Конспект по теории сложности
III семестр
Современное программирование, факультет математики и компьютерных наук, СПбГУ
(лекции Гирша Эдуарда Алексеевича)

Тамарин Вячеслав

December 26, 2020

Contents

Chapter 1

Введение в теорию сложности вычислений

Лекция 1: †

5 nov

1.0.1 Напоминания

Обсудим, что мы решаем.

Обозначение.

- Алфавит будет бинарный $\{0,1\}$;
- Множество всех слов длины n : $\{0,1\}^n$;
- Множество всех слов конечной длины $\{0,1\}^*$;
- Длина слова x: $|x|$.

Определение 1

Язык (задача распознавания, decision problem) — $L \subseteq \{0,1\}^*$.

Индивидуальная задача — пара, первым элементом которой является условие, а второй – решение; принадлежит $\{0,1\}^* \times \{0,1\}^*$.

Массовая задача — некоторое множество индивидуальных задач, то есть бинарное отношение на $\{0,1\}^*$.

Определение 2

Будем говорить, что алгоритм **решает задачу поиска** для массовой задачи R, если для условия x он находит решение w, удовлетворяющее $(x,w) \in R$.

Можем сопоставить массовой задаче, заданной отношением R, язык

$$L(R) - \{x \mid \exists w\colon (x,w) \in R\}.$$

Пример 1.0.1 (Массовая задача и соответствующий язык)**.**

$$\widetilde{\text{FACTOR}} = \{(n,d) \mid d|n,\ 1 < d < n\}.$$

Здесь условием задачи является натуральное число n, а решением некоторый (не 1, и не n) числа n.

Данной задаче соответствует язык

$$L(\widetilde{\text{FACTOR}}) = \text{множество всех составных чисел.}$$

1.0.2 Детерминированная машина Тьюринга

Определение 3: Детерминированная машина Тьюринга

Детерминированная машина Тьюринга —

- конечный алфавит (с началом ленты и пробелом): $\Sigma = \{0, 1, \triangleright, \_\}$;
- несколько лент, бесконечных в одну сторону;
- читающие/пишущие головки, по одной на каждую ленту;
- конечное множество состояний, в том числе начальное (q_S/q_0), принимающее (q_Y/q_{acc}) и отвергающее (q_N/q_{rej});
- управляющее устройство (программа), содержащее для каждых $q, c_1, \dots c_k$ одну инструкцию вида
$$(q, c_1, \dots c_k) \mapsto (q', c'_1, \dots c'_k, d_1, \dots d_k),$$
где $q, q' \in Q$ — состояния, $c_i, c'_i \in \Sigma$ — символы, обозреваемые головками, $d_i \in \{\rightarrow, \leftarrow, \cdot\}$ — направления движения головок.

ДМТ **принимает** входное слово, если заканчивает работу в q_{acc}, и **отвергает**, если заканчивает в q_{rej}.

ДМТ M **распознает язык** A, если принимает все $x \in A$ и отвергает все $x \notin A$.

$$A = L(M).$$

Замечание. Обычно есть отдельная строка только для чтения, куда записаны входные данные, и строка только для вывода, куда нужно поместить ответ. Остальные строки будут рабочими.

Определение 4

Время работы машины M на входе x — количество шагов (применений инструкций) до достижения q_{acc} или q_{rej}.

Используемая память — суммарное крайнее правое положение всех головой на *рабочих* лентах.

Теорема 1.0.1. Для любого $k \in \mathbb{N}$, работу ДМТ M с k рабочими лентами, работающую t шагов, можно промоделировать на ДМТ с двумя рабочими лентами за время $\mathcal{O}(t \log t)$, где константа $O(\dots)$ зависит только от размеров записи машины M).

□

- Перестроим исходную МТ:
  - Запишем все ленты в одну строку по символу из всех лент по очереди.
  - Будем бегать «лентой по головке»: выровняем все ленты, чтобы головки стояли друг над другом и далее будем сдвигать нужную ленту.
  - Заметим, что двустороннюю ленту можно смоделировать на односторонней с увеличением количества операций в константу раз: разрезаем двустороннюю пополам и записываем элементы через один.

- Теперь поймем, как экономично сдвигать ленты в однострочной записи.

  Разобьем строку на блоки начиная от позиции головки в две стороны: справа блоки R_i, слева L_i. При этом $|L_i| = |R_i| = 2^i$. Раздвинем символы, заполняя пустоту специальными символами пустоты, так, чтобы в каждом блоке ровно половина элементов были пустыми.

  Далее будем поддерживать такое условие:

  1. В блоке либо информация, либо пусто, либо наполовину пусто
  2. L_i пустой, согда R_i полный
  3. L_i наполовину пустой, согда R_i наполовину полный
  4. L_i полный, согда R_i пустой

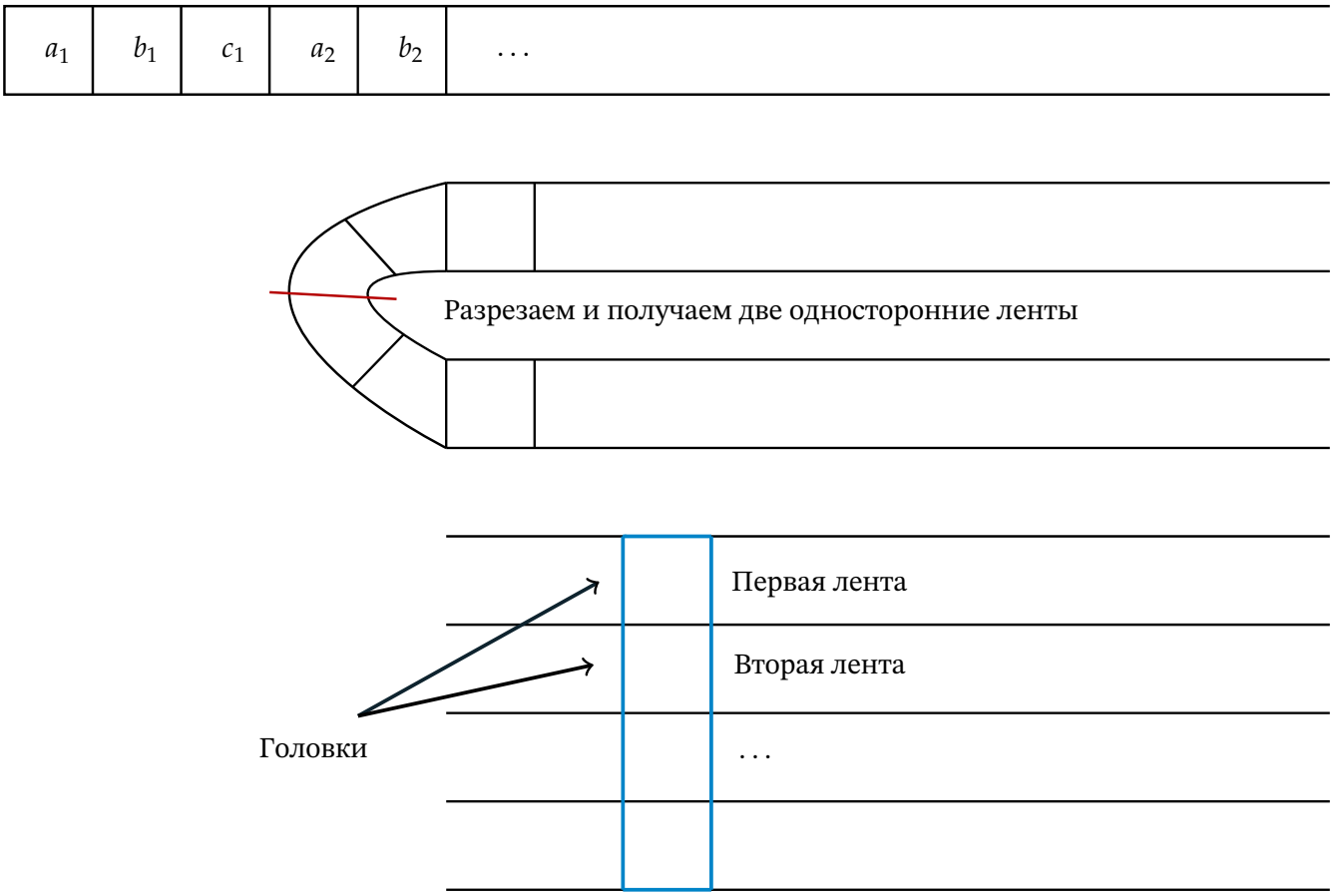

Figure 1.1: Построение новой МТ

Пусть нужно подвинуть головку влево. Найдем слева первый не пустой блок L_i. Возьмем из него правую половину и разложим по пустым $L_{<i}$ так, чтобы порядок сохранился и каждый из $L_{<i}$ стал полупустым, а первый символ попал под головку.

Так получится сделать, так как всего перемещаемых символов 2^{i-1}, а в j-й блок будет помещено 2^{j-1} символов, поэтому всего в $L_{<i}$ поместится

$$1 + 2 + 4 + \ldots + 2^{i-2} = 2^{i-1} - 1.$$

И один символ под головку.

Чтобы инвариант сохранился нужно теперь исправить правую часть.

Так как первые $i - 1$ левых блоков были пусты, первые $i - 1$ правых блоков полны, а R_i пуст. Заполним половину в R_i символами из R_{i-1}. Теперь R_{i-1} пустой, а меньшие полные. Проделаем ту же операцию еще раз для $i - 1$, потом для $i - 2$ и так далее.

Кода мы дойдем до R_1, положим туда элемент из-под головки.

Итого, инвариант сохранился.

- Посчитаем количество операций. В алгоритме мы переносим различные отрезки из одного места в другое. Чтобы делать это за линию, сначала скопируем нужный участок на вторую ленту, а затем запишем с нее.

  Тогда при перераспределении происходит $c \cdot 2^i$ операций: каждый символ переносили константное число раз (на вторую ленту, со второй ленты) плюс линейное перемещение от L_i к R_i несколько раз.

  Докажем, что с i-м блоком происходят изменения не чаще 2^{i-1} шагов. Пусть L_i пустой хотя бы наполовину заполнен. Когда мы забрали половину из него, мы заполнили все $L_{<i}$ и $R_{<i}$ наполовину.

  Поэтому, чтобы изменить L_i еще раз, нужно сначала опустошить все $L_{<i}$. При перераспределении в левой части становится на один элемент меньше, а всего там 2^{i-1} заполненное место. Для того, чтобы все они ушли из левой половины, придется совершить 2^{i-1} сдвигов.

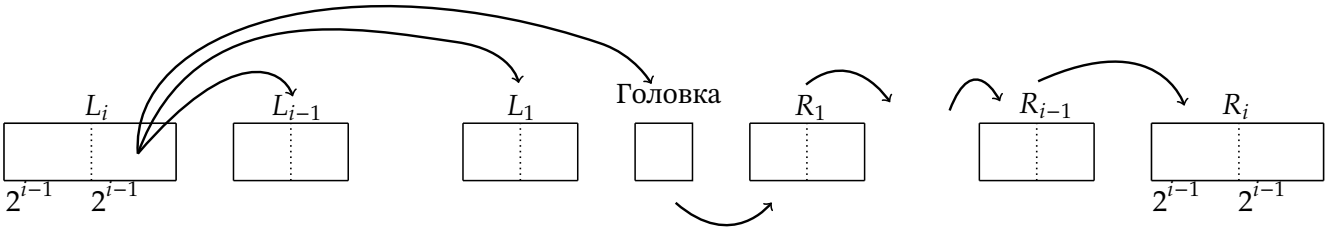

Figure 1.2: Структура блоков

Итого, для t шагов исходной машины будет

$$\sum_i c \cdot 2^i \cdot \frac{t}{2^{i-1}} = \mathcal{O}(t \log t).$$

■

Теорема 1.0.2 (Об универсальной МТ)**.** Существует ДМТ U, выдающая на входе $(M,\ x)$ тот же результат, что дала бы машина M на входе x, за время $\mathcal{O}(t \log t)$, где t — время работы M на входе x.

□ Используем прием из прошлой теоремы 1.0.1. ■

1.1 Классы сложности

1.1.1 Классы DTime и P

Определение 5: Конструируемая по времени функция

Функция $t\colon \mathbb{N} \to \mathbb{N}$ называется **конструируемой по времени**, если

- $t(n) \geqslant n$;
- двоичную запись $t(|x|)$ можно найти по входу x на ДМТ за $t(|x|)$ шагов.

Определение 6: Класс DTime

Язык L принадлежит классу $\mathbf{DTime}[t(n)]$, если существует ДМТ M, принимающая L за время $\mathcal{O}(t(n))$, где t конструируема по времени.
Константа может зависеть от языка, но не от длины входа.

Определение 7: Класс P

Класс языков, распознаваемых за полиномиальное время на ДМТ —

$$\mathbf{P} = \bigcup_c \mathbf{DTime}[n^c].$$

Будем обозначать задачи, заданные отношениями волной.

Определение 8

Массовая задача R **полиномиально ограничена**, если существует полином p, ограничивающий длину кратчайшего решения:

$$\forall x \left(\exists u\colon (x,u) \in R \Longrightarrow \exists w\colon \big((x,w) \in R \wedge |w| \leqslant p(|x|)\big) \right).$$

Массовая задача R **полиномиально проверяема**, если существует полином q, ограничивающий время проверки решения: для любой пары (x, w) можно проверить принадлежность $(x, w) \stackrel{?}{\in} R$ за время $q(|(x, w)|)$.

Определение 9: Класс $\widetilde{\mathbf{NP}}$

$\widetilde{\mathbf{NP}}$ — класс задач поиска, задаваемых полиномиально ограниченными полиномиально проверяемыми массовыми задачами.

Определение 10: Класс $\tilde{\mathbf{P}}$

$\tilde{\mathbf{P}}$ — класс задач поиска из $\widetilde{\mathbf{NP}}$, разрешимых за полиномиальное время.

То есть класс задач поиска, задаваемых отношениями R, что для всех $x \in \{0,1\}^*$ за полиномиальное время можно найти w, для которого $(x, w) \in R$.

Ключевой вопрос теории сложности $\tilde{\mathbf{P}} \stackrel{?}{=} \widetilde{\mathbf{NP}}$

Определение 11: Класс NP

NP — класс языков (задач распознавания), задаваемых полиномиально ограниченными полиномиально проверяемыми массовыми задачами:

$$\mathbf{NP} = \{L(R) \mid R \in \widetilde{\mathbf{NP}}\}.$$

Замечание. $L \in \mathbf{NP}$, если существует массовая п.о.п.п.[1] задача, такая, что

$$\forall x \in \{0,1\}^* \colon x \in L \iff \exists w \colon (x, w) \in R.$$

Определение 12: Класс P

P— класс языков (задач распознавания), распознаваемых за полиномиальное время.

$$\mathbf{P} = \{L(R) \mid R \in \mathbf{P}\}.$$

Замечание. Очевидно, $\mathbf{P} \subseteq \mathbf{NP}$.

Ключевой вопрос теории сложности $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

Лекция 2: †

12 nov

1.2 Недетерминированная машина Тьюринга

Определение 13: Недетерминированная машина Тьюринга

Недетерминированная машина Тьюринга — машина Тьюринга, допускающая более одной инструкции для данного состояния $q \in Q$ и $c_1, \dots c_k \in \Sigma$, то есть для состояния q и символа c функция δ будет многозначной.

Из такого определения получаем **дерево вычислений** вместо последовательности состояний ДМТ.

Мы говорим, что НМТ[2] **принимает** вход, если существует путь в дереве вычислений, заканчивающийся принимающим состоянием.

[1] полиномиально ограниченная полиномиально проверяемая

[2] недетерминированная машина Тьюринга

Утверждение. *В машины (ДМТ / НМТ) с заведомо ограниченным временем работы можно встроить и считать время вычислений на входах одной длины всегда . Для этого можем просто записать на дополнительную ленту $t(n)$ единиц и стирать по одной за ход.*

Определение 14: Эквивалентное определние НМТ

Недетерминированная машина Тьюринга — ДМТ, у которой есть дополнительный аргумент (конечная подсказка w на второй ленте).

▢ Докажем эквивалентность. Представим дерево вычисления как бинарное дерево и пронумеруем ребра из каждой вершины 0 и 1. Теперь запишем нужную принимающую ветку на ленту-подсказку. По подсказке можем построить дерево, где будет нужный путь. ■

Определение 15

Еще одно определение класса **NP** — класс языков, принимаемых полиномиальными по времени НМТ.

1.3 Сведения и сводимости

Определение 16: Сведение по Карпу

Язык L_1 **сводится по Карпу** к языку L_2, если существует полиномиально вычислимая функция f такая, что

$$\forall x\colon x \in L_1 \iff f(x) \in L_2.$$

Определение 17: Сведение по Левину

Задача поиска (отношение) R_1 **сводится по Левину** к задаче R_2, если существуют функции f, g, h такие, что для всех x_1, y_1, y_2 верно

- $R_1(x_1, y_1) \implies R_2(f(x_1), g(x_1, y_1))$;
- $R_1(x_1, h(f(x_1), y_2)) \impliedby R_2(f(x_1), y_2)$;
- f, g, h полиномиально вычислимые.

Замечание. Первое условие нужно для того, чтобы образы каждого входа, имеющего решение первой задачи, имели решение и второй задачи.

Теорема 1.3.1. Классы $\mathbf{P}, \mathbf{NP}, \tilde{\mathbf{P}}, \widetilde{\mathbf{NP}}$ замкнуты относительно этих сведений.

▢ Рассмотрим $R_2 \in \tilde{\mathbf{P}}$ и $R_1 \to R_2$. Тогда должно выполняться $R_1 \in \tilde{\mathbf{P}}$. Аналогично для $\widetilde{\mathbf{NP}}$. В обоих случаях R_2 задано п.о.п.п. Если говорим про $\tilde{\mathbf{P}}$, то еще есть алгоритм, работающий за полиномиально время.

Что можно узнать про R_1? Если есть решение для R_1, то функция g дает решение R_2, которое не на много длиннее. Еще есть h, которая позволяет построить из решения R_2 обратно построить решение R_1.

- Пусть есть некоторое решение y_1 для задачи R_1. Для него можно получить некоторое решение R_2 – $g(x_1, y_1)$.
  Так как R_2 полиномиально ограничено, для него есть полиномиальное решение поменьше. Поэтому, когда оно будет возвращено функцией h, исходное решение y_2 тоже окажется коротким.
- Полиномиальная проверяемость проверяется аналогично.
- Про алгоритм: если есть алгоритм для R_2 и x_1, сначала перегоняем $x_1 \to f(x_1)$, далее применяем алгоритм, получаем y_2, а далее, используя h, перегоняем обратно в R_1.

■

Определение 18: Оракульная МТ

Оракульная МТ — МТ с доступом к оракулу, который за один шаг дает ответ на некоторый вопрос.

Обозначение.
При переходе в состояние q_{in} происходит «фантастический переход» в состояние q_{out}, заменяющий содержимое некоторой ленты на ответ оракула.

M^B — оракульная машина M, которой дали оракул B.

Определение 19: Сведение по Тьюрингу

Язык или задача A **сводится по Тьюрингу** к B, если существует оракульная машина полиномиальная по времени $M^\bullet$ такая, что M^B решает A.

Например, если A — язык, $A = L(M^B)$.

Пример 1.3.1. Классы $\mathbf{P}$ и $\tilde{\mathbf{P}}$ замкнуты относительно сведений по Тьюрингу. А $\mathbf{NP}$ и $\widetilde{\mathbf{NP}}$ могут быть незамкнуты: если $A = \texttt{UNSAT}$, $B = \texttt{SAT}$, $M^O(x) = \overline{(x \in O)}$, и A сводится по Тьюрингу к B, но $B \in \mathbf{NP}$, а про A не известно.

1.3.1 Трудные и полные задачи

Определение 20: Трудный и полные задачи

Задача A называется **трудной** для класса $\mathbf{C}$, если $\forall C \in \mathbf{C}$: $C \to A$.

Задача A называется **полной** для класса $\mathbf{C}$, если она трудная и принадлежит $\mathbf{C}$.

Теорема 1.3.2. Если A — $\mathbf{NP}$-трудная и $A \in \mathbf{P}$, то $\mathbf{P} = \mathbf{NP}$.

Следствие 1. Если A — $\mathbf{NP}$-полная, то $A \in \mathbf{P} \iff \mathbf{P} = \mathbf{NP}$.

Задача об ограниченной остановке

Определение 21: $\texttt{BH}$

Определим задачу об ограниченной остановке $\widetilde{\texttt{BH}}(\langle M, x, 1^t\rangle, w)$[a] так: дана НМТ M и вход x, требуется найти такую подсказку w, чтобы M распознавала x не более чем за t шагов.

Соответствующая задача распознавания — ответить, существует ли такая подсказка.

[a] здесь 1^t — служебные t единиц

Теорема 1.3.3. Задача об ограниченной остановке $\widetilde{\mathbf{NP}}$-полная, а соответствующий язык $\mathbf{NP}$-полный.

□

- Принадлежность $\widetilde{\mathbf{NP}}$ и $\mathbf{NP}$ следует из существования универсальной ДМТ, которая за $\mathcal{O}(t \log t)$ промоделирует вычисление ДМТ, описание которой дано ей на вход.
- Проверим, что язык $\mathbf{NP}$-трудный. Пусть язык L принадлежит $\mathbf{NP}$, что равносильно существованию для соответствующего отношения R машины Тьюринга $M^*(x, w)$, которая работает за $\mathbf{poly}(|x|)$.

  Сведем L к задаче $\texttt{BH}$. Рассмотрим тройку $\langle M^*, x, 1^{\mathbf{poly}(|x|)}\rangle$. Пусть функция $f(x)$ будет равна этой тройке.

  Если и только если существует подсказка w для тройки $f(x)$ принадлежит $\texttt{BH}$, а наш язык L и определяется машиной M^*.
- Аналогично для задач поиска.

■

1.3.2 Булевы схемы

Определение 22: Булева схема

Булева схема — ориентированный граф без циклов,в вершинах которого записаны бинарные, унарные или нульарные операции над битами ($\wedge, \vee, \oplus$), при этом есть специальные вершины-входы и вершины-выходы.

Определение 23: `CIRCUIT_SAT`

$$\widetilde{\texttt{CIRCUIT\_SAT}} = \{(C, w) \mid C \text{ — булева схема}, C(w) = 1\}.$$

Очевидно, что $\texttt{CIRCUIT\_SAT} \in \mathbf{NP}$. Чтобы доказать $\mathbf{NP}$-трудность, сведем $\texttt{BH} \to \texttt{CIRCUIT\_SAT}$: будем рисовать конфигурацию МТ на схемах.

- Пусть каждый этаж системы — конфигурация ДМТ. Всего этажей будет столько же, сколько шагов в МТ, то есть t. Если в последнем этаже q_{acc}, то результат 1.
- Пересчет конфигураций: меняются только гейты рядом с положением головки. Выделим подсхему, которая должна по состоянию и элементу рядом с головкой получить новое состояние, положение головки и поменять элемент, то есть построить новый уровень с измененными элементами.

  Для хранения головки будем после каждого символа с ленты хранить d_i равное единице, если головка на символе c_i перед ним.

  Если $d_i = 0$, то новый $c_i' = c_i$, иначе нужно заменить c_i на c_i' из программы МТ:

  $$(q, c_{i-1}, c_i, c_{i+1}, d_{i-1}, d_i, d_{i+1}) \mapsto (q', c_i', d_i').$$

- Так как входная строка x нам дана, запаяем ее в схему. Тогда входом схемы останется подсказка w для НМТ, а выходом — попадание в q_{acc}.

Таким образом, мы по M, x, t построили схему $C(w)$.

Лекция 3: †

1.4 Теорема Кука-Левина

$\widetilde{\texttt{3-SAT}} = \{(F, A) \mid F \text{ — в 3-КНФ}, F(A) = 1\}$.

Пример 1.4.1.

$$((\neg x \vee \neg y \vee \neg z) \wedge (y \vee \neg z) \wedge (x), [x = 0, y = 1, z = 1]) \in \widetilde{\texttt{3-SAT}}.$$

Теорема 1.4.1 (Кук-Левин). $\widetilde{\texttt{3-SAT}}$ — $\mathbf{NP}$-полная задача.

□ Мы уже доказали полноту задач выполнимости булевых схем, поэтому будем сводить к ней.

Пусть у нас есть некоторая схема. Для каждого гейта заведем по переменной, которая обозначает результат операции в этом гейте. Входы тоже остаются гейтами в схеме.

Запишем для гейтов клозы длины 3, которые выражают результат в зависимости от аргументов.

Например, для входов x, y и операции $\oplus = g(x, y)$,

$$(x \vee y \vee \neg g)$$
$$(\neg x \vee \neg y \vee \neg g)$$
$$(x \vee \neg y \vee g)$$
$$(\neg x \vee y \vee g)$$

Еще для последнего гейта g (выходного) запишем клоз (g).

Значение в полученной схеме будет соответствовать результату конъюнкции всех переменных и наоборот: по формуле можем построить булеву схему и входные переменные выполняют ее. ■

Теорема 1.4.2. Пусть $R \in \widetilde{\mathbf{NP}}$, и соответствующий язык полон $L(R)$ — **NP**-полон. Тогда R сводится по Тьюрингу к $L(R)$.

□ Во-первых, задача поиска из **NP** сводится к $\widetilde{\mathtt{SAT}}$. При этом $\mathtt{SAT}$ сводится к $L(R)$.

Осталось свести $\widetilde{\mathtt{SAT}} \to \mathtt{SAT}$. Пусть нам дали выполняющий набор для F. Подставим первую переменную x_1 как 0 и как 1 в F (это тоже формулы) и спросим у оракула $\mathtt{SAT}$ про $F[x_1 = 0] \in \mathtt{SAT}$ и $F[x_1 = 1] \in \mathtt{SAT}$.

Так как F выполнима, хотя бы одна из полученных схем выполнима. Выберем ее и продолжим подставлять в нее. Так мы дойдем до конечной истиной формулы. Следовательно, последовательность подставляемых значений x_i и будет выполняющим набором. ■

1.4.1 Оптимальный алгоритм для $\widetilde{\mathbf{NP}}$-задачи

Пусть мы хотим решить задачу, заданную отношениям R, с входом x. Давайте переберем все машины (не только полиномиальные) и, если какая-то машина выдала результат y проверим его $R(x, y)$ за полином. Если ответ подошел, то просто заканчиваем работу, иначе продолжаем ждать других результатов.

Если машины были бы запущены параллельно, то мы бы нашли ответ за время самой быстрой машины на данном входе.

А мы будем делать шаги «змейкой»: выделим для l-ой машины 2^{-l} времени.

На очередном этапе $2^l(1 + 2k)$ будем моделировать k-ый шаг машины M.

Посчитаем замедление алгоритма. Мы хотим выдать ответ не сильно позже, чем самая быстрая машина. Но заметим, что такая машина одна, поэтому l это константа, следовательно, множитель 2^l тоже константа.

Как моделировать эти машины? Если было бы быстрое обращение к каждому элементу памяти, то $t(x) \leqslant \mathrm{const}_i \cdot t_i + p(|x|)$ (последнее на проверку), в случае с ДМТ получаем $t(x) \leqslant \mathrm{const}_i \cdot p(t_i(x))$.

Замечание. Если **P** = **NP**, то построенный алгоритм может решить $\mathtt{SAT}$ за полиномиальное от времени работы самой быстрой машины $p(t_i(x))$, но и оно полиномиально в случае **P** = **NP**.

Теорема 1.4.3. Если **P** ≠ **NP**, то существует язык $L \in \mathbf{NP} \setminus \mathbf{P}$, **не** являющийся **NP**-полным.

□ Найдем задачу, которая и не в **P** и не является **NP**-полной.

Занумеруем все полиномиальные ДМТ с полиномиальными будильниками (настроенными на по полиномиальное время):

$$M_1, M_2, \dots.$$

Аналогично пронумеруем полиномиальные сведедения с будильниками, то есть ожидаем какой-то ответ на выходе:

$$R_1, R_2, \dots.$$

Построим следующий язык $\mathcal{K} = \{x \mid x \in \mathtt{SAT} \wedge f(|x|) = 0 \pmod 2\}$, где $f(n)$ ведет себя следующим образом:

1. за n шагов вычисляем $f(0), f(1), \dots f(i) =: k$ (k — последнее значение, которое мы успели вычислить).
2. тоже за n шагов:
   (a) если $k = 0 \pmod 2$, то проверим, что $\exists z\colon M_{\frac{k}{2}}(z) \neq \mathcal{K}(z)$, и в случае успеха вернем $k + 1$, иначе (или истратили n шагов) k;
   (b) если $k = 1 \pmod 2$, проверим, что очередное $R_{\frac{k-1}{2}}$ правильно сводит $\mathtt{SAT}$ к R: если $\exists z\colon \mathcal{K}(R_{\frac{k-1}{2}}(z)) \neq \mathtt{SAT}(z)$, возвращаем $k + 1$, в любом другом случае — k.

Здесь первый пункт проверяет $\mathcal{K} \in \mathbf{P}$, а второй — $\mathtt{SAT} \to \mathcal{K}$, то есть $\mathcal{K} \in \mathbf{NP}$-трудная.

Для какого-то огромного n найдутся контр-примеры и мы получим $k + 1$.

3. $f(0) = 0$

Заметим, что $\mathcal{K} \in \mathbf{NP}$, так как выполняющий набор можно проверить принадлежность SAT за полином и подставить в f, которая тоже работает полином ($2n$ шагов).

- Пусть $\mathcal{K} \in \mathbf{P}$, тогда есть полиномиальная машина M, которая принимает этот язык. Поэтому в пункте (a) мы дальше этой машины не пройдем, так как контр-примера там нет. То есть с некоторого момента $f(n) = 0 \pmod 2$, по определению с некоторого места $\mathcal{K} = \mathtt{SAT}$, кроме конечного числа случаев, их можно разобрать отдельно. Тогда $\mathcal{K} \in \mathbf{NP\text{-}complete}$ и $\mathcal{K} \in \mathbf{P}$, поэтому $\mathbf{P} = \mathbf{NP}$.
- Если $\mathcal{K} \in \mathbf{NP\text{-}complete}$, то с некоторого момента мы не пройдем пункт (b), так как у нас действительно будет сводимость к $\mathcal{K}$. С некоторого места $f(n) = 1 \pmod 2$, а тогда $|\mathcal{K}| < \infty$. Следовательно, $\mathcal{K} \in \mathbf{P}$. Опять противоречие.

■

1.5 Полиномиальная иерархия

Обозначение. Пусть есть два класса языков $\mathcal{C}, \mathcal{D}$. Можно построить класс $\mathcal{C}^{\mathcal{D}}$, который состоит из языков вида C^D, где $D \in \mathcal{D}$ и C — машина для языка из $\mathcal{C}$.

Определение 24: Класс дополнений

$$\mathbf{co}\text{-}\mathcal{C} = \{L \mid \overline{L} \in \mathcal{C}\}.$$

Пример 1.5.1. $\mathtt{SAT} \in \mathbf{NP}$, дополнение к SAT, то есть {всюду ложные формул} $\in \mathbf{co\text{-}NP}$.

1.5.1 Полиномиальная иерархия

Самый нижний класс иерархии — $\mathbf{P}$. Остальные классы строятся также из $\mathbf{NP}$ и $\mathbf{co\text{-}NP}$.

$$\Sigma^0\mathbf{P} = \Pi^0\mathbf{P} = \Delta^0\mathbf{P} = \mathbf{P}$$
$$\Sigma^{i+1}\mathbf{P} = \mathbf{NP}^{\Pi^i\mathbf{P}}$$
$$\Pi^{i+1}\mathbf{P} = \mathbf{co\text{-}NP}^{\Sigma^i\mathbf{P}}$$
$$\Delta^{i+1}\mathbf{P} = \mathbf{P}^{\Sigma^i\mathbf{P}}$$
$$\mathbf{PH} = \bigcup_{i \geqslant 0} \Sigma^i\mathbf{P}$$

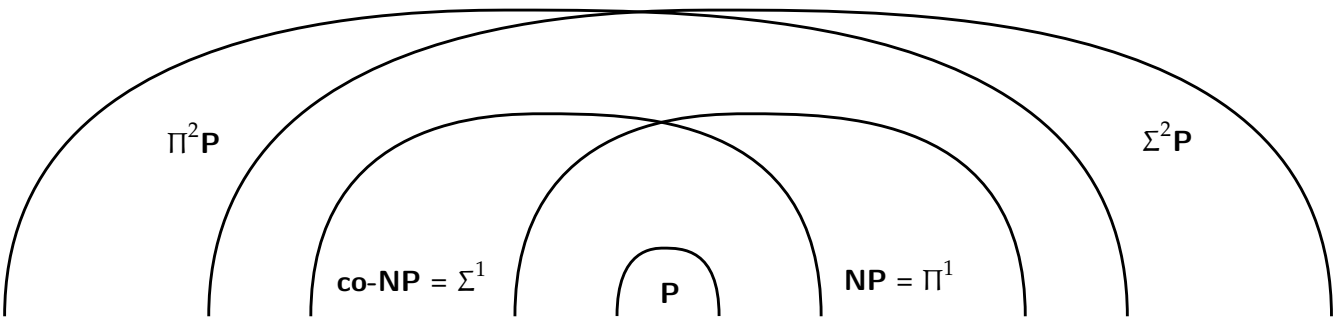

Figure 1.3: Полиномиальная иерархия

Лемма 1. $\mathbf{NP}^{\Pi^i\mathbf{P}} = \mathbf{NP}^{\Sigma^i\mathbf{P}}$

▢ Пусть есть машина M с оракулом A. Рассмотрим оракул $\overline{A}$ и построим машину M', чтобы $M'^{\overline{A}}$ вела себя аналогично M^A. Для этого просто M' будет переворачивать любой ответ, полученный от оракула, все остальные действия повторяем за M.

Из этого следует, что можно поменять $\Pi^i\mathbf{P}$ на $\overline{\Pi^i\mathbf{P}} = \mathbf{co\text{-}NP}^{\Pi^{i-1}\mathbf{P}} = \Sigma^i\mathbf{P}$. ■

Теорема 1.5.1. $L \in \Sigma^k\mathbf{P}$, когда [a] существует полиномиально ограниченное отношение $R \in \Pi^{k-1}\mathbf{P}$ такое, что для всех x :

$$x \in L \iff \exists y\colon R(x,y).$$

[a] тогда и только тогда, когда

▢

$\boxed{1 \Longrightarrow 2}$ Докажем по индукции.

- База: по определению $\Sigma^1\mathbf{P} = \mathbf{NP}$.
- Переход: $k-1 \to k$. Пусть $L = L(M^O)$, где M — полиномиальная НМТ, $O \in \Sigma^{k-1}\mathbf{P}$.

  По предположению индукции для O существует полиномиально ограниченное $S \in \Pi^{k-2}\mathbf{P}$ такое, что $\forall q\colon q \in O \iff \exists w\colon S(q,w)$.

  Сконструируем из этого R:

  – $R(x,y) = 1$, если y — принимающая ветвь вычисления M^O, при этом положительные ответы оракула должны быть снабжены сертификатами w: $S(q,w) = 1$.
  То есть все переходы, основанные на ответе оракула «да», должны также содержать «доказательство».
  Проверим, что это отношение из $\Pi^{k-1}\mathbf{P}$, и, что оно задает язык L.
  – $R \in \Pi^{k-1}\mathbf{P}$: детерминировано проверяем корректность y, а далее проверяем, что ответы оракула были верными. Для ответов «нет» нужно проверить, что O для него равно нулю ($\Pi^{k-1}\mathbf{P}$ вычисление), а для ответов «да» — что S равно 1, то есть проверить сертификат ($\Pi^{k-2}\mathbf{P}$ вычисление).
  Нужно, чтобы все $\Pi^{k-1}\mathbf{P}$ и $\Pi^{k-2}\mathbf{P}$ (это частный случай $k-1$) вычисления вернули «да». Для этого построим схему: присоединим к большой конъюнкции все вычисления, чтобы ответ был положительным, нужно, чтобы все ветки $\Pi^{k-1}\mathbf{P}$ (то есть **co-NP**) вернули одно и то же, при этом мы остаемся в $\Pi^k\mathbf{P}$.

  Для машины M существует принимающее вычисление и оно может быть дано в качестве y.
  Наоборот, если у нас есть корректное вычисление машины M, то оно и будет принадлежать L.

$\boxed{2 \Longrightarrow 1}$ Пусть у нас есть отношение R. Возьмем машину с оракулом R. Она будет недетерминировано выбирать y и проверять, то есть спрашивать у оракула, $R(x,y)$.

■

Теорема 1.5.2. $L \in \Pi^k\mathbf{P}$, когда существует полиномиально ограниченное отношение $R \in \Sigma^{k-1}\mathbf{P}$ такое, что для всех x :

$$x \in L \iff \forall y\colon R(x,y).$$

Следствие 2. $L \in \Sigma^k\mathbf{P}$, когда существует полиномиально ограниченное отношение $R \in \mathbf{P}$ такое, что для всех x:

$$x \in L \iff \exists y_1 \forall y_2 \exists y_3 \ldots R(x, y_1, y_2, \ldots y_k).$$

Следствие 3. $L \in \Pi^k\mathbf{P}$, когда существует полиномиально ограниченное отношение $R \in \mathbf{P}$ такое, что для всех x:

$$x \in L \iff \forall y_1 \exists y_2 \forall y_3 \ldots R(x, y_1, y_2, \ldots y_k).$$

Лекция 4: †

1.6 $\Sigma^k\mathbf{P}$-полная задача

Определение 25: Язык $\texttt{QBF}_k$

Язык $\texttt{QBF}_k$ — язык, состоящий из замкнутых истинных формул вида

$$\exists X_1 \forall X_2 \exists X_3 \dots X_k \varphi,$$

где φ — формула в КНФ или ДНФ, а $\{X_i\}_{i=1}^k$ — разбиение множества переменных этой формулы на непустые подмножества.

Теорема 1.6.1. $\texttt{QBF}_k$ — $\Sigma^k\mathbf{P}$-полный язык.

□

- Во-первых, проверим принадлежность. По следствию из прошлой теоремы достаточно найти полиномиально ограниченное отношение $R \in \mathbf{P}$ такое, что $\forall x\colon x \in \texttt{QBF}_k \iff \exists y_1 \forall y_2 \exists y_3 \dots R(x, y_1, y_2, y_3, \dots)$.
  Возьмем просто R, которое для R(формула с $\varphi, x_1, x_2, \dots$) выдает результат подстановки x_i в φ.
  Проверка будет работать за полином, так как $R \in \mathbf{P}$.
- Докажем, что к этой задаче сводится любой язык из $\Sigma^k\mathbf{P}$.
  Пусть есть язык $L \in \Sigma^k\mathbf{P}$. Тогда существует полиномиально ограниченное отношение $R \in \mathbf{P}$:

  $$\forall x\colon x \in L \iff \exists y_1 \forall y_2 \exists y_3 \dots R(x, y_1, y_2, \dots).$$

  – Если в конце формулы идет квантор $\exists$, то запишем язык R в таком виде (как в теореме Кука-Левина):

  $$R(z) \iff \exists w \Phi(z, w).$$

  Тогда определение языка будет иметь следующий вид:

  $$\forall x\colon x \in L \iff \exists y_1 \forall y_2 \exists y_3 \dots \exists y_k \exists w \Phi(x, y_1, y_2, \dots, y_k, w).$$

  Но так как последние два квантора можно объединить в один, получаем формулу из $\texttt{QBF}_k$.
  – Иначе запишем $\overline{R}$ в виде булевой формулы Ψ, как в теореме Кука-Левина:

  $$\overline{R}(z) \iff \exists w \Psi(z, w).$$

  Тогда определение языка будет иметь следующий вид:

  $$\forall x\colon x \in L \iff \exists y_1 \forall y_2 \exists y_3 \dots \forall y_k \forall w \overline{\Psi}(x, y_1, y_2, \dots, y_k, w).$$

  Аналогично можно объединить последние два квантора.

■

Следствие 4. Если в определении $\texttt{QBF}_k$ заменить кванторы существования на кванторы всеобщности и наоборот, получится $\Pi^k\mathbf{P}$-полный язык.

1.6.1 Коллапс полиномиальной иерархии

Теорема 1.6.2. Если $\Sigma^k\mathbf{P} = \Pi^k$, то $\mathbf{PH} = \Sigma^k\mathbf{P}$.

□ Достаточно показать, что следующий уровень равен текущему: $\Sigma^{k+1}\mathbf{P} = \Pi^k\mathbf{P}$.
Пусть $L \in \Sigma^{k+1}\mathbf{P}$, то есть $L = \{x \mid \exists y\colon R(x, y)\}$ для $R \in \Pi^k\mathbf{P} = \Sigma^k\mathbf{P}$.

Тогда существует $S \in \Pi^k\mathbf{P}$ такое, что

$$R(x,y) \iff \exists z\colon S(x,y,z).$$

После подстановки получаем

$$x \in L \iff \underbrace{\exists y \exists z}_{\exists t}\colon \underbrace{S(x,y,z)}_{S(x,t)}.$$

То есть $L \in \Sigma^k\mathbf{P}$.

Замечание. Доказали про следующие Σ классы, но из этого следует и, что Π классы тоже будут совпадать.

■

Следствие 5. Если существует **PH**-полная задача, то полиномиальная иерархия коллапсирует (конечна).

□ Полный язык лежит в каком-то $\Sigma^k\mathbf{P}$, при этом все остальные к нему сводятся, следовательно, они тоже лежат в $\Sigma^k\mathbf{P}$. ■

1.7 Классы, ограниченные по времени и памяти

Определение 26: DSpace

$$\mathbf{DSpace}[f(n)] = \{L \mid L \text{ принимается ДМТ с памятью } \mathcal{O}(f(n))\},$$

где $f(n)$ должна быть неубывающей и вычислимой с памятью $\mathcal{O}(f(n))$ (на входе 1^n, на выходе двоично представление $f(n)$).

$$\mathbf{PSPACE} = \bigcup_{k\geq 0} \mathbf{DSpace}[n^k].$$

1.7.1 PSPACE-полная задача

Определение 27: Язык QBF

Язык QBF — язык, состоящий из замкнутых истинных формул вида

$$q_1 x_1 q_2 x_2 \ldots \varphi,$$

где φ — формула в КНФ, $q_i \in \{\forall, \exists\}$.

Теорема 1.7.1. Язык QBF **PSPACE**-полон.

□

- QBF $\in$ **PSPACE**. Рассмотрим дерево перебора всех значений переменных. В каждом листе будет записано значение формулы. Значение булевой формулы можем вычислить поиском в глубину. Для этого понадобится

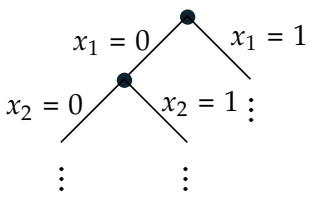

Figure 1.4: Дерево перебора

память равная глубине дерева, то есть пропорциональная количеству кванторов, еще нужна память для вычисления значения φ, для этого тоже вполне хватит полинома.

- Сведем $L \in$ **PSPACE** к `QBF`. Для языка L есть МТ. Модифицируем ее немного: если у нее было многого принимающих состояний, например, из-за каких-то рабочих данных на других лентах, будем перед переходом в принимающее состояние очищать все, кроме q_{acc}.

  Теперь построим граф, где состояния будут вершинами, а ребрами будут означать, что из одно конфигурации можно получить другую.

  Так как МТ детерминированная, выходит будет ровно одно ребро (кроме конечной).

  Поскольку память полиномиальна, всего вершин будет $2^{p(n)}$, где $p(n)$ — некоторый полином.

  Длина пути точно не более $2^{p(n)} + 1$, либо он зацикливается.

  Запишем теперь это в формулу. Пусть

$$\varphi_i(c_1, c_2) = \text{существует путь из } c_1 \text{ в } c_2 \text{ длины } \leqslant 2^i.$$

  Возьмем вершину d посередине пути $c_1 \to c_2$. Заметим, что можно записать так

$$\varphi_i(c_1, c_2) = \exists d\ \forall x \forall y \colon \big((x = c_1 \wedge y = d) \vee (x = d \wedge y = c_2)\big) \Longrightarrow \varphi_{i-1}(x, y).$$

  Формула, которую мы будем получать рекурсивно подставляя в $\varphi_{p(n)}(q_0, q_{acc})$, будет иметь длину $p(n^2)$, то есть полином.

$$\varphi_0(c_1, c_2) = \text{можно ли из первого состояния за один шаг получить второе.}$$

Замечание. После построения формулы нужно будет привести ее к нужному виду: перенести кванторы в начало, перевести формулы в КНФ.

■

Следствие 6. Если **PH** = **PSPACE**, то **PH** коллапсирует.

1.7.2 Иерархия по памяти

Теорема 1.7.2. $\mathbf{DSpace}[s(n)] \neq \mathbf{DSpace}[S(n)]$, где $s(n) = o(S(n))$ и для всех $n > n_0 \colon S(n) \geqslant \log n$.

□ Рассмотрим следующий язык:

$$L = \left\{ x = M01^k \;\middle|\; \begin{array}{l} k \in \mathbb{N} \cup \{0\}, \\ |M| < S(|x|), \\ M \text{ отвергает } x \text{ с памятью } \leqslant S(|x|) \end{array} \right\}.$$

Заметим, что $L \in \mathbf{DSpace}[S(n)]$, так как можем промоделировать работу на универсальной машине Тьюринга, а памяти дополнительной она почти не использует.

Докажем, что $L \notin \mathbf{DSpace}[s(n)]$. Пусть M_* распознает L с памятью $s_*(|x|) = \mathcal{O}(s(|x|))$.

Выберем N_1 таким, что $\forall n > N_1 \colon s_*(n) < S(n)$.

Еще найдем такое $N_* > \max(N_1, 2^{|M_*|})$.

Теперь посмотрим как себя ведет M^*. Если $M_*\underbrace{\left(M_*01^{N_* - |M_*| - 1}\right)}_{x} = 1$, то M_* точно не отвергает x, имеет размер меньше $S(|x|)$ и использует не более $S(|x|)$ памяти. Но тогда $x \notin L$.

Иначе все условия принадлежности к L выполнены и $x \in L$.

Противоречие. Получили, что $\mathbf{DSpace}[s(n)] \subset \mathbf{DSpace}[S(n)]$. ■

1.7.3 Если памяти совсем мало...

Две теоремы без доказательств для общего развития.

Теорема 1.7.3. $\mathbf{DSpace}[\log\log n] \neq \mathbf{DSpace}[\mathcal{O}(1)]$

Теорема 1.7.4. $\mathbf{DSpace}[(\log\log n)^{1-\varepsilon}] = \mathbf{DSpace}[\mathcal{O}(1)]$

1.7.4 Иерархия по времени

Теорема 1.7.5. $\mathbf{DTime}[t(n)] \neq \mathbf{DTime}[T(n)]$, где $t(n)\log t(n) = o(T(n))$, $T(n) = \Omega(n)$.

□ Аналогично ситуации с памятью рассмотрим следующий язык:

$$L = \left\{ x = M01^k \;\middle|\; \begin{array}{l} k \in \mathbb{N} \cup \{0\}, \\ |M| < T(|x|), \\ M \text{ отвергает } x \text{ за время } \leqslant (S\dfrac{|x|}{\log T(|x|)}) \end{array} \right\}.$$

Также за счет универсальной МТ получаем принадлежность $\mathbf{DTime}[T(|x|)]$ (именно из-за этого нам нужны логарифмы), она $f(n)$ шагов произвольной МТ моделирует за $\mathcal{O}(f(n)\log f(n))$ шагов.

Остальное полностью аналогично иерархии по памяти. ■

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \ldots \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP} = \bigcup_{k\geqslant 0} \mathbf{DTime}[2^{n^k}].$$

Про **PSPACE** мы доказали, что внутри есть неравенство. Следовательно, в каком-то из этих включений тоже должно быть строгое, но пока не известно какое.

1.8 Полиномиальные схемы

Определение 28

$L \in \mathbf{Size}[f(n)]$, если существует семейство булевых схем $\{C_n\}_{n\in\mathbb{N}}$ такое, что

- $\forall n\colon |C_n| \leqslant f(n)$;
- $\forall x\colon x \in L \iff C_{|x|}(x) = 1$.

Определение 29: Полиномиальные схемы

$$\mathbf{P/poly} = \bigcup_{k\in\mathbb{N}} \mathbf{Size}[n^k].$$

Утверждение. $\boldsymbol{P \subsetneq P/poly}$.

□ Например, $L = \{1^n \mid n$ — номер останавливающейся МТ, $M_n(n) = 1\}$. $L \notin \mathbf{P}$, но легко вычисляется схемами

$$C_n(x) = \begin{cases} 0, & 1^n \notin L \\ 0, & x \neq 1^n \\ 1, & 1^n \in L \end{cases}$$

■

Определение 30

$L \in \mathbf{Size}[f(n)]$, если имеются $R \in \mathbf{P}$ и последовательность строк (подсказка) $\{y_n\}_{n\in\mathbb{N}}$ полиномиальной

длины такие, что

$$\forall x\colon x \in L \iff R(x, y_{|x|}) = 1.$$

Замечание (связь определений).

$\boxed{1 \Longrightarrow 2}$ $\{C_n\}$ — подсказки, $R(x, C_{|x|} = C_{|x|}(x)$.
$\boxed{2 \Longrightarrow 1}$ Берем R, строим $C_n = C'_n(\ldots, y_n)$ — подставили y_n.

Лекция 5: †

Теорема 1.8.1 (Карп-Липтон). $\mathbf{NP} \subseteq \mathbf{P/poly} \Longrightarrow \mathbf{PH} = \Sigma^2\mathbf{P}$.

▢ Покажем, что $\Sigma^3\mathbf{P}$-полный язык лежит в $\Sigma^2\mathbf{P}$. Тогда все следующие классы тоже схлопываются. Мы знаем $\Sigma^3\mathbf{P}$-полный язык

$$\mathtt{QBF}_3 = \{F \text{ — формула в КНФ} \mid \exists x \forall y \exists x\ F(x, y, z)\}.$$

Докажем, что он лежит в $\Sigma^2\mathbf{P}$.

Заметим, что корректность некоторой схемы $G \in \mathtt{SAT}$ можно проверить так: подставим в первую переменную сначала 0, а потом 1 и проверим полученные схемы. Исходная корректна, согда хотя бы одна из полученных корректна. Можно записать это так:

$$\forall G\colon C_{|G|} \stackrel{?}{=} C_{|G[x_1 := 0]|}\left(G[x_1 := 0]\right) \vee C_{|G[x_1 := 1]|}\left(G[x_1 := 1]\right).$$

Хотим доказать, что $\mathtt{QBF}_3$ можно уложить в два квантора. Сделаем это следующим образом:

$$\begin{aligned}\left(\exists x \forall y \exists z\ F\right) \in \mathtt{QBF}_3 \iff\ & \exists \text{ схемы } C_1, \ldots C_{|F|} \text{ размера, ограниченного полиномом} \\ & \exists x \\ & \forall y \\ & \forall G \text{ — булева формула длина не более } |F| \\ & (\text{семейство } \{C_i\} \text{ корректно для } G) \wedge C_{|F|}(F(x, y, z)) = 1\end{aligned}$$

Такая запись принадлежит $\Sigma^2\mathbf{P}$. ■

1.8.1 Схемы фиксированного полиномиального размера

Теорема 1.8.2. $\forall k\colon \Sigma^4\mathbf{P} \not\subseteq \mathbf{Size}[n^k]$

▢ Заметим, что существует функция $f\colon \{0,1\}^n \to \{0,1\}$, зависящая только от первых $c \cdot k \cdot \log n$ битов, для которой нет булевой схемы размера n^k, так как всего функций с ограничением на биты $2^{2^{c \cdot k \cdot \log n}}$, а схем $2^{n^k \log n}$.

Найдем такую в $\Sigma^4\mathbf{P}$:

$$y \in L \iff \exists f\ \forall c\ (\text{схемы размера } n^k)\ \forall f'\ \exists x \exists c'\ (\text{схема размера } n^k)\ \forall x'\colon$$
$$\underbrace{f(x) \neq c(x)}_{\text{не принимается схемой}} \wedge \underbrace{\left((f < f') \vee f'(x') = c'(x')\right)}_{\text{лексикографически первая } f} \wedge \underbrace{\left(f(y) = 1\right)}_{\text{значение}}$$

Все кванторы имеют полиномиальные размеры. ■

Следствие 7. $\forall k\colon \Sigma^2\mathbf{P} \cap \Pi^2\mathbf{P} \not\subseteq \mathbf{Size}[n^k]$[a].

[a] Здесь берется пересечение, так как схемам все равно, выдавать 0 или 1

▢ Пусть $\Sigma^2\mathbf{P} \cap \Pi^2\mathbf{P} \subseteq \mathbf{Size}[n^k]$. Тогда $\mathbf{NP} \subseteq \mathbf{P/poly}$, поэтому можно применить теорему Карпа-Липтона:

$$\mathbf{PH} = \Sigma^2\mathbf{P} \cap \Pi^2\mathbf{P} \subseteq \mathbf{Size}[n^k].$$

Но $\Sigma^4\mathbf{P} \subseteq \mathbf{PH} \subseteq \mathbf{Size}[n^k]$. Противоречие. ■

1.8.2 Класс NSpace

Определение 31: NSpace

$\mathbf{NSpace}[f(n)] = \{L \mid L$ принимается НМТ с пасятью $\mathcal{O}(f(n))\}$.

Замечание.
- $f(n)$ должна быть конструируемая по памяти;
- входная лента read-only, выходная лента write-only, память на них не учитывается;
- ленту подсказки можно читать только слева направо.

$$\mathbf{NPSPACE} = \bigcup_{k \geqslant 0} \mathbf{NSpace}[n^k].$$

1.9 Логарифмическая память

Определение 32

$\texttt{STCON} = \{(G, s, t) \mid G$ — ориентированный граф, $s \rightsquigarrow t\}$.

Лемма 2. $\texttt{STCON} \in \mathbf{DSpace}[\log^2 n]$.

▢ Будем делить путь пополам и искать путь от начала до середины и от середины до конца. Пусть $\mathrm{PATH}(x, y, i)$ равно 1, если есть путь из x в y длины не более 2^i.

$$\begin{aligned} \mathrm{PATH}(x, y, 0) &= (x, y) \in E \\ \mathrm{PATH}(x, x, 1) &= 1 \\ \mathrm{PATH}(x, y, i) &= \bigvee_z \big(\mathrm{PATH}(x, z, i-1) \wedge \mathrm{PATH}(z, y, i-1)\big) \end{aligned}$$

Перебираем промежуточную вершину. Будем хранить «задания» (пары, для которых проверяем путь) в стеке. Достаем оттуда одно «задание» и заменяем его на два меньших. Когда-то мы либо найдем 1, тогда нужно вернуться к проверке второй половины, либо 0, тогда нужно перейти к следующей промежуточной вершине, так как с текущей пути нет.

Оценим память: на счетчик промежуточных вершин и на стек нужен логарифм. Тогда всего не более $\log^2 n$.

■

Теорема 1.9.1. $\mathbf{NSpace}[f(n)] = \mathbf{DSpace}[f^2(n)]$ для $f(n) = \Omega(\log n)$.

▢ Построим граф псеводоконфигураций: вершина — состояние, содержимое *рабочих* лент и положение *всех* головок.

Всего памяти для хранения псевдоконфигурации требуется $2^{f(n)\cdot k}$ памяти (состояний конечное число, положение занимает $\log n$, остается содержимое).

Считаем, что у нас только одно принимающее состояние.

Заметим, что граф нам нужен только, когда хотим выяснить достижимость за один шаг.

Делаем такой же стек, как и выше, на него уходит порядка $f(n)$ памяти (на каждую «задачу» три числа: начало, конец и длина).

Теперь посмотрим на «задачу» $(z, t, 0)$. Мы хотим выяснить, есть ли переход из z в t. Для этого сравниваем их посимвольно кроме состояния.

Вспоминаем, что мы проверяем, что данная НМТ M на данном входе выдает 1. Поэтому для сравнения z и t осталось посмотреть табличку перехода M и на текущий символ на входной ленте (его индекс хранится вместе с рабочими индексами) и понять, можно ли из z получить t.

Так как один стек занимает $\mathcal{O}(f(n))$ памяти, то для всей работы достаточно $\mathcal{O}(f^2(n))$. ■

Следствие 8. $\mathbf{PSPACE} = \mathbf{NPSPACE}$

Лемма 3. `STCON` является **NL**-полной (относительно logspace-сведений, то есть сводящая функция использует логарифмическую память).

□

- Принадлежность **NL**: храним только текущую вершину и угадываем следующее состояние, пока не дойдем до конца.
- Полнота: если есть задача из **NL**, то есть НМТ M, которая использует логарифмическую память.

$$M(x) = 1 \iff start \rightsquigarrow accept.$$

Поэтому функция сведения должна построить граф по машине Тьюринга, а ребра в нем рисует, когда это ребро есть для данного входа x.

■

Утверждение. *Для неориентированного графа алгоритм гораздо сложнее:* `USTCON` $\in$ **L** *(Reingold, 2004).*

Утверждение. *Все задачи из* **L** *являются* **L***-полными.*

1.10 Равномерные полиномиальные схемы

Определение 33

Семейство схем $\{C_n\}_{n\in\mathbb{N}}$ **равномерно**, если существует полиномиальный алгоритм A такой, что $A(1^n) = C_n$.

Лемма 4. Равномерные схемы задают класс **P**.

□ Полиномиальная МТ на входе x сначала запускает $A(|x|)$, потом запускает полученную схему $C(x)$.

Наоборот, пусть дана полиномиальная МТ, про вход мы знаем, что $|x| = n$, тогда можно нарисовать схему из состояний, которая будет работать как МТ. Каждый уровень можно породить алгоритмом, который будет их пересчитывать по функции перехода. ■

Определение 34

Семейство схем $\{C_n\}_{n\in\mathbb{N}}$ logspace-**равномерно**, если существует полиномиальный алгоритм A, использующий $\mathcal{O}(\log n)$ памяти, такой, что $A(1^n) = C_n$.

Глубина булевой схемы — время параллельного вычисления.

Определение 35: Nick's class

Класс $\mathbf{NC}^i = \{L \mid$ для L есть logspace-равномерные схемы глубины $\mathcal{O}(\log n)\}$.

Класс $\mathbf{NC} = \bigcup_i \mathbf{NC}^i \subseteq \mathbf{P}$.

Лемма 5. Композиция двух logspace-функций $f_2(f_1(x))$ принадлежит logspace.

□ Будем запускать и f_1 и f_2. Храним счетчики позиций на лентах: для f_1 на выходе и для f_2 на входе.

Пусть f_2 нужен очередной бит входа с номером y_i. Запускаем f_1, как только ее счетчик достигает y_i, останавливаем f_1, продолжаем работу f_2.

Если f_2 требуется новый y_j (возможно $j < i$), то мы запускаем f_1 снова.

И так пока f_2 не закончит. Во время работы f_1, f_2 запущено только два экземпляра, поэтому память логарифмическая (для счетчиков – логарифм размера ленты).

Замечание. f_1 мы моделируем без выходной ленты, помним только последний символ.

■

Теорема 1.10.1. Если L — **P**-полный (относительно logspace-сводимости), то

$$L \in \mathbf{L} \Longrightarrow \mathbf{P} = \mathbf{L}.$$

□ Если $L' \in L$, то он сводится к L, так как второй **L**-полный. А L решается logspace-алгоритмом. Получили дыве функции, обе logspace, значит, $L' \in \mathbf{L}$. ■

Лекция 6: †

10 dec

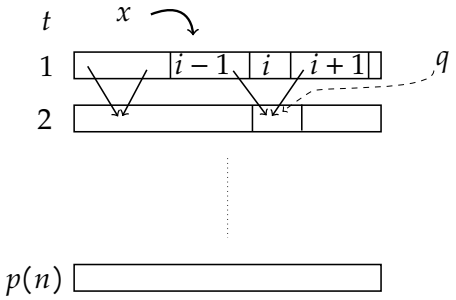

Figure 1.5: Построение схемы по ДМТ

Пример 1.10.1 (**P**-полный язык)**.** Приведем пример **P**-полного языка.

$$\texttt{CIRCUIT\_EVAL} = \{(\text{схема } C, \text{вход } x \mid C(x) = 1\}.$$

Нам уже дан вход, остается только его проверить. Это легко и быстро сделать.

Докажем полноту. Вспомним, как строили схему по недетерминированной МТ в доказательстве теоремы Кука-Левина. Построим аналогично по детерминированной: каждый слой отвечает за состояние, переход к следующему – маленькие подсхемки, на вход только x.

Единственное, что нужно проверить — что нарисовать можно с логарифмической памятью.

Описание машины — константа. От x нам нужна только длина, чтобы узнать количество этажей.

На каждом уровне, чтобы построить следующий, нам нужно знать положение головки, три бита прошлой строки и состояние. Будем хранить t — номер уровня, i — положение головки, логарифма для этого хватит.

Теорема 1.10.2. $\mathbf{NC}^1 \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{NC}^2$.

□

1. Начнем с последнего включения. У нас есть недетерминированная машина, использующая логарифмическую память, хотим ее параллелизовать.

   Вспомним про граф псевдоконфигураций нашей детерминированной машины. Так как память логарифмическая, конфигураций полиномиальное число.

   Можем задать допустимые переходы НМТ матрицей смежности A ($k \times k$). Чтобы найти пути длины k (а больше нам не нужно), возведем булево в степень k.

   Это $\log k$ последовательных умножений: $A^2, (A^2)^2, \ldots$, причем каждое умножение пары булевых матриц вычисляется схемой глубины $\mathcal{O}(\log k)$.

   Тогда общая глубина $\log^2 k$, из чего следует последнее включение.

2. Среднее включение тривиально.

3. Докажем первое. Мы знаем, что logspace функции можно комбинировать и получится тоже logspace.

   Сейчас находимся в $\mathbf{NC}^1$, поэтому у нас есть семейство схем $\{C_n\}$, каждая вычисляется с логарифмической памятью.

   Построим композицию трех функций:

   - Первая функция вычисляет схему. Заметим, что эта схема будет логарифмической глубины (так как мы в $\mathbf{NC}^1$).
   - Докажем, что эту схему можно преобразовать в формулу тоже с логарифмической памятью.
     Пусть есть некоторая схема. Склонируем все гейты, которые используются два раза в следующих гейтах. Так как глубина логарифмическая, больше полинома вершин в полученном дереве не окажется. Научимся делать это формально с logspace. Запустим dfs от выхода схемы и будем записывать все

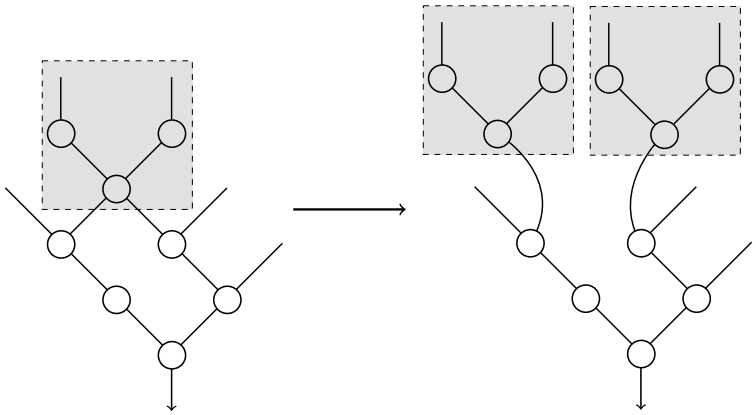

Figure 1.6: Схема → дерево

пути. Заметим, что каждый путь представляет из себя как раз одну из полученных веток с клонированными уникальными гейтами.

После построения одного пути, начинаем из корня, храним только текущую вершину. «Сам путь нам подскажет, где мы ходили налево, а где еще нет»[3].

- Теперь вычисляем значение формулы на входе x. Аналогично запускаем dfs, храним текущую вершину и результаты в поддеревьях, когда обошли оба поддерева, поднимемся выше и передадим результат операции гейта.

■

Теорема 1.10.3. Если L — $\mathbf{P}$-полный, то все параллелизуется

$$L \in \mathbf{NC} \iff \mathbf{P} = \mathbf{NC}.$$

□ Если $L' \in \mathbf{P}$, его можно свести к L (logspace), а $L \in \mathbf{NC}$. То есть $\exists i\colon L \in \mathbf{NC}^i$.
Логарифмические вычисления мы умеем превращать в параллельные с $\log^2$, поэтому $L' \in \mathbf{NC}^{i+2}$. ■

1.10.1 Замкнутость NSpace относительно пополнения

Теорема 1.10.4 (Immerman, Szelepcsenyi)**.** $\texttt{STCON} \in \mathbf{co\text{-}NL}$. Альтернативная формулировка $\overline{\texttt{STCON}} \in \mathbf{NL}$.

□ Придумаем такие подсказки, которые будут нас убеждать, что пути между s и t нет.
Пусть S_i — множество вершин на расстоянии не более i от s.

[3]Тогда уж «сила подскажет», но я не понимаю как

Сертификат того, что $x \in S_i$ — последовательность вершин от s до x. Так как эти подсказки будут в недетерминированной подсказке, память занимать они не будут.

Теперь посмотрим на такой сертификат *непринадлежности* ($x \notin S_i$):

- все вершины S_i с сертификатами принадлежности,
- размер S_i[4]

Теперь нужно проверить все вершины из S_i, что ни одна из них не совпала с x, и, что их столько, сколько нужно.

Будем строить сертификат $|S_i|$ индуктивном:

- $|S_{i-1}|$ мы знаем,
- переберем все u и проверим, что $u \in S_i$:
  - переберем все v, проверяя $(v, u) \in E$, и, если такое ребро есть, сертификат принадлежности (то есть путь) для $v \in S_{i-1}$,
  - в процессе проверки, подсчитываем количество правленых сертификатов, в итоге должно сойтись с $|S_{i-1}|$.

  Теперь мы знаем общее число вершин из S_i.

Замечание. Что мы храним? Храним номера вершин и счетчики, поддерживаем простые арифметические операции. Еще нужно хранить обращение к входу (несколько счетчиков для поиска ребер), считаем, что граф записан в нормальном виде, например, как перечисление ребер или матрица смежности.

■

Следствие 9. Если $s(n) = \Omega(\log n)$, то $\mathbf{NSpace}[s(n)] = \mathbf{coNSpace}[s(n)]$.

□ Пусть есть МТ M для языка из $\mathbf{NSpace}[s(n)]$. Тогда нас интересует достижимость в графе конфигураций из $2^{s(n)}$.

Только что мы научились проверять достижимость в классе **co-NL**, то есть доказывать достижимость в классе **NL**.

$$\log 2^{s(n)} \approx s(n).$$

Поэтому теперь мы можем запустить алгоритм из прошлой теоремы, единственное, что нужно изменить — теперь не нужно читать граф с входной ленты, а нужно проверять смежность двух конфигураций. ■

[4]Это тоже нужно сертифицировать и об этом будет написано ниже.