что $\alpha(\sigma)$ определена;

(2) $P^{\mathcal{A}}(\sigma)$ определена в том и только в том случае, когда существует такая линейная программа $\alpha \in L(\mathfrak{T}_P)$, что $\alpha(\sigma)$ определена, причем в этом случае $P^{\mathcal{A}}(\sigma) = \alpha(\sigma)$.

Доказательство. Рутинное упражнение на понимание определений.

Рассмотрим, например, пункт (1). Предположим противное, то есть что существуют такие система $\mathcal{A}$ и ее оценка σ, что одновременно сыщутся две такие различные линейные программы $\alpha, \beta \in \mathfrak{T}_P$, что $\alpha(\sigma)$ и $\beta(\sigma)$ определены. Пусть

$$\alpha = \alpha_1 \alpha_2 \dots \alpha_n,$$

$$\beta = \beta_1 \beta_2 \dots \beta_m.$$

Поскольку $\alpha \neq \beta$, найдется минимальное k, $0 < k \leq m$, $0 < k \leq n$, что

$$\alpha_1 = \beta_1, \alpha_2 = \beta_2, \alpha_{k-1} = \beta_{k-1},$$

но $\alpha_k \neq \beta_k$. Тогда из определения 7.3.2 непосредственно следует, что α_k есть φ, и β_k есть $\neg(\varphi)$ для некоторой формулы φ (либо же наоборот).

Действительно, в левых выводах этих слов в рассматриваемой грамматике найдется первый шаг, когда используются разные правила. По определению путевой грамматики, это только может быть случай, когда используется правило для нетерминала S_i и I_i является переходом.

По определению 7.3.1,

$$\alpha(\sigma) = (\alpha_{k+1} \alpha_{k+2} \dots \alpha_n)(\alpha_k((\alpha_1 \alpha_2 \dots \alpha_{k-1})(\sigma))),$$

$$\beta(\sigma) = (\beta_{k+1} \beta_{k+2} \dots \beta_m)(\beta_k((\beta_1 \beta_2 \dots \beta_{k-1})(\sigma))),$$

Но так как $\alpha_1 = \beta_1$, $\alpha_2 = \beta_2$, $\dots, \alpha_{k-1} = \beta_{k-1}$, то и

$$\alpha_1 \alpha_2 \dots \alpha_{k-1} = \beta_1 \beta_2 \dots \beta_{k-1},$$

а следовательно,

$$(\alpha_1 \alpha_2 \dots \alpha_{k-1})(\sigma) = (\beta_1 \beta_2 \dots \beta_{k-1})(\sigma) = \sigma'$$

для некоторой σ'.

Тогда либо $\sigma' \models \varphi$, либо $\sigma' \models \neg\varphi$. По определению 7.3.1, тогда либо

$$\alpha_k((\alpha_1\alpha_2\ldots\alpha_{k-1})(\sigma)) = \alpha_k(\sigma') = \varphi(\sigma'),$$

либо

$$\beta_k((\beta_1\beta_2\ldots\beta_{k-1})(\sigma)) = \beta_k(\sigma') = \neg(\varphi)(\sigma')$$

не определено. Значит, либо

$$\alpha(\sigma) = (\alpha_{k+1}\alpha_{k+2}\ldots\alpha_n)(\alpha_k((\alpha_1\alpha_2\ldots\alpha_{k-1})(\sigma))) =$$

$$(\alpha_{k+1}\alpha_{k+2}\ldots\alpha_n)(\varphi(\sigma')),$$

либо

$$\beta(\sigma) = (\beta_{k+1}\beta_{k+2}\ldots\beta_m)(\beta_k((\beta_1\beta_2\ldots\beta_{k-1})(\sigma))) =$$

$$(\beta_{k+1}\beta_{k+2}\ldots\beta_m)(\neg\varphi(\sigma'))$$

не определено, что противоречит выбору α и β. Этим доказан пункт (1) теоремы.

Читателю в качестве упражнения предлагается самостоятельно доказать пункт (2). ∎

Теорема 7.2. *Пусть $P = \langle\alpha_1, \alpha_2, \ldots, \alpha_n\rangle$ — простая программа в некоторой сигнатуре Ω, а*

$$\mathfrak{T}_P = \langle\mathfrak{E}, \{S_1, S_2, \ldots, S_n\}, S_1, \Pi\rangle$$

является путевой грамматикой программы P. Язык $L(\mathfrak{T}_P)$ регулярен.

Доказательство. Непосредственное следствие определения 7.3.2. Дело в том, что в простую программу инструкции *call* не входят, следовательно, все правила КС-грамматики $\mathfrak{T}_P$ имеют вид $S \to e$ или $S \to aS'$, где e — пустое слово, a — терминальный символ, S, S' — нетерминальные символы. Значит, грамматика $\mathfrak{T}_P$ может быть преобразована к праволинейной, и язык $L(\mathfrak{T}_P)$ является регулярным (см. теорему 2.26).

Для преобразования грамматики $\mathfrak{T}_P$ к праволинейной надо только избавиться от правил вида $S_i \to e$, где e — пустое слово, $i \neq 1$. Пусть

$$S_{i_1} \to e, \ldots, S_{i_r} \to e$$

есть все такие e-правила грамматики $\mathfrak{T}_P$. Заметим, что по определению 7.3.2 в грамматике нет других правил с левой частью

$$S_{i_1}, S_{i_2}, \ldots, S_{i_r}.$$

Выбросим из грамматики $\mathfrak{T}_P$ все эти правила, и в правых частях оставшихся правил заменим все вхождения символов

$$S_{i_1}, S_{i_2}, \ldots, S_{i_r}$$

на пустые слова. Полученная таким образом КС-грамматика эквивалентна $\mathfrak{T}_P$ и является, очевидно, праволинейной. ∎

Теперь мы хотим перейти к рассмотрению вычислимых глобальных предикатов. В частности, мы хотим показать, что глобальные предикаты, задаваемые при помощи рекурсивных программ, $\mathcal{P}$-вычислимы.

Для этого нам нужны точные определения $\mathcal{P}$-вычислимого предиката, и в частности, определение кода конечной алгебраической системы и ее состояния.

Определение 7.3.3. *Рассмотрим сигнатуру*

$$\Omega = \langle f_1^{(i_1)}, f_2^{(i_2)}, \ldots, f_l^{(i_l)}, R_1^{(j_1)}, R_2^{(j_2)}, \ldots, R_m^{(j_m)} \rangle,$$

где $f_1, f_2, \ldots, f_l$ — функциональные символы местности, соответственно, $i_1, i_2, \ldots, i_l$, а $R_1, R_2, \ldots, R_m$ — предикатные символы местности, соответственно, $j_1, j_2, \ldots, j_m$. Мы считаем, что все элементы Ω упорядочены в приведенном порядке.

Пусть $\mathcal{A}$ — некоторая конечная алгебраическая система сигнатуры Ω, то есть конечное основное множество $|\mathcal{A}|$ вместе с отображением, которое каждому предикатному символу $R_k \in \Omega$ ставит в соответствие j_k -местный предикат

$$R_k^{\mathcal{A}} \subseteq |\mathcal{A}|^{j_k}$$

на множестве $|\mathcal{A}|$, а каждому функциональному символу $f_k \in \Omega$ — i_k-местную функцию

$$f_k^{\mathcal{A}} : |\mathcal{A}|^{i_k} \to |\mathcal{A}|$$

на множестве $|\mathcal{A}|$.

Без ограничения общности будем считать, что множество $|\mathcal{A}|$ представляет собой начальный отрезок натурального ряда, то есть что, для $n = ||\mathcal{A}||$, $|\mathcal{A}| = \{1, 2, \ldots, n\}$. При этом все элементы $|\mathcal{A}|$ естественным образом упорядочены. Этот порядок на элементах индуцирует и порядок на кортежах элементов, именно, для произвольного $q \in \{0, 1, \ldots\}$ и для произвольных

$$\alpha_1, \alpha_2, \ldots, \alpha_q, \beta_1, \beta_2, \ldots, \beta_q \in |\mathcal{A}|$$

считаем, что

$$\langle \alpha_1, \alpha_2, \dots, \alpha_q \rangle$$

меньше

$$\langle \beta_1, \beta_2, \dots, \beta_q \rangle,$$

если существует такое $s \le q$, что

$$\alpha_1 = \beta_1, \alpha_2 = \beta_2, \dots, \alpha_{s-1} = \beta_{s-1},$$

но $\alpha_s < \beta_s$. Пусть

$$\vartheta_1^q, \vartheta_2^q, \dots, \vartheta_{n^q}^q$$

перечисляют все кортежи из q элементов множества

$$|\mathcal{A}| = \{1, 2, \dots, n\},$$

взятые в определенном выше порядке, то есть
$\vartheta_1^q = \langle 1, 1, \dots, 1 \rangle$, $\vartheta_2^q = \langle 1, 1, \dots, 1, 2 \rangle, \dots,$ $\vartheta_{n^q}^q = \langle n, n, \dots, n \rangle$.

Кодом натурального числа $t \in \{0, 1, \dots\}$ будем называть слово $\#(t)$ в алфавите $\{0, 1\}$, представляющее собой двоичную запись числа t.

Кодом предиката $R_k^{\mathcal{A}} \subseteq |\mathcal{A}|^{j_k}$, будем называть слово

$$\#(R_k^{\mathcal{A}}) = \xi_1 \xi_2 \dots \xi_{n^{j_k}} \in \{0, 1\}^*$$

длины n^{j_k}, в котором для $u \le n^{j_k}$

$$\xi_u = \begin{cases} 1, & \textit{если } \vartheta_u^{j_k} \in R_k^{\mathcal{A}}; \\ 0, & \textit{если } \vartheta_u^{j_k} \notin R_k^{\mathcal{A}}. \end{cases}$$

Кодом функции

$$f_k^{\mathcal{A}} : |\mathcal{A}|^{j_k} \to |\mathcal{A}|$$

будем называть слово

$$\#(f_k^{\mathcal{A}}) = \xi_1 \xi_2 \dots \xi_{n^{j_k+1}} \in \{0, 1\}^*$$

длины n^{j_k+1}, представляющее собой код графика функции $f_k^{\mathcal{A}}$, то есть $j_k + 1$-местного предиката $G(f_k^{\mathcal{A}})$, где

$$\langle a_1, a_2, \dots, a_{j_k}, a_{j_k+1} \rangle \in G(f_k^{\mathcal{A}}) \Leftrightarrow f_k^{\mathcal{A}}(a_1, a_2, \dots, a_{j_k}) = a_{j_k+1}.$$

Наконец, кодом самой системы $\mathcal{A}$ будем называть слово

$$\#(\mathcal{A}) \in \{0, 1, *\}^*,$$

где

$$\#(\mathcal{A}) = *\#(f_1^{\mathcal{A}}) * \#(f_2^{\mathcal{A}}) * \dots * \#(f_l^{\mathcal{A}}) * \#(R_1^{\mathcal{A}}) * \#(R_2^{\mathcal{A}}) * \dots * \#(R_m^{\mathcal{A}}) * .$$

Определение 7.3.4 (X-кода). *В обозначениях определения 7.3.3, пусть фиксированы некоторая оценка*

$$\sigma : \{x_1, x_2, \dots\} \to |\mathcal{A}|$$

и конечный список переменных

$$X = \{x_{h_1}, x_{h_2}, \dots, x_{h_v}\},$$

где $0 < h_1 < h_2 < \dots < h_v$. *$X$-кодом оценки σ будем называть слово* $\#(X, \sigma) \in \{0, 1, *\}^*$,

$$\#(X, \sigma) = *\#(\sigma(x_{h_1})) * \#(\sigma(x_{h_2})) * \dots * \#(\sigma(x_{h_v})).$$

Главным результатом настоящего параграфа является

Теорема 7.3. *Пусть Ω — некоторая конечная сигнатура, P — рекурсивная программа в сигнатуре Ω, m — число использованных в программе P переменных, $0 < v \leq m$ — натуральное число, $X = \{x_{h_1}, x_{h_2}, \dots, x_{h_v}\}$ — последовательность первых v переменных программы P, взятых в порядке возрастания их номеров. Пусть $\mathfrak{A}$ — класс всех конечных алгебраических систем сигнатуры Ω и $\Sigma_{\mathcal{A}}$ — множество всех оценок в системе $\mathcal{A}$ для $\mathcal{A} \in \mathfrak{A}$. Пусть*

$$\mathfrak{P} = \{\#(\mathcal{A})\#(X, \sigma) \mid \mathcal{A} \in \mathfrak{A}, \sigma \in \Sigma_{\mathcal{A}}, \langle \sigma(x_{h_1}), \sigma(x_{h_2}), \dots, \sigma(x_{h_v})\rangle \in P^{\mathcal{A}, v}\}.$$

Тогда $\mathfrak{P} \in \mathcal{P} = PTIME$.

Прежде, чем приступить к доказательству теоремы 7.3, отметим очевидное

Следствие 7.4. *В обозначениях теоремы 7.3, $\mathfrak{A}$-глобальный предикат*

$$P(v) : \mathcal{A} \longmapsto P^{\mathcal{A}, v}$$

$\mathcal{P}$-вычислим.

Доказательство теоремы 7.3. Пусть $\mathfrak{E}$ — множество всех использованных в программе P присваиваний, формул, а также отрицаний этих формул и

$$\mathfrak{T}_P = \langle \mathfrak{E}, \{S_1, S_2, \dots, S_n\}, S_1, \Pi\rangle$$

является путевой грамматикой программы P. Пусть список

$$Z = \langle x_{h_1}, x_{h_2}, \dots, x_{h_m}\rangle$$

представляет собой последовательность всех использованных в P переменных, в порядке возрастания их номеров.

Пусть $\mathcal{A} \in \mathfrak{A}$ — произвольная конечная алгебраическая система, $||\mathcal{A}|| = n$,

$$\sigma : \{x_1, x_2, \dots\} \to |\mathcal{A}|$$

является произвольной оценкой для системы $\mathcal{A}$.

Рассмотрим множество $L(\mathfrak{E}_{\mathcal{A},\sigma})$ всех таких слов в алфавите $\mathfrak{E}$ (то есть линейных программ), что для произвольного $\alpha \in \mathfrak{E}^*$ имеет место

$$\alpha \in L(\mathfrak{E}_{\mathcal{A},\sigma}) \Leftrightarrow \alpha(\sigma) \text{ определена.}$$

Наша цель — показать, что $L(\mathfrak{E}_{\mathcal{A},\sigma})$ — регулярное множество в алфавите $\mathfrak{E}$, причем распознающий его конечный автомат может быть построен эффективно по $\#(\mathcal{A})$, $\#(X, \sigma)$ и по множеству $\mathfrak{E}$.

Z-оценкой будем называть отображение множества Z во множество $|\mathcal{A}|$, то есть ограничение оценки на множество Z. Ясно, что для произвольного $\gamma \in \mathfrak{E}$ (а, значит, и для произвольного $\gamma \in \mathfrak{E}^*$), и для произвольных оценок $\sigma, \sigma' \in \Sigma_{\mathcal{A}}$

$$\sigma \restriction Z = \sigma' \restriction Z \Longrightarrow \gamma(\sigma) = \gamma(\sigma'),$$

где $\sigma \restriction Z$ — ограничение оценки σ на множество переменных Z.

В дальнейшей части доказательства мы будем понимать под оценкой Z-оценку. Очевидно, что в системе $\mathcal{A}$ существует в точности n^m различных Z-оценок. Пусть эти оценки образуют множество

$$\Sigma_{\mathcal{A},Z} = \{\sigma_1, \sigma_2, \dots, \sigma_k\}, k = n^m.$$

Пусть

$$\mathfrak{D} = \{d_1, d_2, \dots, d_r\}$$

задает все присваивания из $\mathfrak{E}$ и

$$\mathfrak{F} = \{t_1, t_2, \dots, t_w\}$$

перечисляет все формулы из $\mathfrak{E}$. Будем говорить, что d_i имеет тип (j, q), если $d_i(\sigma_j) = \sigma_q$. Будем говорить, что t_i имеет тип j, если $\sigma_j \models t_i$. Итак, нами приписан тип каждому элементу $\gamma \in \mathfrak{E}$. Пусть

$$\mathfrak{D}_{i,j} = \{d_1^{i,j}, d_2^{i,j}, \dots, d_{r_{i,j}}^{i,j}\}$$

задает множество всех присваиваний типа (i, j) из $\mathfrak{E}$, а

$$\mathfrak{F}_i = \{t_1^i, t_2^i, \dots, t_{w_i}^i\}$$

задает множество всех формул типа i из $\mathfrak{E}$.

Приступим к построению недетерминированного конечного автомата

$$\mathfrak{I}_{\mathcal{A},i} = \langle \mathfrak{E}, \Sigma_{\mathcal{A},Z}, \sigma_i, \Sigma_{\mathcal{A},Z}, R \rangle,$$

где $\mathfrak{E}$ — входной алфавит, $\Sigma_{\mathcal{A},Z}$ — алфавит состояний, совпадающий с множеством заключительных состояний, σ_i — начальное состояние и

$$R \subseteq (\Sigma_{\mathcal{A},Z} \times \mathfrak{E} \times \Sigma_{\mathcal{A},Z})$$

является множеством правил, определенным следующим образом.

а). Для каждых $\sigma_j, \sigma_q \in \Sigma_{\mathcal{A},Z}$ в R включены правила

$$\langle \sigma_j, d_l^{j,q}, \sigma_q \rangle$$

для всех $d_l^{j,q} \in \mathfrak{D}_{j,q}$.

б). Для каждого $\sigma_j \in \Sigma_{\mathcal{A},Z}$ в R включены правила

$$\langle \sigma_j, t_l^j, \sigma_j \rangle$$

для всех $t_l^j \in \mathfrak{F}_j$.

Очевидно, что построенный таким образом НКА $\mathfrak{I}_{\mathcal{A},i}$ отличается тем свойством, что для всякого $\alpha \in \mathfrak{E}^*$ имеет место

$$\alpha \in L(\mathfrak{I}_{\mathcal{A},i}) \Leftrightarrow \alpha(\sigma_i) \text{ определено.}$$

Следовательно, $L(\mathfrak{E}_{\mathcal{A},\sigma})$ — регулярное множество в алфавите $\mathfrak{E}$, поскольку оно распознается НКА $\mathfrak{I}_{\mathcal{A},i}$.

Очевидно также, что НКА $\mathfrak{I}_{\mathcal{A},i}$ может быть построен эффективно по $\#(\mathcal{A})$, $\#(Z, \sigma_i)$ и по множеству $\mathfrak{E}$.

Рассмотрим многоленточную детерминированную машину Тьюринга $M_{\mathfrak{E}}$, которая получает на входной ленте слово $\#(\mathcal{A})\#(X, \sigma)$ и пишет на выходную ленту слово $\#(\mathfrak{I}_{\mathcal{A},i})$, то есть код НКА $\mathfrak{I}_{\mathcal{A},i}$ (см.§6.5). Здесь i есть номер такой Z-оценки, что $\sigma_i(x) = \sigma(x)$ для всякого $x \in X$ и $\sigma_i(y) = \sigma(x_{h_1})$ для всякого $y \notin X$, где, напомним,

$$X = \{x_{h_1}, x_{h_2}, \ldots, x_{h_v}\} \subseteq Z$$

и X определено в теореме 7.3.

Единственный не вполне тривиальный шаг в построении $M_{\mathfrak{E}}$ согласно описанной выше неформально процедуре — построение для каждого $i \leq k$ множества

$$\mathfrak{F}_i = \{t_1^i, t_2^i, \ldots, t_{w_i}^i\},$$

задающего множество всех формул типа i из $\mathfrak{E}$. Здесь для определения истинности атомных формул на данной оценке следует пользоваться $\#(\mathcal{A})$ — кодом системы $\mathcal{A}$, для определения истинности конъюнкций, дизъюнкций и отрицаний — стандартными правилами определения истинности, а для определения истинности формул с кванторами следует производить перебор всех оценок, отличающихся от данной только на переменной, которая стоит под квантором.

Легко видеть, что построенная $M_{\mathfrak{E}}$ такова, что время ее работы $t_{M_{\mathfrak{E}}}$ асимптотически мажорируется некоторым полиномом от длины $\#(\mathcal{A})\#(X,\sigma)$.

Для завершения доказательства теоремы 7.3 рассмотрим детерминированный МТ распознаватель M_P, который получает на входе

$$\#(\mathcal{A})\#(X,\sigma)$$

и работает следующим образом.

а). Пишет $\#(\mathfrak{T}_P)$ на первую рабочую ленту, где

$$\mathfrak{T}_P = \langle \mathfrak{E}, \{S_1, S_2, \ldots, S_n\}, S_1, \Pi\rangle$$

является путевой грамматикой программы P, $\#(\mathfrak{T}_P)$ — ее код (см. §6.5).

б). Используя простой метод теоремы 2.29, преобразует $\#(\mathfrak{T}_P)$ на первой рабочей ленте в $\#(\mathfrak{J}_P)$ на второй рабочей ленте, где $\#(\mathfrak{J}_P)$ — код такого МП-автомата (см. §6.5), что $L(\mathfrak{J}_P) = L(\mathfrak{T}_P)$.

в). Использует обсуждавшуюся выше МТ $M_{\mathfrak{E}}$ и пишет $\#(\mathfrak{J}_{\mathcal{A},i})$ на вторую же рабочую ленту после $\#(\mathfrak{J}_P)$ через два знака ;; (точки с запятой).

г). Теперь, в соответствии с теоремой 6.11, определяет пустоту пересечения $L(\mathfrak{J}_{\mathcal{A},i})$ и $L(\mathfrak{J}_P)$, то есть пишет 0 или 1 на выходную ленту (этот шаг может быть сделан за полиномиальное время согласно теореме 6.11).

Итак, за полиномиальное от длины входа (т.е. от длины кода $\#(\mathcal{A})\#(X,\sigma)$, а значит, и от n) время машина M_P определяет, пусто ли пересечение $L(\mathfrak{J}_{\mathcal{A},i})$ и $L(\mathfrak{J}_P)$. Тем самым она определяет и пусто ли пересечение $L(\mathfrak{J}_{\mathcal{A},i})$ и $L(\mathfrak{T}_P)$.

Отметим, что

$$L(\mathfrak{J}_{\mathcal{A},i}) \bigcap L(\mathfrak{T}_P) \neq \emptyset$$

тогда и только тогда, когда существует такая линейная программа $\alpha \in L(\mathfrak{T}_P)$, что $\alpha(\sigma_i)$ определена, то есть, по теореме 7.1, в том и только том случае, когда $P^{\mathcal{A}}(\sigma_i)$ определена. Но, с учетом определения σ_i по X и

σ, $P^{\mathcal{A}}(\sigma_i)$ определена тогда и только тогда, когда

$$\langle \sigma(x_{h_1}), \sigma(x_{h_2}), \ldots, \sigma(x_{h_v}) \rangle \in P^{\mathcal{A},v}.$$

Итак

$$L(M_P) = \mathfrak{P} = \{\#(\mathcal{A})\#(X,\sigma) \mid$$
$$\mathcal{A} \in \mathfrak{A}, \sigma \in \Sigma_{\mathcal{A}}, \langle \sigma(x_{h_1}), \sigma(x_{h_2}), \ldots, \sigma(x_{h_v}) \rangle \in P^{\mathcal{A},v}\} \in \mathcal{P}.$$

Надо, конечно, в самом начале работы машины M определять, записан ли на ней код некоторой конечной системы и некоторого ее состояния. Это делается тривиально. ∎

Фактически нами неявно введено фундаментальное понятие сложности глобального предиката.

Определение 7.3.5 (сложности глобального предиката).

Пусть Ω — некоторая конечная сигнатура, $v > 0$ — натуральное число. Пусть $\mathfrak{A}$ — некоторый класс конечных алгебраических систем сигнатуры Ω. Пусть Ξ — v-местный $\mathfrak{A}$-глобальный предикат,

$$\Xi : \mathcal{A} \longmapsto \Xi^{\mathcal{A}}.$$

Пусть $\mathfrak{P}$ — такой язык в алфавите $\{0, 1, *\}$, что

$$\mathfrak{P} = \{\#(\mathcal{A}) * \#(a_1) * \#(a_2) * \cdots * \#(a_v) \mid$$
$$\mathcal{A} \in \mathfrak{A}; a_1, a_2, \ldots, a_v \in |\mathcal{A}|, \langle a_1, a_2, \ldots, a_v \rangle \in \Xi^{\mathcal{A}}\}.$$

Если $\mathcal{C}$ — некоторый такой класс сложности, что $\mathfrak{P} \in \mathcal{C}$, то будем говорить, что $\mathfrak{A}$-глобальный предикат Ξ имеет сложность $\mathcal{C}$.

Следствие 7.5. *В обозначениях теоремы 7.3, $\mathfrak{A}$-глобальный предикат*

$$P[v] : \mathcal{A} \longmapsto P^{\mathcal{A},v}$$

имеет сложность $\mathcal{P}$.

Упражнения

Упражнение 7.3.1. *Докажите пункт (2) теоремы 7.1.*

Упражнение 7.3.2. *Пусть Ω — некоторая конечная реляционная сигнатура, $\mathfrak{A}$ — некоторый класс конечных алгебраических систем сигнатуры Ω.*

Покажите, что для всякой рекурсивной программы P и всякого натурального m найдется такая формула φ сигнатуры Ω, что $\mathfrak{A}$-глобальные предикаты $P[m] : \mathcal{A} \longmapsto P^{\mathcal{A},m}$ и $\varphi : \mathcal{A} \longmapsto \varphi^{\mathcal{A}}$ совпадают.

7.4 Полнота рекурсивных алгоритмов для полиномиальных глобальных предикатов

В предыдущих параграфах мы рассматривали разные способы задания глобальных предикатов (формулами логики предикатов, программами). Из приводившегося примера с транзитивным замыканием ясно, что рекурсивные программы представляют собой более мощный способ задания глобальных предикатов, чем формулы. Однако мы можем предложить и более изощренные способы описания глобальных предикатов, с помощью которых сможем задать глобальные предикаты, не задаваемые рекурсивными программами.

Фактическим преимуществом рекурсивных программ перед формулами с точки зрения задания глобальных предикатов является то, что сложность, которую имеют глобальные предикаты, остается той же самой. Действительно, нетрудно показать, что задаваемые формулами глобальные предикаты имеют сложность $\mathcal{P}$ и ни в какой меньший класс детерминированной временной сложности не попадают (см. упражнение 7.4.1).

Определение 7.4.1 (полноты класса глобальных предикатов). *Пусть Ω — некоторая конечная сигнатура, $\mathfrak{A}$ — некоторый класс конечных алгебраических систем сигнатуры Ω, $\mathcal{C}$ — некоторый класс сложности.*

Пусть $\mathfrak{B}$ — некоторый такой класс $\mathfrak{A}$-глобальных предикатов, что каждый $\mathfrak{A}$-глобальный предикат $\mathfrak{P} \in \mathfrak{B}$ имеет сложность $\mathcal{C}$. Будем говорить, что $\mathfrak{B}$ является $\mathcal{C}$-полным для $\mathfrak{A}$, если для всякого $\mathfrak{A}$-глобального предиката $\Xi : \mathcal{A} \longmapsto \Xi^{\mathcal{A}}$, имеющего сложность $\mathcal{C}$, найдется такой $\mathfrak{P} \in \mathfrak{B}$, что для всякой системы $\mathcal{A} \in \mathfrak{A}$ предикаты $\Xi^{\mathcal{A}}$ и $\mathfrak{P}^{\mathcal{A}}$ совпадают.

Реально преимущество программ перед формулами проявляется тогда, когда в системе наличествуют некоторые функции (см. упражнение 7.3.2). Фактически в дальнейшем мы будем рассматривать поэтому только такие конечные системы, которые представляют собой начальные отрезки натурального ряда с функцией следования, и некоторыми другими предикатами функциями (еще раз напомним, что символ равенства -"считается логическим, то есть не входит в сигнатуру, но всегда может быть использован при построении формул; при этом мы всегда интерпретируем его как равенство).

Мы будем рассматривать конечные алгебраические системы специального вида.

Определение 7.4.2 (арифметической системы).

Пусть $\Omega = \langle o, f, \dots \rangle$ — сигнатура, содержащая нульместный функциональный символ o (константу), одноместный функциональный символ f и, возможно, некоторые другие функциональные и предикатные символы. Пусть $\mathcal{A}$ — конечная алгебраическая система сигнатуры Ω, в которой:

а) $|\mathcal{A}| = \{0, 1, 2, \dots, n\}$ для некоторого n;

б) $o^{\mathcal{A}} = 0$;

в)

$$f(i) = \begin{cases} i+1, & \text{если } i < n; \\ n & \text{в остальных случаях} \end{cases}$$

для $i \in |\mathcal{A}|$.

Такую систему $\mathcal{A}$ будем называть арифметической.

Основным результатом данного параграфа является

Теорема 7.6. *Пусть $\Omega = \langle o, f, \dots \rangle$ — сигнатура, содержащая нульместный функциональный символ o (константу), одноместный функциональный символ f и, возможно, некоторые другие функциональные и предикатные символы, $\mathfrak{A}$ — класс всех конечных арифметических алгебраических систем сигнатуры Ω.*

Пусть $\mathfrak{B}$ — класс $\mathfrak{A}$-глобальных предикатов, задаваемых рекурсивными программами, то есть для каждого $\mathfrak{A}$-глобального предиката $\mathfrak{P} \in \mathfrak{B}$ существуют такие рекурсивная программа P и натуральное m, что

$$\mathfrak{P} = P[m] : \mathcal{A} \longmapsto P^{\mathcal{A}, m}.$$

$\mathfrak{B}$ является $\mathcal{P}$-полным для $\mathfrak{A}$.

Для доказательства нам понадобится ввести одно вспомогательное понятие и доказать ряд технических лемм.

Определение 7.4.3 (простой программы со стеком). *Пусть Ω — произвольная сигнатура.Простой программой со стеком (в сигнатуре Ω) будем называть конечную последовательность пронумерованных числами от 1 до n стековых n-инструкций, где n — число элементов последовательности. Стековой n-инструкцией называется либо n-инструкция (см. определение 7.2.2), либо $\langle I, m \rangle$, где m — натуральное число, $1 \le m \le n$, а I — одна из следующих конструкций:*

а) $push\ 0$ (запись в стек 0);

б) $push\ 1$ (запись в стек 1);

в) if pop $goto$ i (переход по вершине; здесь $1 \le i \le n$);
г) if $empty$ $goto$ i (переход по пустоте; здесь $1 \le i \le n$).

Простая программа со стеком определяет частичное отображение множества всех оценок в данной алгебраической системе в себя. Дадим рекурсивное определение этого отображения.

Определение 7.4.4. *Пусть $P = \langle \alpha_1, \alpha_2, \dots, \alpha_n \rangle$ — простая программа со стеком, где $\alpha_1, \alpha_2, \dots, \alpha_n$ — стековые n-инструкции, Σ — множество всех оценок в системе $\mathcal{A}$, то есть отображений*

$$\sigma : \{x_1, x_2, \dots\} \to |\mathcal{A}|.$$

Определим для всякого $i \in \{1, 2, \dots, n\}$ частичное отображение

$$\mathfrak{S}(P, \mathcal{A}, i) : (\Sigma \times \{0,1\}^*) \to (\Sigma \times \{0,1\}^*)$$

следующим образом. Пусть $\alpha_i = \langle I_i, m_i \rangle$ и $\sigma : \{x_1, x_2, \dots\} \to |\mathcal{A}|$ является некоторой оценкой из Σ, $s \in \{0,1\}^*$ (s является стеком). Тогда

$$\mathfrak{S}(P,\mathcal{A},i)(\sigma,s) = \begin{cases} \mathfrak{S}(P,\mathcal{A},m_i)(I_i(\sigma),s), & \text{если } I_i \text{ есть присваивание;} \\ \mathfrak{S}(P,\mathcal{A},j)(\sigma,s), & \\ \qquad \text{если } I_i \text{ есть } if\ \varphi\ goto\ j \text{ и } \sigma \models \varphi; & \\ \mathfrak{S}(P,\mathcal{A},m_i)(\sigma,s), & \\ \qquad \text{если } I_i \text{ есть } if\ \varphi\ goto\ j \text{ и } \sigma \models \neg\varphi; & \\ \mathfrak{S}(P,\mathcal{A},m_i)(\sigma,\xi s), & \text{если } I_i \text{ есть } push\ \xi; \\ \mathfrak{S}(P,\mathcal{A},j)(\sigma,s'), & \\ \qquad \text{если } s = 1s' \text{ и } I_i \text{ есть } if\ pop\ goto\ j; & \\ \mathfrak{S}(P,\mathcal{A},m_i)(\sigma,s'), & \\ \qquad \text{если } s = 0s' \text{ и } I_i \text{ есть } if\ pop\ goto\ j; & \\ \mathfrak{S}(P,\mathcal{A},j)(\sigma,s), & \\ \qquad \text{если } s \text{ пуст и } I_i \text{ есть } if\ empty\ goto\ j; & \\ \mathfrak{S}(P,\mathcal{A},m_i)(\sigma,s), & \\ \qquad \text{если } s \text{ непуст и } I_i \text{ есть } if\ empty\ goto\ j; & \\ \langle \sigma, s \rangle, & \text{если } I_i \text{ есть } stop. \end{cases}$$

Определим теперь частичное отображение $P_i^{\mathcal{A}} : \Sigma \to \Sigma$ следующим образом:

$$P_i^{\mathcal{A}}(\sigma) = \mathfrak{S}(P, \mathcal{A}, i)(\sigma, e).$$

Напомним, что e обозначает пустое слово.

Затем, определим частичное отображение $P^{\mathcal{A}} : \Sigma \to \Sigma$ так:

$$P^{\mathcal{A}}(\sigma) = P_1^{\mathcal{A}}(\sigma).$$

Для программы P и натурального m, предикат $P^{\mathcal{A},m}$ и глобальный предикат $P[m]$ определяются в соответствии с определением 7.2.5.

Простая программа со стеком отличается от простой только четырьмя новыми инструкциями. Инструкции *push* 0 и *push* 1 осуществляют запись в стек соответственно 0 или 1. Инструкция *if pop goto i* работает подобно *if φ goto i* в простой программе, но осуществляет переход на метку i только в том случае, когда в вершине стека находилась 1. Независимо от того, что находилось в вершине стека, этот элемент вычеркивается (длина стека уменьшается на 1, в вершине стека оказывается следующий элемент). Инструкция *if empty goto i* работает аналогично, но осуществляет переход на метку i только в том случае, когда стек пуст.

Итак, мы имеем еще один способ определения глобальных предикатов — простые программы со стеком. Они оказываются эквивалентными рекурсивным программам. Чтобы показать это, докажем несколько лемм.

Лемма 7.7. *Пусть Ω — некоторая сигнатура, t_0, t_1 — некоторые замкнутые (то есть не содержащие переменных) термы этой сигнатуры.*

Пусть $P = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ — простая программа со стеком, где $\alpha_1, \alpha_2, \ldots, \alpha_n$ — стековые n-инструкции. Существует простая программа со стеком $Q = \langle \beta_1, \beta_2, \ldots, \beta_m \rangle$, где $\beta_1, \beta_2, \ldots, \beta_m$ — стековые n-инструкции, эквивалентная программе P в том смысле, что для произвольного натурального положительного k, не превосходящего числа переменных программы P, и произвольной такой алгебраической системы $\mathcal{A}$, что $t_0^{\mathcal{A}} \neq t_1^{\mathcal{A}}$, предикаты $P^{\mathcal{A},k}$ и $Q^{\mathcal{A},k}$ совпадают, причем в программе Q существует самое большее по одной инструкции каждого из следующих видов:

а) push 0 (запись в стек 0),

б) push 1 (запись в стек 1),

в) if pop goto i (переход по вершине),

г) if empty goto i (переход по пустоте).

Доказательство. Пусть l — максимальный номер переменной, использованной в программе P.

Рассмотрим, как удалить все, кроме одной, инструкции $push\ 0$ в программе P. Пусть $z_1, z_2, \ldots, z_r$ — номера (метки) всех n-инструкций программы P, инструкцией которых является $push\ 0$. Пусть эти n-инструкции имеют вид, соответственно,

$$\langle push\ 0, v_1\rangle, \langle push\ 0, v_2\rangle, \ldots, \langle push\ 0, v_r\rangle.$$

Неформально, идея состоит в том, чтобы, используя r новых переменных и отличные друг от друга по значению замкнутые термы $t_0^{\mathcal{A}}, t_1^{\mathcal{A}}$, вместо выполнения инструкции $push\ 0$ с меткой z_i, запомнить v_i, перейти на уникальную инструкцию $push\ 0$, и, после ее выполнения, на метку v_i.

Итак, построим новую простую программу со стеком P', которая получена из P заменой стековой n-инструкции

$$z_i : \langle push\ 0, v_i\rangle$$

на стековые инструкции:

$$\begin{array}{rr}
z_i : & \langle if\ x_l = x_l\ goto\ n+(i-1)(r+1)+1, z_i\rangle \\
n+(i-1)(r+1)+1 : & \langle x_{l+1} := t_0, n+(i-1)(r+1)+2\rangle \\
n+(i-1)(r+1)+2 : & \langle x_{l+2} := t_0, n+(i-1)(r+1)+3\rangle \\
\ldots & \ldots \\
n+(i-1)(r+1)+(i-1) : & \langle x_{l+i-1} := t_0, n+(i-1)(r+1)+i\rangle \\
n+(i-1)(r+1)+i : & \langle x_{l+i} := t_1, n+(i-1)(r+1)+i+1\rangle \\
n+(i-1)(r+1)+(i+1) : & \langle x_{l+i+1} := t_0, n+(i-1)(r+1)+i+2\rangle \\
n+(i-1)(r+1)+(i+2) : & \langle x_{l+i+2} := t_0, n+(i-1)(r+1)+i+3\rangle \\
\ldots & \ldots \\
n+(i-1)(r+1)+r : & \langle x_{l+r} := t_0, n+(i-1)(r+1)+r+1\rangle \\
n+(i-1)(r+1)+(r+1) : & \langle if\ x_{l+1} = x_{l+1}\ goto\ n+r(r+1)+1, z_i\rangle
\end{array}$$

Затем добавим к полученной стековой программе стековые инструкции:

$$\begin{array}{rr}
n+r(r+1)+1 : & \langle push\ 0, n+r(r+1)+2\rangle \\
n+r(r+1)+2 : & \langle if\ x_{l+1} = t_1\ goto\ v_1, n+r(r+1)+3\rangle \\
n+r(r+1)+3 : & \langle if\ x_{l+2} = t_1\ goto\ v_2, n+r(r+1)+4\rangle \\
\ldots & \ldots \\
n+r(r+1)+r+1 : & \langle if\ x_{l+r} = t_1\ goto\ v_r, n+r(r+1)+r+1\rangle
\end{array}$$

Итак, мы получили новую простую программу со стеком (с

$$n + (r+1)(r+1)$$

инструкциями), в которой инструкция $push\ 0$ либо вообще не встречается (если ее не было в P), либо же встречается ровно один раз.

Легко видеть, что это преобразование сохраняет эквивалентность программ в смысле формулировки леммы.

Аналогично можно устранить все, кроме, быть может, одной, инструкции $push\ 1, if\ pop\ goto\ i, if\ empty\ goto\ i$. В последних двух случаях надо после "удачного"выполнения, например, инструкции $if\ pop\ goto\ i$ идти на блок переходов к "удачным"меткам, иначе — на блок перехода к "неудачным"меткам. ∎

Лемма 7.8. *Пусть Ω — некоторая сигнатура. Пусть P — рекурсивная программа. Существует простая программа со стеком Q, эквивалентная программе P в том смысле, что для произвольного натурального положительного k, не превосходящего числа переменных программы P, и произвольной алгебраической системы $\mathcal{A}$ предикаты $P^{\mathcal{A},k}$ и $Q^{\mathcal{A},k}$ совпадают.*

Доказательство. Пусть n — число инструкций в программе P, t_0 — некоторый терм, содержащий только входящие в программу P переменные (без ограничения общности можно считать, что такой существует). Для простоты рассмотрим случай, когда в P существует ровно две рекурсивные n-инструкции $call$ с номерами (метками) z_1, z_2 вида

$$z_1 : \langle call\ u_1,\ v_1\rangle$$

$$z_2 : \langle call\ u_2,\ v_2\rangle$$

Пусть, кроме того, последовательность

$$z_3 : \langle stop,\ v_3\rangle$$

$$z_4 : \langle stop,\ v_4\rangle$$

$$\dots$$

$$z_r : \langle stop,\ v_r\rangle$$

содержит все инструкции $stop$ из программы P.

Неформально, идея преобразования состоит в том, чтобы заменить вызов $call\ u_1$ записью в стек 0 и переходом к u_1, вызов $call\ u_2$ записью в стек 1 и переходом к u_2, а *stop* — чтением верхнего элемента стека и одним из следующих действий:

а) переходом к v_1, если прочитанный верхний элемент стека есть 0;

б) переходом к v_2, если прочитанный верхний элемент стека есть 1;

в) остановом (*stop*), если стек пуст.

Формально, построим новую программу Q, заменив в P инструкцию с номером z_1 на инструкцию:

$$z_1 : \quad \langle if\ t_0 = t_0\ goto\ n+1, z_1\rangle,$$

инструкцию с номером z_2 на инструкцию:

$$z_2 : \quad \langle if\ t_0 = t_0\ goto\ n+3, z_2\rangle,$$

инструкции с номерами $z_3, z_4, \ldots, z_r$ на инструкции:

$$\begin{aligned}
&z_3 : \langle if\ t_0 = t_0\ goto\ n+5, v_3\rangle\\
&z_4 : \langle if\ t_0 = t_0\ goto\ n+5, v_4\rangle\\
&\ldots\ldots\\
&z_r : \langle if\ t_0 = t_0\ goto\ n+5, v_r\rangle,
\end{aligned}$$

и добавив инструкции:

$$\begin{aligned}
&n+1 : \langle push\ 0, n+2\rangle\\
&n+2 : \langle if\ t_0 = t_0\ goto\ u_1, n+2\rangle\\
&n+3 : \langle push\ 1, n+4\rangle\\
&n+4 : \langle if\ t_0 = t_0\ goto\ u_2, n+4\rangle\\
&n+5 : \langle if\ empty\ goto\ n+6, n+7\rangle\\
&n+6 : \langle stop, n+6\rangle\\
&n+7 : \langle if\ pop\ goto\ v_2, n+8\rangle\\
&n+8 : \langle if\ t_0 = t_0\ goto\ v_1, n+8\rangle
\end{aligned}$$

Итак, мы построили простую программу со стеком Q, содержащую $n+8$ инструкций и, очевидно, эквивалентную рекурсивной программе P.

Случаи, когда программа P вовсе не содержит инструкций *call*, или содержит ровно одну такую инструкцию, трудностей не представляют. Когда программа P содержит более двух инструкций *call*, следует действовать примерно по такой же схеме, но запоминать номер соответствующей инструкции *call* не одним, а несколькими элементами стека. Это также не приводит к существенному усложнению доказательства. ∎

Докажем теперь обращение леммы 7.8.

Лемма 7.9. *Пусть Ω — некоторая сигнатура, t_0, t_1 — некоторые замкнутые (то есть не содержащие переменных) термы этой сигнатуры.*

Пусть P — простая программа со стеком. Существует рекурсивная программа Q, эквивалентная программе P в том смысле, что для произвольного натурального положительного k, не превосходящего числа переменных программы P, и произвольной такой алгебраической системы $\mathcal{A}$, что $t_0^{\mathcal{A}} \neq t_1^{\mathcal{A}}$, предикаты $P^{\mathcal{A},k}$ и $Q^{\mathcal{A},k}$ совпадают.

Доказательство. В силу леммы 7.7, можно считать, что P содержит не более, чем по одной инструкции каждого из следующих видов:

а) $push$ 0 (запись в стек 0),

б) $push$ 1 (запись в стек 1),

в) $if\ pop\ goto\ i$ (переход по вершине),

г) $if\ empty\ goto\ i$ (переход по пустоте).

Пусть, например, это есть инструкции:

$$\begin{aligned} z_1 :& \langle push\ 0, v_1\rangle \\ z_2 :& \langle push\ 1, v_2\rangle \\ z_3 :& \langle if\ pop\ goto\ u_3, v_3\rangle \\ z_4 :& \langle if\ empty\ goto\ u_4, v_4\rangle \end{aligned}$$

Содержательно, строя программу Q по программе P, мы поставим в Q две инструкции $call$: первую, соответствующую записи в стек 0, и вторую, соответствующую записи в стек 1. Первой будет инструкция $call\ v_1$, второй будет $call\ v_2$.

Инструкциям

$$\langle if\ pop\ goto\ u_3, v_3\rangle,$$

$$\langle if\ empty\ goto\ u_4, v_4\rangle$$

будут соответствовать инструкции $stop$ (в некоторой переменной мы будем запоминать, какая именно инструкция из них сработала). При этом при "возвращении"после $call\ v_1$ мы понимаем, что в вершине стека должен был бы быть 0, а после $call\ v_2$ быть 1. Тут есть еще некоторые тонкости при работе с пустым стеком. Для этого нам необходимо сделать еще один вызов $call$ в самом начале работы программы.

Саму инструкцию *stop* при этом надо обрабатывать некоторым специальным образом. Пусть, в частности,

$$\begin{aligned} z_5 :&\langle stop, v_5\rangle \\ z_6 :&\langle stop, v_6\rangle \\ \ldots&\ldots \\ z_r :&\langle stop, v_r\rangle \end{aligned}$$

будут все инструкции *stop* в программе P, а

$$1 : \langle \alpha_1, w\rangle$$

есть первая инструкция в программе P. Без ограничения общности будем считать, что z_1, $z_2, \ldots, z_r$ отличны от 1.

Так как приводимые ниже построения довольно запутаны, мы будем комментировать некоторые инструкции справа. Эти комментарии, конечно, не являются частями инструкций.

Пусть l — максимальный номер переменной, использованной в программе P. Формально, строя программу Q, заменим в P инструкцию

$$1 : \langle \alpha_1, w\rangle$$

на инструкцию

$$1 : \langle if\ t_0 = t_0\ goto\ n + 14, w\rangle,$$

инструкцию

$$z_1 : \langle push\ 0, v_1\rangle$$

на инструкцию

$$z_1 : \langle if\ t_0 = t_0\ goto\ n + 1, v_1\rangle \quad (\text{переход на } call\ v_1),$$

инструкцию

$$z_2 : \langle push\ 1, v_2\rangle$$

на инструкцию

$$z_2 : \langle if\ t_0 = t_0\ goto\ n + 6, v_2\rangle \quad (\text{переход на } call\ v_2),$$

инструкцию

$$z_3 : \langle if\ pop\ goto\ u_3, v_3\rangle$$

на инструкцию

$$z_3 : \langle if\ t_0 = t_0\ goto\ n + 11, v_3\rangle,$$

а инструкцию

$$z_4 : \langle if\ empty\ goto\ u_4, v_4\rangle$$

на инструкцию

$$z_4 : \langle if\ t_0 = t_0\ goto\ n+13, v_4\rangle.$$

Затем, заменим все инструкции с номерами $z_5, z_6, \ldots, z_r$ (то есть инструкции *stop*) на инструкции

$$z_i : \langle if\ t_0 = t_0\ goto\ n+21, z_i\rangle,$$

где $i = 5, 6, \ldots, r$.

Теперь добавим инструкции

$n+1 : \langle call\ v_1, n+2\rangle$	
$n+2 : \langle if\ x_{l+2} = t_0\ goto\ n+3, n+22\rangle$	Был ли запрос на останов или на переход по стеку?
$n+3 : \langle if\ x_{l+1} = t_0\ goto\ n+4, n+5\rangle$	Был ли запрос на переход по пустоте, или по вершине?
$n+4 : \langle call\ v_4, n+2\rangle$	Был запрос на переход по пустоте, но "стек"непуст.
$n+5 : \langle if\ t_0 = t_0\ goto\ v_3, v_3\rangle$	Был запрос на переход по вершине,а она "содержала"0.
$n+6 : \langle call\ v_2, n+7\rangle$	
$n+7 : \langle if\ x_{l+2} = t_0\ goto\ n+8, n+22\rangle$	Был ли запрос на останов или на переход по стеку?
$n+8 : \langle if\ x_{l+1} = t_0\ goto\ n+9, n+10\rangle$	Был ли запрос на переход по пустоте, или по вершине?
$n+9 : \langle call\ v_4, n+7\rangle$	Был запрос на переход по пустоте, но "стек"не пуст.

$n+10 : \langle if\ t_0 = t_0\ goto\ u_3, u_3\rangle$	Был запрос на переход по вершине,и она "содержала"1.
$n+11 : \langle x_{l+1} := t_1, n+12\rangle$	Отмечаем, что запрос будет по вершине.
$n+12 : \langle stop, n+12\rangle$	"Запрос"к стеку.
$n+13 : \langle x_{l+1} := t_0, n+12\rangle$	Отмечаем, что запрос будет по пустоте.
$n+14 : \langle \alpha_1, n+15\rangle$	Выполнение первой инструкции в программе P.
$n+15 : \langle x_{l+2} := t_0, n+16\rangle$	Отмечаем, что, по умолчанию, будут переходы по стеку, а не остановы.
$n+16 : \langle call\ w, n+17\rangle$	Самый первый вызов в программе, необходимый для работы с "пустым"стеком.
$n+17 : \langle if\ x_{l+2} = t_0\ goto\ n+18, n+22\rangle$	Был ли запрос на останов или на переход по стеку?
$n+18 : \langle if\ x_{l+1} = t_0\ goto\ n+19, n+20\rangle$	Стек был бы пуст. Какой был запрос?
$n+19 : \langle call\ u_4, n+17\rangle$	Был переход по пустоте.
$n+20 : \langle call\ v_3, n+17\rangle$	Был переход по вершине. В обоих случаях, осуществив переход, мы снова осуществляем вызов.
$n+21 : \langle x_{l+2} := t_1, n+22\rangle$	Выполнился действительный *stop*. Зациклим его.
$n+22 : \langle stop, n+22\rangle$	

Этим построение программы Q завершено. Легко видеть, что программы P и Q эквивалентны в смысле формулировки леммы 7.9. ∎

Лемма 7.10. *Пусть* $\Omega = \langle o, f, \dots\rangle$ *— сигнатура, содержащая нульместный функциональный символ* o *(константу), одноместный функциональный символ* f *и, возможно, некоторые другие функциональные* *и предикатные символы, $\mathcal{A}$ — произвольная арифметическая система сигнатуры Ω, σ — некоторая оценка в системе $\mathcal{A}$, m — некоторое натуральное число. Существует формула $\varphi_{\mathcal{A},\sigma,m}$ сигнатуры Ω, описывающая систему $\mathcal{A}$ и состояние σ для m переменных, то есть такая, что*

(1) $\sigma \models \varphi_{\mathcal{A},\sigma,m}$,

(2) для произвольной арифметической системы $\mathcal{B}$ сигнатуры Ω и произвольного состояния σ' в системе $\mathcal{B}$

$$\sigma' \models \varphi_{\mathcal{A},\sigma,m}$$

тогда и только тогда, когда существует такой изоморфизм $\tau : \mathcal{A} \to \mathcal{B}$, что

$$\tau(\sigma(x_i)) = \sigma'(x_i)$$

для всякого $i = 1, 2, \ldots, m$.

Доказательство. Рутинное упражнение. ■

Теперь мы готовы доказать теорему 7.6.

Доказательство теоремы 7.6. В силу следствия 7.5, достаточно показать, что для произвольного m-местного $\mathfrak{A}$-глобального предиката $\mathfrak{Q}$, имеющего сложность $\mathcal{P}$, существует рекурсивная программа P такая, что $\mathfrak{Q} = P[m] : \mathcal{A} \longmapsto P^{\mathcal{A},m}$.

Итак, рассмотрим произвольный m-местный $\mathfrak{A}$-глобальный предикат $\mathfrak{Q}$, имеющий сложность $\mathcal{P}$. По определению это значит, что язык

$$\mathfrak{P} = \{\#(\mathcal{A}) * \#(a_1) * \#(a_2) * \cdots * \#(a_m) \mid$$

$$\mathcal{A} \in \mathfrak{A}; a_1, a_2, \ldots, a_m \in |\mathcal{A}|; \langle a_1, a_2, \ldots, a_m \rangle \in \mathfrak{Q}(\mathcal{A})\}$$

принадлежит $\mathcal{P}$, то есть существует такой 1-МТ распознаватель M, что $t_M(n) \ll n^k$ для некоторого k и $L_M = \mathfrak{P}$. Напомним, что $\ll$ — отношение асимптотической мажорируемости функций.

Тогда, по теореме 6.9, существует такая $log(n) - DAPDA$

$$D = \langle \{0, 1, *, \wedge\}, Q, \pi, q_0 \rangle,$$

что $L_D = L_M = \mathfrak{P}$.

Построим по D такую простую программу со стеком P, что

$$\mathfrak{P} = \{\#(\mathcal{A}) * \#(a_1) * \#(a_2) * \cdots * \#(a_m) \mid$$

$$\mathcal{A} \in \mathfrak{A}; a_1, a_2, \ldots, a_m \in |\mathcal{A}|; \langle a_1, a_2, \ldots, a_m \rangle \in P^{\mathcal{A},m}\}.$$

Дадим сначала неформальные пояснения. Мы будем писать программу P таким образом, чтобы каждому состоянию $q \in Q$ соответствовал свой фрагмент программы. Таким образом, при моделировании работы D мы в каждый момент знаем, в каком состоянии находится моделируемая $DAPDA$ D. Для моделирования нам надо еще знать, каков верхний элемент стека, какой символ обозревается $DAPDA$ на входной ленте, какой на выходной, и какая из рабочих лент является текущей. Заметим, что, согласно замечанию 6.4.1, D можно выбрать таким образом, что головка на входной ленте не выходит за пределы записанного на ней слова более, чем на одну клетку.

В простой программе со стеком у нас прямо есть стек для запоминания 0 и 1. Всю остальную информацию мы будем кодировать при помощи значений некоторого набора переменных. Для простоты будем предполагать, что число элементов в рассматриваемых алгебраических системах не меньше, чем число число рабочих лент D и число символов в ее входном алфавите (то есть $\{0, 1, *, \wedge\}$). Это предположение не приводит к потере общности, так как число алгебраических систем, выпадающих при этом из рассмотрения, конечно (с точностью до изоморфизма), и эти системы могут быть охвачены затем при помощи леммы 5.

По определению, в стеке $DAPDA$ D могут появляться символы $0, 1, *$, а в стеке программы — только 0 и 1. Поэтому мы будем вынуждены при моделировании стека D один элемент кодировать двумя. Договоримся например, что когда D пишет в стек 0 или $*$, мы в программе будем писать в стек соответственно 0 или 1, и затем еще одну 1. Когда же D пишет в стек 1, мы будем писать в стек 00. Соответственно, при чтении мы будем считывать два элемента стека вместо одного.

Фактически при помощи значений переменных мы должны кодировать:

(1) номер позиции на входной ленте;
(2) номер текущей рабочей ленты;
(3) номер позиции на каждой из рабочих лент;
(4) содержимое всех рабочих лент.

Нетрудно видеть, что длина входного слова, то есть слова

$$\#(\mathcal{A}) * \#(a_1) * \#(a_2) * \cdots * \#(a_m)$$

не превосходит $(||\mathcal{A}||)^l - 2$ для некоторого l, другими словами, для кодирования положения головки на входной ленте достаточно l переменных со значениями из $|\mathcal{A}|$.

По предположению, число рабочих лент не превосходит $||\mathcal{A}||$, то есть для кодирования номера текущей рабочей ленты достаточно одной переменной.

По определению $log(n)-DAPDA$, длина каждой из рабочих лент не превосходит, в предыдущих обозначениях, $llog(||\mathcal{A}||)$, следовательно, для кодирования позиции на каждой из рабочих лент достаточно, начиная с достаточно больших алгебраических систем, одной переменной, а для кодирования содержимого каждой из рабочих лент (число возможных записей на каждой из которых равно $4^{llog(||\mathcal{A}||)}$), достаточно l' переменных для некоторой константы l'.

Поясним это. Фактически на всякой рабочей ленте во всякий момент времени записано некоторое "число"в 4-ичной системе счисления: в каждой клетке записано 0, 1, $*$ или $\wedge$. При этом для простоты будем считать, что при записи в 4-ичной системе счисления слова в алфавите $\{\wedge,1,0,*\}$ $\wedge$ соответствует цифре 0, 1 соответствует цифре 1, 0 соответствует цифре 2 и $*$ соответствует цифре 3. При кодировании этого слова при помощи l' переменных, мы могли бы просто перевести это число в $||\mathcal{A}||$-ичную систему счисления. Произвольный элемент

$$a\in|\mathcal{A}|=\{0,1,2,\ldots,||\mathcal{A}||\}$$

представляет собой соответствующую цифру в $||\mathcal{A}||$-ичной системе счисления.

Для технического удобства мы будем поступать при кодировании рабочих лент не совсем так. А именно, пусть

$$\theta_1\theta_2\ldots\theta_{j-1}$$

является словом на данной рабочей ленте слева от головки,

$$\theta_{j+1}\theta_{j+2}\ldots\theta_{j+u}$$

является словом на данной рабочей ленте справа от головки, а θ_j — обозреваемый головкой на данной рабочей ленте символ. Здесь

$$\theta_1,\theta_2,\ldots,\theta_{j+u}\in\{0,1,*,\wedge\}.$$

Будем кодировать содержимое данной рабочей ленты при помощи $2l'+1$ переменной таким образом. При помощи первых l' переменных закодируем слово

$$\theta_1\theta_2\ldots\theta_{j-1},$$

рассматриваемое как число в 4-ичной системе счисления, числом в $||\mathcal{A}||$-ичной системе счисления. Затем при помощи других l' переменных закодируем слово

$$\theta_{j+u}\theta_{j+u-1}\dots\theta_{j+1}$$

(то есть взятое в обратном порядке) аналогичным образом. Наконец, в последней переменной запомним сам символ θ_j.

Мы, конечно, не можем при помощи значений переменных кодировать входную ленту D. Однако, зная позицию на входной ленте и зная, что при работе на $\mathcal{A}$, $\langle a_1, a_2, \dots, a_m\rangle$ на входной ленте записано в точности слово

$$\#(\mathcal{A}) * \#(a_1) * \#(a_2) * \dots * \#(a_m),$$

мы, при помощи предикатов и функций системы $\mathcal{A}$ можем определить, какой символ записан в искомой клетке входной ленты.

Сами символы алфавита $\{0, 1, *, \wedge\}$ мы также будем по мере необходимости запоминать в переменных, при этом 1 кодировать 1, 0 кодировать 2, $*$ кодировать 3, $\wedge$ кодировать 0 (мы договорились, что $||\mathcal{A}|| \geq 4$).

Вполне формальное построение P слишком громоздко. Рассмотрим поэтому только принципиальные моменты.

В программе P будут использованы переменные

$$x_1, x_2, \dots, x_m$$

для получения доступа к $a_1, a_2, \dots, a_m$, и кроме того, еще некоторые переменные

$$x_{m+1}, x_{m+2}, \dots, x_{m+v}.$$

Точное число этих переменных зависит от числа и местности сигнатурных функций и предикатов, от числа m, от числа рабочих лент D.

Сама программа P будет состоять из начальной инициализации и из $r+1$ частей, соответствующих состояниям q_0, $q_1, \dots, q_r$, где $Q = \{q_0, q_1, \dots, q_r\}$.

Начальная инициализация состоит в приписывании значений всем переменным программы, кроме x_1, $x_2, \dots, x_m$.

Так, если в программе переменные $x_{e+1}, x_{e+2}, \dots, x_{e+l}$ отведены для запоминания положения головки D на входной ленте, мы "запишем"в них $||\mathcal{A}||$-ичное число 1, то есть выполним в программе следующую последовательность присваиваний:

$$x_{e+1} := o, x_{e+2} := o, \dots, x_{e+l} := f(o).$$

Это соответствует тому, что в начальный момент времени головка на входной ленте стоит в крайней левой позиции.

Аналогично, если x_j отведена для запоминания положения головки на некоторой рабочей ленте, то мы запишем в нее также $||\mathcal{A}||$-ичную 1, то есть выполним в программе присваивание $x_j := f(o)$.

Так же, если x_t отведена для запоминания номера текущей рабочей ленты, то, поскольку в начальный момент времени текущей рабочей лентой должна быть первая, мы запишем в нее также $||\mathcal{A}||$-ичную 1, то есть выполним в программе присваивание $x_t := f(o)$.

Далее, если в программе переменные

$$x_{j+1}, x_{j+2}, \ldots, x_{j+2l'+1}$$

отведены для запоминания некоторой рабочей ленты, мы "запишем"в них $||\mathcal{A}||$-ичное число 0, то есть выполним в программе следующую последовательность присваиваний:

$$x_{j+1} := o, x_{j+2} := o, \ldots, x_{j+2l'+1} := o.$$

Это соответствует тому, что в начальный момент времени на всех рабочих лентах во всех позициях стоят $\wedge$.

На этом начальная инициализация заканчивается и работа программы продолжается с фрагмента, соответствующего состоянию q_0.

Напомним, что работа D происходит согласно определению 6.4.1.

Итак, для моделирования очередного такта работы D нам надо установить, какой символ обозревается на входной ленте, какой — на текущей рабочей, каков верхний элемент стека (если стек не пуст).

Рассмотрим поочередно, как получить эту информацию.

1. **Вычисление символа, обозреваемого головкой на входной ленте.**

По условию, в некоторых переменных, скажем, в

$$x_{e+1}, x_{e+2}, \ldots, x_{e+l}, \tag{43}$$

запомнено положение головки на входной ленте в $||\mathcal{A}||$-ичной системе счисления. Возьмем еще один комплект переменных

$$x'_{e+1}, x'_{e+2}, \ldots, x'_{e+l} \tag{44}$$

и выполним последовательность присваиваний

$$x'_{e+1} := o, x'_{e+2} := o, \ldots, x'_{e+l} := f(o).$$

Другими словами, мы записали в них $||\mathcal{A}||$-ичную 1. Это соответствует положению головки на входной ленте, в котором обозревается первая $*$ в слове

$$\#(\mathcal{A}) * \#(a_1) * \#(a_2) * \cdots * \#(a_m).$$

Если при этом переменные (43) и (44) попарно равны, то мы достигли нужного результата и можем присвоить переменной, соответствующей символу на входной ленте, значение $f(f(f(o)))$ (то есть 3), равное коду $*$. Если же $x_{e+1} = x_{e+2} = \cdots = x_{e+l} = o$, то на входной ленте обозревается символ $\wedge$, и тогда присвоим соответствующей переменной значение o — код $\wedge$.

Если же (43) и (44) попарно не равны, и $x_{e+1}, x_{e+2}, \ldots, x_{e+l}$ не все равны 0, то мы должны моделировать движение по входной ленте вправо, то есть входить в зону, где записан код функционального или предикатного символа (мы знаем, какого, так как все они упорядочены фиксированным способом).

Само движение вправо моделируется переприсваиванием на переменных (44). Грубо говоря, мы хотим увеличить записанное в них $||\mathcal{A}||$-ичное число на 1 и мы оставляем написание этой части программы читателю.

Рассмотрим для определенности ситуацию, когда мы обозреваем начало кода некоторого j-местного предиката R (случай с функциями рассматривается аналогично).

Мы возьмем новые переменные

$$x_1'', x_2'', \ldots, x_j'' \tag{45}$$

и будем в них перебирать в лексикографическом порядке j-местные кортежи элементов $|\mathcal{A}|$ параллельно с движением по ленте вправо, начиная с кортежа $\langle o, o, \ldots o\rangle$. Если в какой то момент все переменные (45) имеют своим значением последний элемент $|\mathcal{A}|$ (это проверяется формулой $f(x') = x'$), то это значит, что мы закончили просмотр кода данного предикатного символа, и тогда мы перейдем к следующему или же к части

$$*\#(a_1) * \#(a_2) * \cdots * \#(a_m). \tag{46}$$

Напомним, что в конце кода каждого предиката и каждой функции также стоят $*$, что надо учитывать.

Если же в какой то момент переменные (43) и (44) оказываются попарно равными, то мы достигли нужного результата. При этом переменные (45) у нас имеют значением некоторый кортеж. Можно присвоить

переменной, соответствующей символу на входной ленте, значение 1, если
$$\langle x_1'', x_2'', \ldots, x_j'' \rangle \in R,$$
и 2 в противном случае.

Наконец, если мы движемся уже в части (46), то значение нам известно (по условию, значение a_i хранится в переменной x_i, и остается вычислить соответствующую цифру записи заданного числа в двоичной системе счисления).

2. **Вычисление символа, обозреваемого головкой на текущей рабочей ленте.**

Нам известно, что текущая рабочая лента закодирована переменными, скажем,
$$x_{e+1}, x_{e+2}, \ldots, x_{e+2l'+1} \tag{47}$$
(см. выше). Тогда код обозреваемого на ней символа хранится в переменной $x_{e+2l'+1}$.

3. **Работа со стеком** производится тривиально.

Остается еще промоделировать в программе один такт работы машины. Единственный нетривиальный шаг здесь — работа с текущей рабочей лентой.

Пусть текущая рабочая лента закодирована переменными, скажем, (47). Тогда код обозреваемого на ней символа хранится в переменной $x_{e+2l'+1}$. Присвоим переменной $x_{e+2l'+1}$ код символа, который записывается на эту клетку рабочей ленты по определению машины. Теперь осталось сдвинуть головку на этой ленте.

Пусть головка, согласно определению машины, должна двигаться вправо (и еще не находится на последней клетке ленты справа — это проверяется по переменной, кодирующей положение головки на данной ленте тривиально). Напомним, что в переменных
$$x_{e+1}, x_{e+2}, \ldots, x_{e+l'} \tag{48}$$
храниться код слова, записанного на этой ленте слева от головки, в переменных
$$x_{e+l'+1}, x_{e+l'+2}, \ldots, x_{e+2l'} \tag{49}$$
хранится код инверсии слова, записанного справа от головки.

Пусть i, j, k будут $||\mathcal{A}||$-ичные числа, записанные в переменных (48), в переменных (49), и в переменной $x_{e+2l'+1}$, соответственно. Запишем в переменные (48), в переменные (49) и в переменной $x_{e+2l'+1}$ $||\mathcal{A}||$-ичные числа $4i+k$, целую часть числа, полученного от деления j на 4 и остаток от деления j на 4. Все эти три числа легко вычислимы в программе. Очевидно теперь, что в переменной $x_{e+2l'+1}$ хранится код символа, который был записан на ленте справа от головки. Соответственно в переменных (48) и в переменных (49) хранятся код слова, записанного слева от головки с приписанным к нему справа обозреваемым символом, и код взятого в обратном порядке слова, записанного на ленте справа от головки без первого символа в нем.

Итак, увеличив на 1 значение переменной, кодирующей положение головки на данной рабочей ленте, мы закончим моделирование движения на рабочей ленте на одну клетку вправо.

Детали построения оставляем читателю. ∎

Упражнения

Упражнение 7.4.1 (*). *Покажите, что*

а). Любой задаваемый формулой логики предикатов глобальный предикат имеет сложность $\mathcal{P}$.

б). Для любого натурального m существует формула логики предикатов φ такая, что задаваемый ей глобальный предикат $\mathcal{A} \longmapsto \varphi^{\mathcal{A}}$ не имеет сложность $DTIME(n^m)$ (то есть имеет некоторую большую сложность).

Указание: рассмотрите формулы с $(m+1)$ квантором.

Упражнение 7.4.2. *Докажите лемму 7.8 (то есть постройте программу Q) в общем случае.*

Упражнение 7.4.3. *Докажите лемму 7.10.*

Упражнение 7.4.4 (**). *Нами дано описание $\mathcal{P}$-полного класса глобальных предикатов для арифметических систем как предикатов, задаваемых рекурсивными программами. Покажите, что в этом же смысле класс глобальных предикатов, задаваемых простыми программами, $DLOGSPACE$-полон.*

Упражнение 7.4.5 (***). *В том же смысле, приведите описание*

$$NLOGSPACE,\ DLINEARSPACE,\ NP,\ EXPTIME\text{-}$$

полных классов глобальных предикатов.

Предметный указатель

X-код, 337

автоматная грамматика, 73
аксиома ИВ, 105
аксиомы ИП, 142
алфавит, 11
арифметическая система, 343
асимптотически мажорируется, 275
асимптотической мажорантой, 275
ассоциативное исчисление, 237
атомная формула, 128

бескванторная формула, 154
булевы эквивалентности, 114

Высказывания, 100

верхняя оценка детерминированной (недетерминированной) временной (емкостной) сложности, 275
временная сложность, 274
вставимые нетерминалы, 58
вхождение высказывания в формулу, 101
вывод в КС-грамматике, 36
высота грамматики, 58
высота дерева, 96
высота символа, 58
высота языка, 58

глобальный предикат, 323
гомоморфизм графов, 66
грамматика без пустых правил, 44
грамматика с сильным самовставлением, 58
грамматикой с самовставлением, 58
граф, 62

дерево, 93
дерево вывода, 94
дерево доказательства, 107
деревом вывода слова, 95
детерминированный конечный автомат, 21
дизъюнктивная нормальная форма, 117
днф, 117
доказательство, 106, 143
дополнительные схемы правил вывода, 106

ёмкостная сложность, 274

замкнутость семейства языков, 29
запрос к базе данных, 323
значение терма на состоянии, 127
значение формулы, 102
значение формулы на состоянии, 130

индекс грамматики, 61
истинность секвенции, 104

Конкатенацией, 13

классы временной сложности, 285
классы емкостной сложности, 285
кнф, 117
код алгебраической системы, 337
код натурального числа, 336
код операции, 336

код предиката, 336
конечный автомат, 20
контекстно-свободная грамматика, 36
конфигурация 1-НМТ, 273
конфигурация МТ, 203
конъюнктивная нормальная форма, 117
критерий доказуемости кнф, 118
критерий доказуемости секвенции, 109
критерий доказуемости эд, 119
кс-грамматика, 36

леворекурсивная грамматика, 52
леворекурсивный символ, 52
лемма о замене в ИВ, 113
лемма о разрастании, 33
лемма об эквивалентности формул, 113
лента МТ, 203
линейная программа, 331

МП-автомат, 78

маркированной точкой, 309
машина Тьюринга, 200
машины Тьюринга с одной рабочей лентой, 270
минимизация, 179
модель множества формул, 158

Недетерминированная одноленточная машина машина Тьюринга, 272

непосредственная эквивалентность в ассоциативном исчислении, 237
непосредственное следствие слова в полусистеме Туэ, 232
нижняя оценка детерминированной (недетерминированной) временной (емкостной) сложности, 275
нормальная форма Грейбах, 55
нормальная форма Хомского, 44
нумерал, 250

объединение, 14
ограниченное перемножение, 181
ограниченное суммирование, 181
ответ на запрос, 323

Подсловом, 14

пересечения, 14
плотные классы сложности, 288
позиция в игре, 62
полнота множества формул, 158
полный класс глобальных предикатов, 342
полуканонический МП-автомат, 83
полусистема Туэ, 232
порядок вершин дерева, 94
правила вывода ИВ, 106
правила вывода ИП, 143
праволинейная грамматика, 73
предваренная формула, 154
префикса, 14
приведенная грамматика, 48
примитивная рекурсия, 178
присваивание, 171, 326
проблема соответствий Поста, 88
программа с метками, 188
простая программа, 326
простая программа со стеком, 343
простейшая функция, 180
противоречивое множества формул, 158
псевдоязык грамматики, 38

путевая грамматика, 332
путь в дереве, 96

работа конечного автомата, 21
работа МП-автомата, 79
работа МТ на СП, 201
работа МТ на ленте, 203
работа программы на состоянии, 172
работа программы с метками, 188
равенство термов, 128
разбор случаев, 182
разности, 14
распознавателем, 272, 274
распознается с детерминированной (недетерминированной) временной сложностью, 275
распознается с детерминированной (недетерминированной) емкостной сложностью, 275
регулярное выражение, 16
регулярность языка, 16
результат подстановки термов вместо переменных, 133
реконструкцией, 309
рекурсивная программа, 329

Сверхслово, 12
Словом, 11

самовставимый символ, 57
сводимые нетерминалы, 58
связь булевых функций и формул, 102
секвенциями, 103
сильная мажоранта, 276
сильно самовставимый символ, 58
символ секвенции, 103
следствие из множества формул, 158
следствие слова в полусистеме Туэ, 232
сложность глобального предиката, 341
сложность формулы логики предикатов, 130
соответствие Поста, 235
состояние, 127
состояние базы данных, 321
стековая машина, 301
структурированная программа, 171
суперпозиция арифметических функций, 177
суффиксом, 14
схема базы данных, 321
схема правил вывода от противного, 105
схемы главных правил вывода ИВ, 105
схемы правил вывода ИП, 142

тавтология, 103
теорема о доказуемости истинных секвенций, 120
теорема о кванторах, 143
теорема о композиции деревьев доказательства, 108
теорема о контекстно-свободности регулярных языков, 42
теорема о нормальной форме Грейбах, 55
теорема о нормальной форме Хомского, 47
теорема о полноте исчисления предикатов, 169
теорема о равенстве, 146
теорема о разрастании для КС-

языков, 96
теорема о совпадении класса регулярных языков и класса языков, задаваемых конечными автоматами, 73
теорема о существовании модели, 160
теорема о существовании днф, 118
теорема о существовании кнф, 117
теорема о существовании приведенной грамматики, 48
теорема о формальной индукции, 245
теорема об истинности доказуемых секвенций, 110
терм, 127
тест, 171
трасса, 68

универсальный язык относительно сводимости, 289
условие, 171

ФА, 244

формальная арифметика, 244
формула логики высказываний, 101
формула логики предикатов, 128
функциональная сложность формулы, 101
функция, вычислимая программой, 174

характеристическая функция, 175

частично рекурсивная функция, 180

эд, 117
эк, 117
эквивалентность автоматов, 22
эквивалентность в ассоциативном исчислении, 237
эквивалентность формул, 112, 150
эквивалентными, 272
элементарная дизъюнкция, 117
элементарная конъюнкция, 117

язык, 12
язык грамматики, 38
язык конечного автомата, 21