Глава 14

Сведения из теории алгоритмов

В ДЛ делается особенный упор на то, чтобы логика была разрешима, по возможности за «разумное» время с «разумным» расходом памяти. Поэтому большинство результатов в ДЛ говорят о разрешимости или неразрешимости той или иной логики, а также об ее сложности. В этом разделе мы введем необходимые понятия и приведем известные факты.

14.1 Разрешимые и неразрешимые проблемы

Определение разрешимой и неразрешимой проблемы. Сведение проблем друг к другу.

14.2 Неразрешимость: проблема домино

Для доказательства неразрешимость какой-либо логики $\mathcal{L}$ (то есть для доказательства неразрешимости проблемы $\mathcal{P}$ выполнимости концептов в логике $\mathcal{L}$) обычно поступают следующим образом. Берут некоторую проблему $\mathcal{P}_0$, неразрешимость которой уже известна, и сводят ее к проблеме $\mathcal{P}$. Под *сведéнием* здесь подразумевается алгоритм, который по исходным данным проблемы $\mathcal{P}_0$ строит исходные данные для проблемы $\mathcal{P}$. Обратите внимание, что сведение должно быть *эффективным*, то есть требуется не просто доказать, что для каждого x (входные данные для $\mathcal{P}_0$) существует y (входные данные для $\mathcal{P}$), а что y строится по x некоторым алгоритмом.

Ясно, что этого достаточно для доказательства неразрешимости проблемы $\mathcal{P}$. Действительно, если бы $\mathcal{P}$ была разрешимой, то есть существовал алгоритм (разрешающая процедура для $\mathcal{P}$), который по каждому y выдает «да» и «нет» и тем самым решает проблему $\mathcal{P}$, то проблема тоже оказалась бы разрешимой: по каждому x мы алгоритмом строим y, а к нему применяем разрешающую процедуру для $\mathcal{P}$.

Возникает вопрос, где взять заведомо неразрешимую проблему $\mathcal{P}_0$. В качестве такой «классической» неразрешимой проблемы в логике часто выбирают так называемую (неограниченную) проблему домино, доказательство неразрешимости которой можно найти в [2].

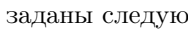
Определение 14.1 (Проблема домино)**.** *Системой домино* называется тройка $\mathcal{D} = (D, H, V)$, где $D = \{d_1, \ldots, d_n\}$ — конечное множество (набор «типов» клеток), $H, V \subseteq D \times D$ — произвольные двуместные отношения (отношения «соседства» клеток). *Покрытием* плоскости $\mathbb{N}\times\mathbb{N}$ системой домино $\mathcal{D}$ называется всюду определенная функция $t\colon \mathbb{N}\times\mathbb{N} \to D$, такая что выполнены следующие условия «склейки» клеток:

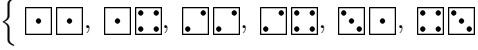
- $\forall x, y \in \mathbb{N} \quad \langle t(x,y), t(x+1,y)\rangle \in H$,
- $\forall x, y \in \mathbb{N} \quad \langle t(x,y), t(x,y+1)\rangle \in V$.

Проблема домино: по системе домино $\mathcal{D}$ узнать, существует ли покрытие плоскости $\mathbb{N}\times\mathbb{N}$ системой $\mathcal{D}$.

Теорема 14.1 ([2])**.** *Проблема домино неразрешима.*

Заметим попутно, что проблема покрытия системой домино плоскости $\mathbb{Z}\times\mathbb{Z}$ тоже неразрешима [2].

Пример 14.1. Приведем пример системы домино. Пусть $D = \{$  $\}$, а отношения соседства $H, V \subseteq D \times D$ заданы следующим образом:

$H =$  , $\quad V =$.

Покрыть этой системой плоскость $\mathbb{N}\times\mathbb{N}$ — значит расположить на клетках $\mathbb{N}\times\mathbb{N}$ элементы множества D (точнее, неограниченное количество их копий) так, чтобы элементы, соседствующие по горизонтали (и аналогично, по вертикали), стыковались согласно отношению H (соответственно V). В общем случае, конечно, система домино может состоять не из 6, а из произвольного числа элементов.

14.3 Основные понятия теории сложности вычислений

Теория алгоритмов делится на два раздела: **теория вычислимости** и **теория сложности вычислений**. Первая изучает вопрос о принципиальной возможности построить алгоритм для решения той или иной проблемы. Вторая посвящена изучению того, сколько ресурсов (времени или памяти) требуется наилучшему алгоритму для решения конкретного класса задач.

Пусть машина Тьюринга M принимает на вход слова x в некотором конечном алфавите Σ (будем считать, что алфавит Σ содержит не менее двух символов). Множество всех (конечных) слов в алфавите Σ будем обозначать Σ^*. Начав работу на слове x, машина M может остановиться через T шагов (или не остановиться вовсе) и использовать (то есть побывать) S ячеек ленты. В этом случае мы пишем:

$$\textsc{Time}_M(x) = T \quad \text{и} \quad \textsc{Space}_M(x) = S, \qquad \text{где } T, S \in \mathbb{N} \cup \{\infty\}.$$

Для всякого натурального числа n рассмотрим максимум этих величин по всем словам длины n:

$$\textsc{Time}_M(n) := \max_{|x|=n} \textsc{Time}_M(x) \quad \text{и} \quad \textsc{Space}_M(n) := \max_{|x|=n} \textsc{Space}_M(x).$$

Это уже функции из $\mathbb{N}$ в $\mathbb{N}\cup\{\infty\}$; более того, мы будем далее рассматривать лишь машины Тьюринга, останавливающиеся на всех словах (то есть тотальные), так что данные функции будут из $\mathbb{N}$ в $\mathbb{N}$.

Некоторым стандартным образом можно ввести понятие: машина Тьюринга *допускает* слово x (будем обозначать это как $M(x) = \text{Да}$). Например, можно договориться, что машина M допускает слово x, если, начав работу на слове x, она останавливается через конечное число шагов в завершающем состоянии и обозревает ячейку с цифрой 1, при этом все остальные ячейки ленты пусты. Аналогично, машина M *отвергает* слово x (пишем: $M(x) = \text{Нет}$), если происходит всё то же самое, но с цифрой 0.

14.3.1 Детерминированные классы сложности

Языком будем называть произвольное множество слов $L \subseteq \Sigma^*$.

Язык, распознаваемый машиной Тьюринга M, определяется как $\mathbb{L}(M) = \{\, x \in \Sigma^* \mid M(x) = \text{Да} \,\}$.

Класс P (он же PTime) определяется как класс языков, распознаваемых машинами Тьюринга за полиномиальное время. Формально, язык L лежит в классе P, если существует машина Тьюринга M, распознающая данный язык L, и такая что $\textsc{Time}_M(n)$ ограничена сверху некоторым полиномом $p(n)$:

$$L \in \mathrm{P} \iff \exists M\colon\ L = \mathbb{L}(M) \wedge \exists p\, \forall n \in \mathbb{N}\colon\ \textsc{Time}_M(n) \leqslant p(n). \qquad (*)$$

В частности, машина M останавливается на всех входных словах, а на словах из L выдает «Да». Без ограничения общности можно считать, что на остальных словах она выдает «Нет».

Беря в этом определении вместо $p(n)$ функции $2^{p(n)}$, $2^{2^{p(n)}}$ и т.д., получаем определение языка класса ExpTime, ExpExpTime, ... — то есть языка, распознаваемого за экспоненциальное, двойное экспоненциальное и т.д. время.

Если в $(*)$ ограничивать не время, а память: $\textsc{Space}_M(n) \leqslant p(n)$, то получим определение языка класса PSpace — то есть языка, распознаваемого на полиномиальной памяти. Если вместо $p(n)$ брать $2^{p(n)}$, $2^{2^{p(n)}}$ и т.д., то получим определение языка класса ExpSpace, ExpExpSpace и т.д.

В общем случае, когда в $(*)$ используется функция $f(n)$, получается определение класса $\textsc{Time}(f)$; аналогично для $\textsc{Space}(f)$.

Пример 14.2. Множество простых чисел (записанных в двоичной системе счисления) является языком класса ExpTime. Множества концептов логики $\mathcal{ALC}$, формул логики высказываний, формул логики первого порядка и т.д., формул модальной логики **K** являются языками класса PTime.

14.3.2 Недетерминированные классы сложности

Все данные выше определения основаны на понятии обычной (или *детерминированной*) машины Тьюринга, то есть в программе которой для каждого состояния q и каждой обозреваемой на ленте буквы a существует ровно одна команда вида $qa \to \ldots$ Если разрешить любое количество (0 и более) команд

такого вида, то получится понятие *недетерминированной* машины Тьюринга. На каждом шаге вычислений у такой машины есть «выбор», какую из подходящих команд выполнить следующей. При этом возникает дерево возможных вычислений на данном входном слове x: некоторые из вычислений могут останавливаться через конечное число шагов, другие — работать неограниченно долго.

Говорят, что недетерминированная машина Тьюринга N *допускает* слово x (пишем: $N(x) = \text{Да}$), если существует ее вычисление на входном слове x, приводящее к допускающей конфигурации (в которой машина обозревает 1, находясь в завершающем состоянии). Язык, распознаваемый недетерминированной машиной N, обозначается $\mathbb{L}(N)$ и определяется как выше.

Время работы недетерминированной машины N на входном слове x обозначается $\textsc{Time}_N(x)$ и определяется как *максимум* количества шагов работы машины до остановки (перехода в завершающее состояние), где максимум берется по всем возможным вычислениям, начавшимся на слове x. Обратите внимание, что $\textsc{Time}_N(x)$ определяется как максимальная длина ветви в дереве возможных вычислений, то есть как глубина этого дерева. Использованная машиной N память на входном слове x обозначается $\textsc{Space}_N(x)$ определяется аналогично.

Далее функции $\textsc{Time}_N(n)$ и $\textsc{Space}_N(n)$ определяются как и ранее.

Используя недетерминированные машины вместо детерминированных в определении вида $(*)$, получаем определения соответствующих *недетерминированных классов сложности*. Например, определим важнейший класс NP: язык L лежит в классе NP, если существует недетерминированная машина N, распознающая этот язык L и делающая это за полиномиальное время:

$$L \in \text{NP} \iff \exists N\colon\ L = \mathbb{L}(N) \wedge \exists p\, \forall n \in \mathbb{N}\colon\ \textsc{Time}_N(n) \leqslant p(n). \qquad (**)$$

Заменяя $p(n)$ на $2^{p(n)}$, $2^{2^{p(n)}}$, ..., получаем определение классов NExpTime, NExpExpTime, ... Беря вместо времени Time память Space, получаем определение классов NPSpace, NExpSpace, ...

В общем случае, когда в $(**)$ используется функция $f(n)$, получается определение класса $\text{NTime}(f)$; аналогично для $\text{NSpace}(f)$.

Определение через «угадывание» сертификата

Недетерминированные классы сложности можно определить эквивалентным образом, не прибегая к недетерминированным машинам Тьюринга. Дадим, для примера, такое определение класса NP:

Пусть Σ_1 — ещё один алфавит (быть может, совпадающий с Σ, быть может, отличный), содержащий, как и Σ, не менее двух символов.

Язык $L \subseteq \Sigma^*$ принадлежит классу NP, если существует двуместный предикат на словах $R \subseteq \Sigma^* \times \Sigma_1^*$, такой что выполнены два условия:

1) предикат $R(x, y)$ распознается за полиномиальное время,[1]

2) $\exists$ полином p, такой что $\forall x \in \Sigma^*\colon\ x \in L \iff \exists y \in \Sigma_1^*\colon\ |y| \leqslant p(|x|)\ \&\ R(x, y)$.

Таким образом, чтобы проверить принадлежность слова x языку L, нам нужно найти («угадать») слово y полиномиальной длины (оно называется «подсказкой» или «сертификатом» принадлежности слова x языку L) и затем проверить условие $R(x, y)$ за полиномиальное время. Несложно видеть, что новое определение эквивалентно прежнему. Так, имея недетерминированную машину N, распознающую язык L, и имея входное слово x, в качестве «подсказки» y можно взять последовательность «номеров» тех ветвей, по которым должно идти вычисление N в каждой точке ветвления, чтобы в итоге допустить слово x.

Пример 14.3. Множество составных чисел (записанных в двоичной системе) — язык класса NP.

Недавно (2009 г.) было доказано, что множество простых (а значит и составных) чисел принадлежит классу P. Это значит, что существует полиномиальный алгоритм проверки простоты числа.

В то же время, для задачи *нахождения* простых делителей натурального числа, (которая, очевидно, является задачей класса NP) до сих пор не известно, существует ли полиномиальный алгоритм; с другой стороны, не известно, является ли эта задача NP-полной. Именно на том, что не известен полиномиальный алгоритм решения этой задачи, основаны многие схемы криптографии.

14.3.3 Иерархия классов сложности

Часто требуется от проблемы переходить к ее дополнению. Если K — класс языков, то:

$$\text{co-K} = \{L \subseteq \Sigma^* \mid \overline{L} \in \text{K}\}, \quad \text{где } \overline{L} = \Sigma^* \setminus L.$$

Следующие утверждения легко следуют из определений классов.

[1] То есть существует детерминированная машина Тьюринга M, работающая на всех входах $\langle x, y\rangle$ полиномиальное время и такая что M принимает пару слов $\langle x, y\rangle \iff R(x, y)$ истинно.

Лемма 14.2. *Имеют место следующие соотношения между классами сложности:*
1) Если K — детерминированный класс сложности, то $K \subseteq NK$.
2) $\textsc{Time}(f) \subseteq \textsc{Space}(f)$ *и* $\textsc{NTime}(f) \subseteq \textsc{Space}(f)$ *для всякой функции f.*
3) $\textsc{Space}(f) \subseteq \textsc{Time}(2^{p(f)})$ *для некоторого полинома $p(n)$.*
4) Если K — детерминированный класс сложности, то $\text{co-K} = \text{K}$.

Следующая теорема является важной в теории сложности вычислений:

Теорема 14.3 (Savitch)**.** $\textsc{NSpace}(f) \subseteq \textsc{Space}(f^2)$. *Как следствие:*
$\textsc{NPSpace} = \textsc{PSpace}$, $\textsc{NExpSpace} = \textsc{ExpSpace}$ *и т.д. для любых классов по памяти.*

Из приведенных фактов вытекает следующая цепочка включений:

$$\begin{array}{lllllll} \text{P} & \subseteq & \text{NP} & \subseteq & \textsc{PSpace} & \subseteq & \\ \textsc{ExpTime} & \subseteq & \textsc{NExpTime} & \subseteq & \textsc{ExpSpace} & \subseteq & \\ \textsc{ExpExpTime} & \subseteq & \textsc{NExpExpTime} & \subseteq & \textsc{ExpExpSpace} & \subseteq & \ldots \end{array}$$

Кроме того, $\text{P} \subseteq \text{co-NP} \subseteq \textsc{PSpace}$ и аналогично в остальных строках.
Известно, что по столбцам включения строгие:

$$\begin{array}{l} \text{P} \subset \textsc{ExpTime} \subset \textsc{ExpExpTime} \\ \text{NP} \subset \textsc{NExpTime} \subset \textsc{NExpExpTime} \\ \textsc{PSpace} \subset \textsc{ExpSpace} \subset \textsc{ExpExpSpace} \end{array}$$

Строгость остальных включений неизвестна. Так, открыта следующая знаменитая проблема: $\boxed{\text{P} \neq \text{NP?}}$

14.3.4 Полиномиальная сводимость

Введем способ сравнения сложности двух языков $L_1 \subseteq \Sigma_1^*$ и $L_2 \subseteq \Sigma_2^*$. Язык L_1 *полиномиально сводится* к языку L_2 (пишем: $L_1 \leqslant_p L_2$), если существует всюду определенная функция $f\colon \Sigma_1^* \to \Sigma_2^*$, такая что:
1) для любого слова $x \in \Sigma_1^*$ имеет место эквивалентность: $x \in L_1 \Leftrightarrow f(x) \in L_2$;
2) функция f вычислима за полиномиальное время.

Лемма.
а) Если $L_1 \leqslant_p L_2$ и $L_2 \leqslant_p L_3$, то $L_1 \leqslant_p L_3$.
б) Если $L_1 \leqslant_p L_2$, то $\overline{L_1} \leqslant_p \overline{L_2}$.
в) Если $L_1 \leqslant_p L_2 \in \text{P}$, то $L_1 \in \text{P}$ (то есть класс P является минимальным относительно $\leqslant_p$).

Естественным является желание найти в каком-либо классе K «самый сложный» язык. Априори не очевидно, что такие языки вообще могут существовать. Тем не менее, во всех вышеупомянутых классах самые сложные языки существуют. Более того, практически все встречающиеся логики являются «самыми сложными» в некотором классе. Формализуем это понятие.
Язык L называется K-*трудным*, если любой язык $L' \in \text{K}$ сводится к L, то есть $L' \leqslant_p L$.
Язык L называется K-*полным*, если $L \in \text{K}$ и L является K-трудным.

14.4 Сложность важнейших логик

Известны следующие факты о вычислительной сложности логик:

- классическая логика высказываний — NP-полна;
- интуиционистская логика высказываний — PSpace-полна,
- логика предикатов — неразрешима; детальный анализ показывает, что логика предикатов:
  - с одной переменной — NP-полна;
  - с двумя переменными — NExpTime-полна;
  - с тремя переменными (даже без равенства, констант и функций) — неразрешима;
- интуиционистская логика предикатов — неразрешима; анализ показывает, что эта логика:
  - с одной переменной — разрешима (сложность?);
  - с двумя переменными (даже без равенства, констант и функций) — неразрешима;
- модальная логика $\mathbf{K}_n$ и даже $\mathbf{K}$ — PSpace-полна;
  точнее: проблема (локальной) выполнимости формул в логике $\mathbf{K}_n$ — PSpace-полна;
- проблема глобальной глобальной выполнимости формул в логике $\mathbf{K}_n$ — ExpTime-полна;
- пропозициональная динамическая логика **PDL** — ExpTime-полна.

В данном курсе доказывается, что:

- дескрипционная логика $\mathcal{ALC}$ — PSPACE-полна,
  точнее: проблема выполнимости концептов в логике $\mathcal{ALC}$ — PSPACE-полна;
- дескрипционная логика $\mathcal{ALC} + \mathsf{TBox}$ — EXPTIME-полна,
  точнее: проблема выполнимости концептов отн. терминологий в логике $\mathcal{ALC}$ — EXPTIME-полна.