Математическая Логика и Теория Алгоритмов

Конспект лекций по курсу

Руслан Кулагин

весна 2021

Дисклеймер

Данный конспект не претендует на полноту и корректность. Это всего лишь авторские заметки, которые он пожелал выразить в форме tex файла. Каждый решивший использовать данный конспект для подготовки или изучения делает это на свой страх и риск. Автор рекомендует читать книги и слушать лекции. В случае обнаружения опечаток, помарок или ошибкок, буду рад письму на rus.kulagin2001@yandex.ru или Pull Request на github.

Colophon

Документ был набран при помощи KOMA-Script и LaTeX при использовании класса kaobook.
Исходный код доступен по ссылке:
https://github.com/RKulagin/MLTA
(Вы можете внести свой вклад в развитие этого конспекта!)

Теория NP-полноты | 1

1.1 Два подхода к сложности задачи

Первый подход предлагает считать сложностью алгоритма число тактов работы машины, реализующей этот алгоритм (число тактов работы в Машине Тьюринга, число совершённых подстановок в НАМ, число команд выполненных РАМ).

Второй подход предлагает рассматривать в качестве алгоритма булеву функцию, реализующую этот алгоритм, и в качестве сложности рассматривать число функциональных элементов в схеме из функциональных элементов, реализующих эту функцию. [1]

В любом случае понятно, что для разных индивидуальных задач I сложность будет разной.

Возьмём все индивидуальные задачи I с входом длины $|I| \leq m$. Число таких задач обозначим Z_n. Очевидно, что для конечного алфавита число Z_n — конечно.

Definition 1.1.1 Сложность в худшем назовём $t_Z(n) = \max_{|I|\leq n} t(I)$.

Definition 1.1.2 Сложностью в среднем назовём

$$\frac{\sum\limits_{|I|\leq n} t(I)}{Z_n}.$$

Для каждой задачи может быть сколь угодно много различных алгоритмов $A_Z^1, A_Z^2, \ldots$.

Definition 1.1.3 Сложность задачи — наилучшая из сложностей алгоритмов, решающих эту задачу.

Definition 1.1.4 Будем называть задачу трудно разрешимой, если у всех алгоритмов, решающих её, сложность по меньшей мере экспоненциальная. [1]

Для сравнения масштабов, если 1 такт занимает 10^{-6} секунд, то для входа длины 60 алгоритм со сложностью n^6 будет работать 13 минут, а со сложностью 2^n — 366 веков.

При этом развитие техники на время работы экспоненциальных алгоритмов никак влиять не будет. Даже при увеличении мощностей в 1000 раз, экспоненциальный алгоритм будет работать за адекватное время на задаче с длиной входа на 10 символов больше, чем сейчас.[2]

1: Этот подход рассматривает Глава 2.

Максимальное время работы алгоритма на индивидуальных задачах I с длиной входа $|I| \leq n$.

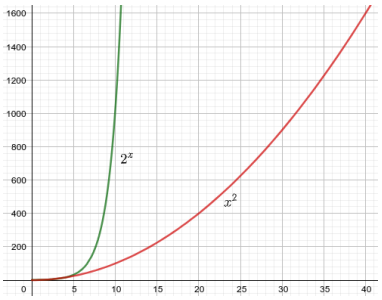

Рис. 1.1: График функций 2^x и x^2. Сравнение графиков функций x^2 и 2^x как пример, насколько сильна разница между экспоненциальной и полиномиальной сложностью. Сравните рост значений в окрестности 10, например.

2: Пусть сейчас наибольшая задача, которая решается за допустимое время имеет размер n, тогда при увеличении мощностей в 1000 раз мы имеем прирост в $\log_2 1000 \simeq 10$

1.2 Определение классов P, NP

Definition 1.2.1 Слово из входных символов допускается (воспринимается) тогда и только тогда, когда машина Тьюринга, начав работу в выделенном начальном состоянии, сделаем последовательность шагов, которые в конце концов приведут её в допускающее состояние.

Definition 1.2.2 Языком L(M), допускаемым машиной M, называют множество всех цепочек (слов) из входных символов, допускаемых M.

Definition 1.2.3 Класс P — это класс языков, допускаемых MT за полиномиальное время.

То есть MT допускает язык и её время работы ограничено полиномом.

Задачи (языки) из класса P называются полиномиально разрешимыми.

Definition 1.2.4 Класс NP — класс языков, допускаемых HMT за полиномиальное время.

Theorem 1.2.1 Если $Z \in NP$, то существует такой полином $p(n)$, что Z может быть решена на детерминированной MT за время $O(2^{p(n)})$.

Доказательство. Если Z решается на HMT за полином $q(n)$, то значит для каждой I найдётся отгадка U(I) после записи которой HMT будет работать как обычная MT и будет проверять отгадку. Время проверки отгадки, написанной угадывающей головкой — $O(q(n))$, а значит и длина отгадки — $O(q(n))$.

Для перебора всех отгадок в алфавите длины k потребуется $k^{q(n)}$ шагов, а значит в сочетании с проверкой всё займёт $O(k^{q(n)} * q(n)) = O(2^{p(n)})^*$. □

*: Просто возьмём достаточно большой полином

1.3 Полиномиальная сводимость

Definition 1.3.1 Язык $L_1 \subset A^*$ сводится к языку $L_2 \subset B^*$, если существует такое отображение $f : A^* \to B^*$, что выполняются два условия.

1. Функция f вычисляется за полиномиальное время. (Например, существует MT, которая за полиномиальное число тактов вычисляет эту функцию.)
2. Для любого входа $I \in A^*$ соотношение $I \in L_1$ выполняется тогда и только тогда, когда выполняется соотношение $f(I) \in L_2$.

Мы знаем, что произведение двух полиномов и полином от полинома также будет полиноном. Получается, если задача Z_1 имеет полиномиальную сложность $O(p(n))$, то, если Z_2 полиномиально сводится к Z_1 за $O(q(n))$, то сложность Z_2 — $O(p(q(n))$. Логично, что если задача может быть сведена к другой за время не меньшее, чем экспоненциальное время, то о принадлежности одним классам говорить нечего.

Полиномиальную сводимость будем обозначать $\leq_P$.

Lemma 1.3.1 Если $Z_1 \leq_P Z_2$ и $Z_2 \leq_P Z_3$, то $Z_1 \leq_P Z_3$

Доказательство. Пусть $Z_1 \leq_P Z_2$ путём отображения f, а $Z_2 \leq_P Z_3$ — g. Тогда рассмотрим суперпозицию gf, которое будет сводить Z_1

к Z_3. При этом, так как f и g полиномы, то и их композиция тоже будет полиномом. □

Lemma 1.3.2 Пусть $Z_2 \in P$, и $Z_1 \leq_P Z_2$. Тогда $Z_1 \in P$.

Доказательство. Пусть A и B — алфавиты языков L_1 и L_2, а функция $f : A^* \to B^*$ осуществляет полиномиальную сводимость L_1 к L_2.

$Z_2 \in P$, значит существует машина Тьюринга M_2, решающая Z_2 за $O(p(n))$ тактов. M_f — машина Тьюринга, реализующая функцию f, работающая за $O(q(n))$ тактов.

Тогда композиция Машин Тьюринга $M_1 = M_2 M_t$, которая сначала преобразует (сведёт) условие задачи Z_1 к условию Z_2, а потом решит её, будет работать за полином.

Если попросили решить задачу, которая может быть полиномиально сведена к уже известной задаче из P, то можно радоваться и писать решение за полином.

□

Theorem 1.3.3 Задача нахождения Гамильтонова Цикла[3] в графе сводится к задаче Комивояжёра.

Доказательство. На вход в задаче о Гамильтоновом цикле поступает граф G = (V, E). Требуется ответить на вопрос: существует ли в графе Гамильтонов цикл.

Входом в задаче Комивояжера является матрица расстояний

$$\left(a_{ij}\right)_{n\times n},$$

число B. Требуется ответить, существует ли обход[4] по всем вершинам суммарной длины $\leq B$.

3: Гамильтонов цикл — цикл в графе, который проходит по каждой вершине графа ровно один раз

$$n = |V|, B = n, a_{ij} = \begin{cases} 1, \{v_i, v_j\} \in G \\ 2, \{v_i, v_j\} \notin G \end{cases}.$$

Осталось доказать, что гамильтонов цикл в G существует тогда и только тогда, когда существует обход длины $\leq n$.

($\Longrightarrow$). Если гамильтонов цикл существует, то всем рёбрам этого цикла соответствует 1 в $\left(a_{ij}\right)$. А значит обход найдётся.

($\Longleftarrow$). Очевидно, что, если обход нашёлся, мы прошли по ребрам длины 1. Значит, в исходном графе мы прошли по существующим рёбрам. Так как обход подразумевает прохождение по всем вершинам, то мы нашли Гамильтонов цикл. □

Theorem 1.3.4 Задача КНФ-выполнимость полиномиально сводится к задаче о клике.

Доказательство. Прежде всего обозначим вход и вопрос для обеих задач.

КНФ-выполнимость на вход получает КНФ

$$K(x_1, \ldots, x_n) = \left(x_{i_1}^{\sigma_1} \cup x_{i_2}^{\sigma_2} \cup \ldots \cup x_{i_p}^{\sigma_p}\right) \& \ldots \left(x_{s_1}^{\kappa_1} \cup x_{j_2}^{\kappa_2} \cup \ldots \cup x_{j_s}^{\kappa_s}\right)$$

Количество скобок в К = m. Вопрос: существует ли такой набор

$$\begin{cases} x_1 = v_1 \\ x_2 = v_2 \\ \ldots \\ x_n = v_n \end{cases}$$

такой, что $K(v_1, v_2, \ldots, v_n) = 1$. Другими словами, ответ "да", если К не тождественный 0.

Задача о клике получает на вход граф $G = (V, E)$, а также число B. Вопрос: существует ли клика, размера $\geq B$ [5] .

4: посещение каждой вершины хотя бы по одному разу

5: Клика — полный подграф.

Пусть V — множество всех литералов[6] в КНФ. При этом даже если x_1 содержится в 1й скобке и во 2-й это будут два разных литерала. Примем B = m. $\{v_i, v_j\} \in G \Leftrightarrow v_i и v_j$ находятся в разных скобках и соответствуют либо разным переменным, либо одинаковым переменным в одинаковой степени.

Докажем, что КНФ выполнима тогда и только тогда, когда построенный нами граф имеет клику.

($\Longrightarrow$). Разложим КНФ на дизъюнкцию мономов. Если КНФ выполнима, то хотя бы один из мономов обращается в 1. Рассмотрим этот моном. Очевидно, что все его литералы были взяты из разных скобок и, так как он обращается в 1, там нет конъюнкции двух одинаковых переменных в разных степенях (иначе он был бы тождественным 0). Получается, что для всех вершин, соответствующих литералам в этом мономе, проведены рёбра между ними. Значит, этому моному соответствует клика. Размер монома — m, значит и размер клики тоже m.

($\Longleftarrow$). Пусть в сгенерированном графе нашлась клика размерности m. Значит, ей соответствуют m литералов из разных скобок, которые образуют моном. Так как произведений $x_i \& \overline{x_i}$ быть не может по условию построения графа, значит этот моном может быть обращён в единицу, а значит КНФ выполнима.

□

1.4 Класс NPC

Definition 1.4.1 Класс NPC (NP Complete) — класс таких задач, что для любой задачи $Z \in NPC$ выполнено условие, что $Z \in NP$ и $\forall Z' \in NP, Z' \leq_P Z$.

Lemma 1.4.1 Если $Z \in NPC$ и $Z \leq_P Z'$, то $Z' \in NPC$

Доказательство этого факта очевидно и прямо следует из определения.

Theorem 1.4.2 Задача о КНФ-выполнимости — NPC.

Доказательство. В данном доказательстве мы не можем пользоваться полиномиальной сводимостью к другим NPC задачам. Только определением класса NP. Задачу КНФ-выполнимость будем сокращать в этом доказательстве как КНФ-в.

По сути мы должны доказать, что $\forall Z' \in NP$, $Z' \leq_P$ КНФ-в.

Задачу Z' можно решить за полином на НМТ, для простоты рассуждений возьмём в качестве полинома — n^k, где n — длина входа $|I|$.

Понятно, что если задача решается за время, меньшее, чем n^k, то она может быть решена и за n^k.

Зададим часть ограничений на нашу НМТ. Пусть она останавливается только в одном конечном состоянии q_{final}

Рассмотрим ту часть работы НМТ, когда она работает как обычная МТ[7] , то есть отгадка уже написана.[8] .

Тогда мы за n^k шагов должны дойти до конечного состояния q_{filal}, проверив отгадку.

Представим работу МТ в виде таблицы, в i-й строке которой записано i-е слово конфигурации этой МТ.

Данная таблица конфигурации будет соответствовать МТ при выполнении 4-х условий одновременно.

1. В каждой ячейке содержится один и только один символ из набора $A \cup S \cup \{\lambda\}$
2. В 0-й строке находятся $-n^k - m$ символов λ после идёт m символов отгадки $u_m, \ldots, u_1$, начальное состояние q_0, l символов условия $v_1, \ldots, v_l$, а после $n^k - l$ символов λ
3. В n^k-й строке есть ячейка, содержащая q_{final}
4. k-я и $(k+1)$-я строки различаются не более чем окошком 2×3, имеющем вид одной из трёх таблиц, которые соответствуют переходам $qb \to q'cR$; $qb \to q'cSt$; $qb \to q'cL$:

№	$-n^k \ldots$	$-m$	$\ldots$	-1	0	1	$\ldots$	l	$\ldots n^k$
0	$\lambda \ldots \lambda$	u_m	$\ldots$	u_1	q_0	v_1	$\ldots$	v_l	$\lambda \ldots \lambda$
n^k	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	q_{final}	$\ldots$	$\ldots$

a	q	b
a	c	q'

a	q	b
a	q'	c

$a, b, c \in A; q, q' \in S$

Если мы сможем перевести эти четыре утверждения в булеву функцию, то значит мы свели NP-задачу к КНФ-в., потому что эти 4 условия выполняются тогда и только тогда, когда МТ успешно проверяет отгадку, а значит ответ на задачу I — да.

Пусть переменная $x_{i,j,\sigma} = 1$, если в ячейке (i, j) таблицы конфигураций содержится символ σ и $= 0$ во всех остальных случаях.

6: Будем разделять литералы и переменные в этом доказательстве. Литералы — конкретные переменные в конкретной степени в конкретной скобке КНФ.

7: Далее под работой МТ будет пониматься именно эта часть

8: Угадывающая головка НМТ работает также за $O(n^k)$

Первая формула запишется как

$$\phi_{line} = \bigcap_{-n^k \le i,j \le n^k} \left(\left(\bigcup_{\sigma \in A \cup S \cup \{\lambda\}} \left(x_{i,j,\sigma}\right) \right) \cap \left(\bigcap_{\substack{\sigma \in A \cup S \cup \{\lambda\}, \\ \xi \in A \cup S \cup \{\lambda\}, \\ \sigma \neq \xi}} \left(\overline{x_{i,j,\sigma}} \cup \overline{x_{i,j,\xi}} \right) \right) \right).$$

Первая часть отвечает за то, чтобы каждая клетка содержала хотя бы один символ из допустимого множества, вторая часть, чтобы в каждой клетке находилось не более одного символа из множества допустимых значений.

Второе утверждение может быть записано формулой:

$$\phi_{start} = \left(\&_{\substack{-n^k \le j < -m \text{ или} \\ l < j \le n^k}} x_{0,j,\lambda} \right) \& \left(\&_{\substack{-m \le j < 0 \text{ или} \\ 0 < j \le l}} \left(\cup_{\sigma \in A} x_{0,j,\sigma} \right) \right) \& \left(x_{0,0,q_0} \right).$$

Форма записи через или является, возможно, не самое элегантной, зато экономит время.

Третье утверждение записывается формулой так:

$$\phi_{accept} = \cup_{-n^k \le j \le n^k} (x_{n^k,j,q_{final}}).$$

По сути эта формула аналогична фрагменту ϕ_1, мы проверяем, что нашёлся хотя бы один элемент q_{final} в последней строке. Тот факт, что q в каждой строке может быть только один, следует из утверждений 1 и 4.

Утверждение 4 обобщённо будет записано так:

$$\phi_{compute} = \&_{i \le i \le n^k} (\phi_{i,(i+1)}).$$

$\phi_{i,(i+1)}$ = (в j-м столбце i-й строки стоит символ состояния)$\cdot$

$$\cdot \left(\bigcup_{u=\overline{1,p}} \left(O_{i,(i+1)} \sim O_u \right) \right).$$

O_p — окошко 2×3 , соответствующее p-й команде. $O_{i,(i+1)}$ — окошко, показывающее различие строк i и i+1

Длина $|\phi_{i,(i+1)}| = O(n^k)$

$$|\phi_0 \cdot \phi_{start} \cdot \phi_{accept} \cdot \phi_{copmute}| = O(n^{2k}.$$

Если ответ да, то существует требуемая таблица выполнимости, и есть в этой таблице финальное состояние, показывающее, что ответ существует. Значит, формула выполнима. □

Theorem 1.4.3 Если $Z \in NPC$ и дополнение Z — $Z^* \in NP \implies co - NP = NP$

Доказательство. $Z_1 \leq_P Z_2 \implies Z_1^* \leq_P Z_2^*$.

Пусть $Z \in NPC, Z^* \in NP$, возьмём $W \in NP$. $W \leq_P Z \implies W^* \in Z^*$. Получается, $W^* \in NP$. Значит $NP = Co - NP$. □

1.5 Методы доказательства NP полноты

Сужение задачи.

Метод заключается в установлении того, что задача Z включает в качестве частного случая задачу $Z' \in NP$. При этом даже не обязательно, чтобы Z' была полной копией частного случая, может возникнуть и очевидная биекция.

Примеры можно увидеть в [2] на страницах 85–88.

Локальная замена.

Выбирается некоторое характерное свойство известной NPC задачи, с помощью него образуется семейство основных модулей, а соответствующие индивидуальные задачи получаются путём единообразной замены каждого основного модуля некоторой другой структурой.

Например, КНФ-выполнимость может быть сведена к КНФ-3-выполнимости[9] .

a	q	b
q'	a	c

Theorem 1.5.1 Задача 3-выполнимость является NPC.

Доказательство. Доказательство можно посмотреть в [2] на страницах 67–69. □

Построение компоненты.

Основная идея: с помощью составных частей рассматриваемой задачи конструируем некоторые "компоненты", соединяя которые можно "реализовать" индивидуальные задачи известной NPC задачи.

Доказательства этим методом является самым сложным, поэтому приводить пример не буду (по крайней мере пока).

1.6 Практическое приложение теории P–NP

Пусть дана задача Z в форме распознавания, которую нужно решить.

- $Z \in P$. Значит для задачи существует полиномиальный алгоритм, который надо найти в источниках и реализовать.
- $Z \in NPC$. Полниомиального алгоритма за 200 лет не нашли, поэтому нужно искать какие-то частные решения или пробовать не точные алгоритмы (об этом в следующей части).
- Если неизвестно, что $Z \in P$ или $Z \in NPC$, то всё будет зависеть от конкретной задачи. В классе NPI (NP Intermediate) не так много задач, имеющих осмысленное приложение. Частные случаи NPC чаще всего будут иметь либо очевидное полиномиальное решение, либо не иметь его совсем.

Схемы из Функциональных Элементов 2

2.1 Определения

Definition 2.1.1 Дерево — связный граф без циклов.

Definition 2.1.2 Корневое дерево — это дерево, у которого помечена одна вершина.

Definition 2.1.3 Помеченное дерево — дерево, все вершины которого помечены (пронумерованы).

При разговоре о изоморфизме, стоит понимать, что два дерева могут быть изоморфны, но при этом их помеченные версии равно как и корневые могут быть не изоморфны.

Также в этом параграфе приведём утверждения, известные из курса Теории Графов.

Proposition 2.1.1 В любом конечном связном графе существует подграф, содержащий все вершины исходного графа и являющийся деревом.

Доказательство. Из каждого цикла будем удалять по одному ребру, пока циклов не останется. Так как мы разрезаем циклы, то связность от этого рушиться не будет. Так как циклов конечное число, процесс когда-нибудь остановится. □

Corollary 2.1.2 Всякий связный граф можно получить из дерева достройкой рёбер.

Proposition 2.1.3 В дереве из n вершин $(n-1)$ ребро.

Definition 2.1.4 Граф называется ориентированным, если каждому его ребру приписано направление. Также такой граф сокращённо называется орграфом.

Для каждого ребра одну вершину будем называть началом, другую — концом. Направление ребра идёт от начала к концу.

Definition 2.1.5 Ориентированный цикл — цикл в ориентированном графе, в котором все рёбра направлены в одну сторону, т.е. конечная последовательность ориентированных рёбер, где конечная вершина совпадает с начальной.

Definition 2.1.6 Степень захода вершины в орграфе — количество рёбер, конец которых находится в данной вершине (направленных в данную вершину).

Степень исхода вершины в орграфе — количество рёбер, начало которых находится в данной вершине.

Lemma 2.1.4 В орграфе без ориентированных циклов существует вершина со степенью исхода 0.

Доказательство. Докажем от противного. Допустим, таких вершин не существует, то есть каждая вершина обязательно является началом для какого-либо ребра. Возьмём случайную вершину и начнём идти по рёбрам. В какой-то момент мы упрёмся в вершину, в которой мы уже были (вершин конечное число, а значит бесконечно в новые мы идти не сможем), получается, в графе есть орцикл. Противоречие. □

Theorem 2.1.5 В любом конечном орграфе из n вершин без ориентированных циклов можно занумеровать вершины первыми n натуральными числами так, чтобы для каждого ребра номер конца был больше, чем номер начала.

Доказательство. Докажем по индукции.

База. Граф из одной вершины можно пометить очевидным образом.

Пусть нам нужно пометить орграф из n вершин, но мы знаем, что пометить можно любой орграф без орциклов из n-1 вершины. Выберем в графе из n вершин такую, у которой степень исхода 0 (Lemma 2.1.4). Удалим её из графа, получим помеченный граф из n-1 вершины, удалённую вершину пометим как n. □

Definition 2.1.7 Схема из функциональных элементов (СФЭ) — конечный ориентированный граф без ориентированных циклов, все вершины которого имеют степени захода не более 2. Вершинам степенью захода 0 приписывается символ-переменная x_i, вершины со степенью захода 1 получают символ $\neg$, вершины со степенью захода 2 помечаются $\wedge$ или $\vee$. Некоторым вершинам приписывается $*$.

Занумеруем вершины графа согласно Theorem 2.1.5. Каждой вершине СФЭ можно сопоставить некоторую булеву функцию по следующему правилу. Пусть всем вершинам с номерами $1, \ldots, n-1$ уже соответствуют булевы функции. Если у n-й вершины степень захода 0, то ей приписана переменная, которую мы и поставим ей в соответствие. Если у n-й вершины степень захода 1, то ей соответствует отрицание функции той вершины, из которой исходит ребро. Если у n-й вершины степень захода 2, то ей будет соответствовать конъюнкция или дизъюнкция функций вершин, из которых исходят рёбра. По правилу нумерации из Theorem 2.1.5 у вершин, из которых исходят рёбра номера меньше, чем у вершин, в которых рёбра входят, значит функции для них уже известны.

2.2 Оценки

Сейчас может быть непонятно, зачем мы приводим некоторые оценки, но они понадобятся в Section 2.3.
В этом разделе речь идёт о связных графах.

Proposition 2.2.1 Имеет место оценка $C_n^k \leq 2^n$.

Theorem 2.2.2 Пусть $\delta^*(q)$ количество неизоморфных корневых деревьев с q рёбрами. Тогда имеет место оценка

$$\delta^*(q) \leq 4^q.$$

Доказательство. Пусть дано корневое дерево. Расположим его по ярусам так, чтобы корень находился в самом низу, а концы всех рёбер находились на соседних ярусах, и зафиксируем эту расстановку. Начинаем ходить по дереву по следующему алгоритму:

1. Если из данной вершины есть ребро с концом в вершине ярусом выше, в которой мы ещё не были, то поднимаемся в эту вершину. Если таких ребёр несколько, то выбираем ребро с самой левой вершиной среди тех, в которых мы ещё не были.
2. Если для данной вершины не осталось рёбер, по которым мы не ходили, спускаемся вниз по ребру, которое соединяет эту вершину и вершину на более низком ярусе.
3. Если мы обошли все рёбра сверху и спуститься не можем, значит мы вернулись в корень и можем остановиться.

Каждому нашему подъёму поставим в соответствие последовательность из 0 и 1. 1 будем писать тогда, когда мы поднимаемся на ярус выше, 0, когда спускаемся. Так как по каждому ребру мы пройдём два раза, то у нас будет записано $2q$ цифр. Количество способов записать последовательность последовательность из $2q$ нулей и единиц — $2^{2q} = 4^q$. Это число не меньше, чем число корневых деревьев. Теорема доказана. □

Theorem 2.2.3 Количество неизоморфных деревьев с q рёбрами $\delta(q) \leq 4^q$.

Доказательство. Очевидно, что количество неизоморфных корневых деревьев больше, чем число неизоморфных деревьев, так как из любого дерева можно получить корневое выбрав одну из n вершин в качестве корня. Следовательно, из Theorem 2.2.2 получаем, что $\delta(q) \leq \delta^*(q) \leq 4^q = 4^{n-1}$ □

Theorem 2.2.4 Количество неизоморфных деревьев с q рёбрами оценивается снизу:

$$\delta(q) \geq \frac{1}{2\sqrt[3]{2}} \left(\sqrt[3]{2}\right)^q.$$

Доказательство. Рассмотрим деревья только определённого вида. Понятно, что количество таких неизоморфных деревьев будет оценкой снизу для общего числа неизоморфных деревьев.

Вид деревьев представлен на рисунке 2.1

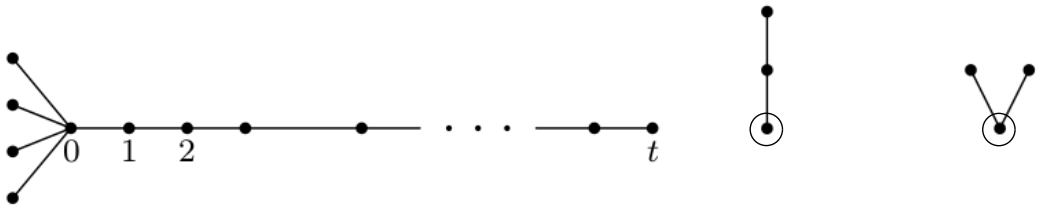

Рис. 2.1: Вид дерева и присоединяемых поддеревьев.

На каждую из вершин $1, \ldots, t$ добавляется один из подграфов, причём совмешается помеченная и обведённая в кружок вершины.

Посчитаем число рёбер в этом графе. $q = 4 + 3t$. 4 ребра исходят из 0, а из остальных t вершин можно считать, что исходит ребро, соединяющее $t-1, t$ и два ребра из присоединённого подграфа.

Понятно, что к каждой из t вершин можно присоединить один из двух подграфов, получается, мы получим 2^t неизоморфных дерева.

Из того, что $t = \frac{q-4}{3}$, выражаем число неизоморфных деревьев через число рёбер:

$$2^t = 2^{\frac{q-4}{3}} = \frac{\left(\sqrt[3]{2}\right)^q}{2\sqrt[3]{2}} \leq \delta(q).$$

□

Theorem 2.2.5 Число неизоморфных графов с n вершинами и m рёбрами

$$\gamma(n, m) \leq n^{m-n} \cdot A^{n+m},$$

$A = const$

Доказательство. Число неизоморфных графов можно оценить сверху как произведение числа неизоморфных деревьев на n вершинах (это уже n-1 ребро, Theorem 2.2.3) и числа способов построить необходимое $m-n+1$ ребро. Количество способов построить k рёбер можно оценить сверху как $\overline{C}_n^k \cdot n \cdot k$ (выбираем один конец ребра и для него выбираем второй конец ребра). Получаем формулу

$$\gamma(n, m) \leq \delta(n-1) \cdot \overline{C}_n^{m-n+1} n^{m-n+1} \leq 4^{n-1} \cdot C_{m+1}^n \cdot n^{m-n+1} \leq$$

$$\leq 4^{n-1} \cdot 2^{m+1} n n^{m-n} \leq 4^{n-1} \cdot 2^{m+1} 2^n n^{m-n} \leq 8^n 8^m n^{m-n} \leq A^{n+m} n^{m-n}.$$

□

2.3 Функция Шеннона

Каждой булевой функции можно поставить в соответствие какой-то функциональный элемент. Если функция от k переменных, то функциональный элемент будет иметь k входов и 1 выход.

Базисом будем называть какое-то множество булевых функций.

Definition 2.3.1 Базис называется полным, если множество функций, которые он содержит полное по Посту.

Theorem 2.3.1 Теорема Поста. Система булевых функций F является полной тогда и только тогда, когда она целиком не принадлежит ни одному из замкнутых классов T_0, T_1, S, M, L.

Если базис полный, то любую функцию можно реализовать в виде формулы в этом базисе.

Будем говорить, что СФЭ S_f реализовывает булеву фукнцию f.[1]

1: Одну и ту же фуннкцию даже в одинаковом базисе могут реализовывать разные СФЭ.

Сложностью схемы из функциональных элементов называется количнство элементов в этой схеме.

$L_B(f)$ — сложность минимальной схемы для f в базисе B.

$L_B(n)$ — функция Шеннона — количество элементов в схеме, которая реализовывает любую функцию от n в базисе B. $L_B(n) = \max\limits_{f \in B_n} L_B(f)$.

Theorem 2.3.2 Если B_1, B_2 — два полных базиса, то $L_{B_1}(n)$ и $L_{B_2}(n)$ отличаются не более чем в константу.

Иными словами

$$L_{B_1}(n) \leq c_1 L_{B_2}(n), L_{B_2}(n) \leq c_2 L_{B_1}(n).$$

без доказательства

Theorem 2.3.3 Справедливо неравенство $L(n) \leq (n+1)2^{n-1} \leq n2^n$

Далее в конспекте будет идти речь о стандартном базисе $\{\neg, \wedge, \vee\}$ и буква B будет опускаться.

Доказательство. Рассмотрим СДНФ булевой функции. По своей форме СДНФ — дизъюнкция мономов, которые являются конъюнкциями n переменных, каждая из которой может быть в степени 1 или 0. Таких мономов не более 2^n, в каждом мономе $(n-1)$ конъюнкций. С отрицаниями поступим просто: для каждой переменной сразу получим её отрицание и там, где оно будет нужно, будем использовать его, таким образом мы используем ещё n функциональных элементов. Получается, каждую булеву функцию можно реализовать не более чем $(n-1)\,2^n + n$ элементами.

Теперь рассмотрим СКНФ булевой функции. При нахождении сложности реализации любой СКНФ можно провести аналогию с СДНФ. На место мономов из n-1 конъюнкции приходит полином из n-1 дизъюнкции, а эти полиномы между собой объединены конъюнкцией. Получается, любая функция реализовывается через СКНФ также не более чем $(n-1)\,2^n + n$ элементами.

Количество конъюнкций в СКНФ зависит от числа нулей в векторе значений, а количество дизъюнкций в СДНФ зависит от числа

единиц в векторе значений. Тогда давайте брать СКНФ, если нулей меньше, чем единиц, а СДНФ во всех остальных случаях. Получается, вместо 2^n мы можем взять 2^{n-1}. Получается, каждая функция может быть реализована не более чем $2^{n-1}(n-1)+n \le 2^{n-1}(n+1)$.

Неравенство $2^{n-1}(n+1) \le_2^n n$ я считаю очевидным. □

Дальнейшие оценки будут ассимптотическими.

Theorem 2.3.4 Сложность универсального конъюнктора от n переменных (СФЭ, которая принимает на вход n переменных, а на выходе имеет все возможные конъюнкции от n переменных) $C(n) \sim 2^n$.

Доказательство. Пусть сложность универсального конъюнктора от n переменных обозначается k_n. Давайте выразим k_n через $k_{\frac{n}{2}}$. Конъюнкций от n переменных — 2^n. Возьмём два универсальных конъюнктора: один от переменных $x_1, x_2, x_{\frac{n}{2}}$, другой от оставшихся переменных. Сложность каждого из них будет $k_{\frac{n}{2}}$, а теперь все $2^{\frac{n}{2}}$ выходов первого нужно попарно проконъюнктировать с выходами второго. Получается, нам потребуется $2^n + 2k_{\frac{n}{2}}$ функциональных элементов для этого.

$$k_n = 2^n + 2 \cdot 2^{\frac{n}{2}} + 2^2 \cdot 2^{\frac{n}{2^2}} + \ldots \sim 2^n.$$

□

Theorem 2.3.5 Справедливо равенство

$$L(n) \lesssim 12 \cdot \frac{2^n}{n}.$$

Доказательство. Будем реализовывать формулу вида

$$f(x_1, x_2, \ldots, x_n) = \bigcup_{\sigma_1,\ldots,\sigma_k} x_1^{\sigma_1} x_2^{\sigma_2} \cdot \ldots \cdot x_k^{\sigma_k} f(\sigma_1, \ldots, \sigma_k, x_{k+1}, \ldots, x_n).$$

Формулу от $q = n - k$ переменных мы гарантированно реализуем за $q2^q$ элементов. Theorem 2.3.4 говорит о том, что все конъюнкции от k переменных можно ассимптотически реализовать за 2^k элементов. Ещё нам понадобится $2^k - 1$ дизъюнктор и 2^k конъюнкторов (для склеивания f с конъюнкциями).

$$L(n) \lesssim 2^k - 1 + 2^k + 2^k + 2^{2^q} \cdot 2^q \cdot q.$$

Выберем $q = \lceil \log_2 n \rceil - 1$, получим оценку

$$L(n) \lesssim 3 \cdot 2^k + (\lceil \log_2 n \rceil - 1)\frac{n}{2} \cdot 2^{\frac{n}{2}} = 3 \cdot 2^{n-\lceil \log_2 n \rceil + 1} + \log_2 n \cdot \frac{n}{2} 2^{\frac{n}{2}} =$$

$$= \frac{3 \cdot 2 \cdot 2^n}{2^{\lceil \log_2 n \rceil}} + \log_2 n \cdot \frac{n}{2} 2^{\frac{n}{2}} \lesssim \frac{3 \cdot 2 \cdot 2 \cdot 2^n}{n} + \log_2 n \cdot \frac{n}{2} 2^{\frac{n}{2}} \lesssim 12 \cdot \frac{2^n}{n}.$$

□

Theorem 2.3.6 (О. Б. Лупанов) Справедливо неравенство

$$L(n) \lesssim \frac{2^n}{n}.$$

Definition 2.3.2 Пусть $M(n)$ — количество объектов, а $M_s(n)$ — количество объектов, обладающих каким-то свойтсвом s. Будем говорить, что почти все объекты из $M(n)$ обладают свойством s, если

$$\lim_{n\to\infty} \frac{M_s(n)}{M(n)} = 1.$$

Theorem 2.3.7 Справедливо неравенство

$$L(n) \gtrsim \frac{2^n}{n}.$$

Доказательство. Введём следующие обозначения:

- $P_2^*(n)$ — количество булевых функций существенно зависящих от n переменных
- $N(n,h)$ — количество функций существенно зависимых от n переменных и реализуемых СФЭ не более чем за h элементов.
- $N'(n,h)$ — количество функций существенно зависимых от n переменных и реализуемых СФЭ ровно за h элементов.
- $N''(n,h)$ — количество СФЭ сложности h для функций существенно зависимых от n переменных.

Всего от n переменных 2^{2^n} функций. Число зависимых от n переменных булевых функций можно оценить следующим образом: $|P_2^*(n)| \gtrsim 2^{2^n} - n2^{2^{n-1}}$, понятно, что $n2^{2^{n-1}} = o(2^{2^n})$, а значит

$$|P_2^*(n)| \sim 2^{2^n}.$$

Очевидно, что $N = N'$ (можно дополнить схему ничего не делающими элементами). Также очевидно, что $N\ 0leN''$.

Покажем, что $\forall \epsilon > 0, h_0 := (1-\epsilon)\frac{2^n}{n}$

$$\frac{N''(n,h_0)}{P_2^*(n)} \to_{n\to\infty} 0,$$

то есть почти все функции от n существенных переменных не могут быть реализованы за h_0. Получается, по определению функции Шеннона $h_0 \le L(n)$, а значит $L(n) \ge \frac{2^n}{n}(1-\epsilon)$, $L(n) \gtrsim \frac{2^n}{n}$.

Величину N будем оценивать, мажорируя её N". Для этого нам нужно посчитать число графов, которые содержат $(n+h)$ вершин (n входов и h ФЭ) и q рёбер. $N(n,h) = \sum_{q=1}^{2h} N''(n,h,q)$. Составляющие суммы будут складываться из:

- $\gamma(h+n,q) = A^{h+n+q} \cdot (h+n)^{q-h-n}$ — числа неизоморфных графов с h+n вершинами и q рёбрами
- $(h+n)^n$ — оценка сверху количества способов выбрать вход СФЭ
- (h+n) — число способов выбрать выход
- 2^q — выбор ориентации рёбер

- 3^h — назначение СФЭ ($\neg, \wedge, \vee$)

$N''(n,h,q) \leq A^{h+n+q}(h+n)^{q-h-n}(h+n)^{n+1}2^q 3^h \leq B^{h+n+q}(h+n)^{q-h+1}$.

$q \leq 2h$

$$N''(n,h,q) \leq B^{3h+n}(h+n)^{h+1}.$$

Теперь получим оценку для $N''(n,h)$, учитывая, что меньше, чем h рёбер быть не может.

$$N''(n,h) \leq \sum_{q=h}^{2h} B^{3h+n}(h+n)^{h+1} = B^{3h+n}(h+1)(h+n)^{h+1} \leq (C(h+n))^{h+n}.$$

Докажем, что $\frac{(C(h+n))^{h+n}}{2^{2^n}} \to 0$ при $h_0 = (1-\epsilon)\frac{2^n}{n}$

$$\log_2\left(\frac{(C(h+n))^{h+n}}{2^{2^n}}\right) = (h+n)\left[\log_2 C + \log_2(h+n)\right] - 2^n \lesssim$$

$$\lesssim (1-\epsilon)\frac{2^n}{n}\left[\log_2\left((1-\epsilon)\frac{2^n}{n} + n\right)\right] - 2^n \lesssim$$

$$\lesssim (1-\epsilon)\frac{2^n}{n}\left[n - \log_2 n\right] - 2^n \lesssim (1-\epsilon)2^n - 2^n = -\epsilon 2^n \to -\infty.$$

Очевидно, что если $\log_2 x \to -\infty$, то $x \to 0$.

□

Логика предикатов 3

3.1 Определения

Definition 3.1.1 n-местный предикат — это отношение на декартовом произведении n множеств.

Если говорить простым языком, предикат это некая функция $A_1 \times A_2 \times \ldots \times A_n \to \mathbb{B}$.

При этом множества $A_1, A_2, \ldots, A_n$ могут быть различными, но если взять $M = A_1 \cup A_2 \cup \ldots \cup A_n$, то можно говорить о n-местном предикате на M^n

Одноместный предикат — свойство[1] , двухместный[2] — бинарное отношение, известное из предыдущих курсов.

1: Например, P(x) = x — чётное число
2: Например, $S(x, y) = x < y$

Definition 3.1.2 Пусть $A(x, y_1, \ldots, y_n)$ — некоторый предикат от $n+1$ переменной $x, y_1, \ldots, y_n$. Высказывание "$A(x, y_1, \ldots, y_n)$ истинно для всех x" (для любого, для каждого) обозначается символом $(\forall x)A(x, y_1, \ldots, y_n)$. Квантор $\forall$ называется квантором всеобщности, переход от предиката $A(x, y_1, \ldots, y_n)$ к $(\forall x)\,A(x, y_1, \ldots, y_n)$ называется навешиванием квантора всеобщности.

Выражение $(\forall x)A(x, y_1, \ldots, y_n)$ зависит только от переменных $y_1, \ldots, y_n$, которые называются свободными, является n местным предикатом. При этом это выражение будет истинно на произвольном наборе свободных переменных тогда и только тогда, когда оно истинно для любого значения x.

Definition 3.1.3 Пусть $A(x, y_1, \ldots, y_n)$ — некоторый предикат от $n+1$ переменной $x, y_1, \ldots, y_n$. Высказывание "$A(x, y_1, \ldots, y_n)$ истинно при некотором x" обозначается символом $(\exists x)A(x, y_1, \ldots, y_n)$. Квантор $\exists$ называется квантором существования, переход от предиката $A(x, y_1, \ldots, y_n)$ к $(\exists x)\,A(x, y_1, \ldots, y_n)$ называется навешиванием квантора существования.

Существует x для которого $A(x, y_1, \ldots, y_n)$

Выражение $(\exists x)\,A(x, y_1, \ldots, y_n)$ является n-местным предикатом, переменная x называется связанной, выражение истинно для данного набора свободных переменных тогда и только тогда, когда существует хотя бы одно значение связанной переменной, при котором оно истинно.

Связанная переменная x и переменная x, которая будет использована в другой части формулы это разные буквы, например, в $P(x) \vee (\forall x)Q(x, y)$ x в P(x) и x рядом с квантором — это разные x.

Буквы начала латинского алфавита (в том числе с числовыми индексами) — предметные константы. Буквы конца латинского алфавита (в том числе с числовыми индексами) — предметные переменные. Буквы A_i^n с числовыми индексами $i > 1, n > 0$ называются предикатными буквами. Буквы f_i^n — функциональные буквы. Верхний индекс предикатной и функциональной буквы обозначает число переменных, нижний индекс служит для различения букв.

функциональные буквы обозначают обычные функции

Функциональные буквы, применённые к предметным переменным и константам порождают термы.

Definition 3.1.4 Определение терма:

1. всякая предметная переменная или предметная постоянная есть терм;
2. если f_i^n функциональная буква и $t_1, t_2, \ldots, t_n$ — термы, то $f_i^n(t_1, \ldots, t_n)$ есть терм;
3. выражение является термом только в том случае, если это следует из двух условий выше (другими словами, других термов нет).

Definition 3.1.5 Если A_i^n — предикатная буква, а $t_1, \ldots, t_n$ — термы, то $A_i^n(t_1, \ldots, t_n)$ — элементарная формула.

Definition 3.1.6 Определение формулы:

1. всякая элементарная формула есть формула;
2. если $\mathcal{A}$ и $\mathcal{B}$ — формулы и y — предметная переменная, то каждое из выражений $(\neg\mathcal{A})$, $(\mathcal{A} \wedge \mathcal{B})$, $(\mathcal{A} \vee \mathcal{B})$, $\forall y\mathcal{A}$ есть формула;
3. выражение является формулой только в том случае, если это следует из двух предущих пунктов.

В пункте 2 использовалась система логических связок $\{\vee, \wedge, \neg\}$, однако также может быть использована любая другая полная систему функций.

Областью действия квантора $\forall y$ называется формула $\mathcal{A}$ в выражении $\forall y\mathcal{A}$.

Кванторы по старшинству операций располагаются после $\neg, \wedge, \vee$.

$\exists x\mathcal{A} = \neg(\forall x(\neg\mathcal{A}))$

Definition 3.1.7 Длина l(F) формулы F — это количество входящих в её запись букв и логических связок.

Definition 3.1.8 Вхождение переменной x в данную формулу называется связанным, если x является переменной входящего в эту формулу квантора $\forall x$, в противном случае вхождение — свободное.

Definition 3.1.9 Формула называется замкнутой, если она не содержит никаких свободных переменных.

3.2 Интерпретации. Выполнимость и истинность. Модели

Definition 3.2.1 Под интерпретацией понимается совокупность:

1. заданного непустого множества M, называемого областью интерпретации;
2. заданного соответствия:
   - Предикатным буквам A^n — некоторые n местные предикаты в M.

Например, зададим интерпретацию $\forall xA(x, y, b)$. Пусть M — множество целых чисел, b = 5, а предикатная буква A выражает предикат "x не меньше y + b".

- Функциональным буквам f^n — некоторые n аргументные функции (операции) в M, отображающие $M^n \to M$.
- Каждой предметной постоянной — некоторые предмет из M.
- Кванторам и связкам — их обычных смысл.

Замкнутая формула представляет собой высказывание, которое может быть либо истинно, либо ложно, а формула со свободными переменными выражает некоторое отношение на области интерпретации.

При этом для любой формулы можно построить бесчисленное множество разных интерпретаций.

Definition 3.2.2 Формула называется выполнимой в данной интерпретации, если она принимает значение Истина хотя бы для одной совокупности возможных значений свободных переменных (если они есть). Если формула не содержит свободных переменных, то она называется выполнимой в том случае, если принимает значение Истина в этой интерпретации.

Definition 3.2.3 Формула называется истинной в данной интерпретации, если она принимает значение Истина для всех возможных значений её свободных переменных, если они есть. Если же свободных переменных нет, то формула называется истинной, когда она принимает значение Истина в этой интерпретации.

Definition 3.2.4 Формула называется ложной в данной интерпретации, если она принимает значение Ложь для всех возможных значений её свободных переменных (если они есть). Если же свободных переменных нет, то формула называется ложной, когда она принимает значение Ложь в этой интерпретации.

Definition 3.2.5 Интерпретация называется моделью для данного множества формул Γ, если $\forall \mathcal{A} \in \Gamma$, $\mathcal{A}$ истинна в данной интерпретации.

Свойства[3] формул в заданной интерпретации:

1. A ложна в данной интерпретации $\leftrightarrow \neg A$ истинна в этой же интерпретации. И наоборот.
2. Никакая формула не может быть одновременно истинной и ложной в данной интерпретации
3. Если в данной интерпретации A — истинна и $A \to B$, то B тоже истинна
4. Формула $A \vee B$ выполнима в данной интерпретации, если хотя бы одна из них выполнима в данной интерпретации
5. Формула $\exists x A$ выполнима в данной интерпретации $\leftrightarrow$ A принимает значение Истина хотя бы для одной совокупности значений её свободных переменных и хотя бы одного значения переменной x.
6. Формула $\forall x A$ истинна в данной интерпретации $\leftrightarrow$ в этой интерпретации истинна A.

3: больше свойств приведено на страницах 59-62 Мендельсона.

Definition 3.2.6 Формула логики предикатов называется логически общезначимой, если она истинна в любой интерпретации.

Например, $\forall x A \to \exists x A$.

Definition 3.2.7 Формула логики предикатов A называется выполнимой, если существует интерпретация, в которой выполнима A.

Proposition 3.2.1 Формула A логически общезначима тогда и только тогда, когда $\neg A$ не является выполнимой.

Proposition 3.2.2 Формула A выполнима тогда и только тогда, когда $\neg A$ не является логически общезначимой.

Definition 3.2.8 Формула логики предикатов $\mathscr{A}$ называется противоречием, если формула $\neg\mathscr{A}$ является логически общезначимой. (Формула A ложна во всякой интерпретации.)

Definition 3.2.9 Формула логики предикатов A логически влечёт B, если в любой интерпретации формула B выполнена на всякой последовательности, на которой выполнена A. (B является логическим следствием A).

Definition 3.2.10 Формулы логики предикатов A и B называются равносильными (эквивалентными), если каждая из них логически влечёт другую. Обозначается $A \sim B$.

Theorem 3.2.3 Формула A логически влечет формулу B тогда и только тогда, когда формула $A \to B$ логически общезначима.

Theorem 3.2.4 Формулы A и B равносильны (логически эквивалентны) тогда и только тогда, когда формула $A \sim B$ логически общезначима.

Theorem 3.2.5 Если формула A логически влечёт формулу B и A истинно в данной интерпретации, то в этой же интерпретации истинно и B.

Definition 3.2.11 Формула B называется логическим следствием формул $A_1, A_2, \ldots, A_n$, если в любой интерпретации формула B принимает значение Истина при каждой совокупности значений свободных переменных (входящих в B и $A_1, \ldots, A_n$ при которых одновременно все формулы $A_1, \ldots, A_n$ приняли значение Истина.

Если формула записана в нормальной форме, то значит, что она записана в выбранном нами базисе, который мы называем нормальным базисом.

Theorem 3.2.6 Правило переименования связанных переменных. В любой формуле связанные переменные могут быть заменены на другие переменные (в кванторе и всюду в области действия квантора),так, чтобы не нарушалось определение формулы, в таком случае мы получим формулу эквивалентную исходной.

Доказательство. Пусть дана формула F. Пусть в этой формуле встречаются переменные (литералы) $x_1, x_2, x_1, x_2, x_1, x_5, \ldots$ (связанные переменные будем записывать только один раз, то есть из формулы $\forall x_1 P(x_1) \vee Q(x_1)$ мы выпишем x_1, x_1, первая - связанная, вторая свободная). Отметим там все вхождения литералов, обозначающие связанные переменные. Пусть мы отметили k литералов. Теперь составим множество всех литералов $x_1, x_2, \ldots, x_s$, и добавим к нему литералы $x_{s+1}, \ldots, x_{s+k}$. Проведём биекцию: первой отмеченной сопоставим x_{s+1}, второй — x_{s+2}, и т. д. Теперь пере обозначим все отмеченные буквы в формуле на те, которые мы поставили им в соответствие, учитывая, что, когда мы переименовываем связанную переменную, нужно переименовать и все её вхождения в её области действия. □

Definition 3.2.12 Формула исчисления предикатов следующего вида $F = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n G$, где $Q_1, \ldots, Q_n$ — кванторы, а формула G не содержит кванторов, называется формулой в предварённой (приведённой) нормальной форме.

Theorem 3.2.7 (без доказательства) Для каждой формулы логики предикатов в нормальной форме со связками $\{\neg, \rightarrow\}$ существует логически эквивалентная формула в нормальной форме со связками $\{\neg, \vee, \wedge\}$, имеющая ту же длину

Theorem 3.2.8 Для любой формулы логики предикатов существует эквивалентная ей формула в предварённой нормальной форме той же длины.

Theorem 3.2.9 Пусть F — формула длины l(F), F_1 — эквивалентная ей формула в приведённой нормальной форме длины $l(F_1)$, тогда $l(F) = C \cdot l(F_1)$.

3.3 Класс P/Poly

Definition 3.3.1 $P_{Poly} = \cup_{c \in \mathbb{N}} SIZE(n^c)$

$SIZE(n^c)$ — множество языков, разрешимых СФЭ полиномиальной сложности.

По сути класс P_{Poly}— семейство всех таких языков $L \subset \{0,1\}^*$, схемная сложность которых есть функция полиномиальной сложности.

Ключевой вопрос — как наращивать сложность, чтобы оценить, полиномиальная она или нет. Рассмотрим одноместный предикат, определённый на множестве двоичных слов. Каждое двоичное слово задаёт булеву функцию от разного числа переменных. Рассматривая такие булевы функции от 1й, 2х и т. д. переменных, функция Шеннона для которых будет ограничена полиномом.

Theorem 3.3.1 $P \subseteq P_{Poly}$

Доказательство. Пусть есть задача $Z \in P$, нужно показать, что эта задача будет лежать и в P_{Poly}.

Для любой индивидуальной задачи I существует таблица выполнимости, которая состоит из n^k строк (так как задача в P, то она разрешима на МТ за полином). Данной таблице выполнимости поставлена в соответствие КНФ $\phi = \phi_{start} \cdot \phi_0 \cdot \phi_{accept} \cdot \phi_{compute}$, длины $O(n^{2k})$ (см. доказательство Теоремы Кука). Значит, СФЭ этой функции также будет $O(n^{2k})$. Получается, для любой индивидуальной задачи можно построить СФЭ полиномиальной сложности, значит $Z \in P_{Poly}$. □

Theorem 3.3.2 $P \neq P_{Poly}$

Доказательство. Пусть $\phi(x) : \mathbb{N} \to 0, 1$ — алгоритмически неразрешимая задача. Мы знаем, что предикат принимает значения либо 0, либо 1. Тогда рассмотрим предикат $P(x_1, x_2, \ldots, x_n) = \phi(|x_1, x_2, \ldots, x_n|)$. Данный предикат не может быть алгоритмически разрешим, так как не существует Машины Тьюринга, которая могла бы вычислить значение ϕ. А значит он не лежит в $\mathcal{P}$. При этом $\forall n P(x_1, \ldots, x_n) = const = \phi(n), \forall x_1, \ldots, x_n$. Так как сложность реализации СФЭ константы всегда константа, то такой предикат P лежит в P_{Poly}. □

Proposition 3.3.3 Чтобы для $Z \in P_{Poly} \to Z \in P$, все $f_1(x_1), \ldots, f_n(x_1, \ldots, x_n)$ должны ограничиваться полиномами и для этой Z должен существовать алгоритм решения на Машине Тьюринга.

Theorem 3.3.4 Задача Z принадлежит классу P тогда и только тогда, когда выполнены два условия

1. $Z \in P_{Poly}$
2. Для $\forall f_n(x_1, \ldots, x_n)$ можно построить СФЭ полиномиальной сложности.

Доказательство. ($\Longrightarrow$). Если $Z \in P$, то по Theorem 3.3.1 выполняются оба условия. ($\Longleftarrow$) Так как $Z \in P_{Poly}$ мы за полином можем вычислить F_Z, то $Z \in P$ □

Классы PSPACE и NPSPACE | 4

4.1 Опеределение

В рамках рассмотрения классов PSPACE и NPSPACE, мы говорим о трёхленточной Машине Тьюринга, состоящей из входной, рабочей и выходной лент. Вся запись вспомогательной информации осуществляется только на рабочую ленту.

Definition 4.1.1 Классом PSPACE называется класс языков, допускаемых детерминированной Машиной Тьюринга, с длиной рабочей и выходной ленты, ограниченной полиномом от длины входной ленты.

Definition 4.1.2 Классом NPSPACE называется класс языков, допускаемых недетерминированной Машиной Тьюринга, с длиной рабочей и выходной ленты, ограниченной полиномом от длины входной ленты.

Definition 4.1.3 Классом $TIME(f(n))$ называется класс языков, допускаемых детерменированной машиной Тьюринга со сложностью (за время) ограниченной $f(n)$ от длины входа n.

Definition 4.1.4 Классом $SPACE(f(n))$ называется класс языков, допускаемых детерминирванной машиной Тьюринга с использованной памятью, ограниченной $f(n)$ от длины выхода n.

$$TIME(p(n) = P.$$

$$TIME(2^{p(n)}) = EXPTIME.$$

$$SPACE(p(n)) = PSPACE.$$

4.2 Отношения классов

Proposition 4.2.1 $P \subseteq PSPACE$

Доказательство. Если детерменированная машина Тьюринга работает за полином, то она сможет прочитать только полиномиальное число ячеек памяти. □

Proposition 4.2.2 $NP \subseteq PSPACE$

Доказательство. При доказательстве теоремы о том, что задачу, решаемую НМТ за $p(n)$, можно решить за $2^{q(n)}q(n)$ на детерминированной МТ, мы использовали факт перебора всех возможных отгадок, полиномиальной длины. Смоделируем работу такой детерминированной МТ следующим образом: в МТ входит генератор

всех слов в алфавите A, длина которых не превосходит $p(n)$ (таких слов $|A|^p(n)$, также входит проверка слова на соответствие определённым требованиям (предварительный синтаксический анализ), после генерации и проверки работает алгоритм исходной НМТ, проверяющий сгенерированное слово.

После проверки слово стирается и на его место записывается новое слово-отгадка. Такая МТ работает детерменированно, за экспоненциальное время, но с полиномиальным ограничением памяти. □

Proposition 4.2.3 $Co - NP \subseteq PSPACE$

Theorem 4.2.4 $PSPACE \subseteq EXPTIME$

Доказательство. Вспомним доказательство Теоремы Кука (Theorem 1.4.2), в которой мы для индивидуальной задачи задачи $Z \in PSPACE$ строим таблицу выполнимости. Количество ячеек в строке Машины Тьюринга ограничена полиномом (без потери общности n^k).

Пусть A — Алфавит, S — множество состояний, $|A \cup S| = m$, получается число слов длины n^k — m^{n^k}.

В таблице выполнимости нет двух повторяющихся строк, так как иначе Машина Тьюринга бы не останавливалась и таблица выполнимости не решала задачу. Получается, всего в таблице выполнимости строк $\leq m^{n^k}$. Так как число строк определяет время работы Машины Тьюринга, то $Z \in EXPTIME$ □

Definition 4.2.1 Задача Z называется $PSPACE$-полной, если $\forall Z' \in PSPACE : Z' \leq_P Z$

Сводимость по времени

4.3 Игра двух лиц

Рассмотрим множество индивидуальных задач I некоторой массоввой задачи Z. Пусть есть два игрока — белый (W) и чёрный (B), которые делают ходы поочереди. Число ходов заранее ограничено, ходы белого обозначаем $w_1, w_2, \ldots$, чёрного — $b_1, b_2, \ldots$.

Выигрыш будем определять по предикату $W(I, w_1, b_1, \ldots)$. В силу отсутствия ничьих, истинность предиката будет означать победу первого игрока, ложность — второго. При этом победа белого не должна зависеть от ходов чёрного, то есть предикат можно записать в следующей форме $\exists w_1 \forall b_1 \ldots \exists w_m \forall b_m W(I, w_1, b_1, \ldots, w_m, b_m)$.

L_W — множество слов I, на которых выигрывают белые, L_B — чёрные.

Theorem 4.3.1 Задача Z лежит в классе PSPACE тогда и только тогда, когда существует игра с полиномиальным от длины входа числом ходов и полиномиально вычислимым результатом такая, что $L_z = L_w$

Доказательство. Пусть число ходов ограничено q(n), A — алфавит задачи Z, $|A| = k$.

Сначала докажем $\Longleftarrow$, то есть что игра лежит в классе PSPACE. Построим корневое дерево T (пример представлен на рисунке 4.1), его корнем будет I^1 затем вершины каждого яруса отражают все возможные ходы первого и второго игроков поочерёдно. Каждой вершине можно приписать метку w или b.

1: условие игры, вход задачи

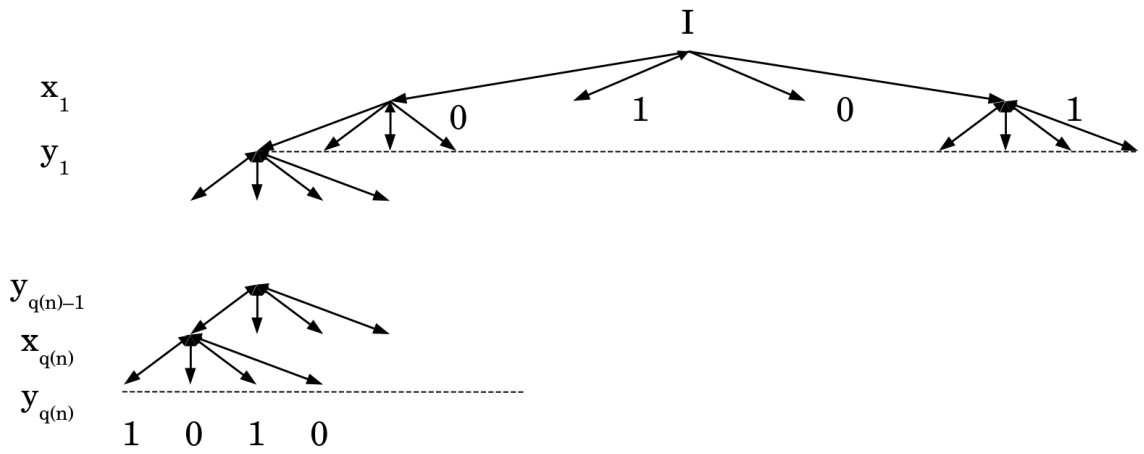

Рис. 4.1: Пример графа, отражающего игру двух лиц

Метки будут приписываться следующим образом: для самого нижнего яруса $q(n)$ хода чёрного значение однозначно задаётся предикатом $W(I, w_1, b_1, \ldots, w_{q(n)}, b_{q(n)})$. Дальше значения будут задаваться рекурсивно, когда мы будем подниматься на ярус выше:

- если ярус отражает ход белого, то он принимает значение w, если хотя бы один из детей (ходов) имеет пометку w, иначе ставим b;
- если ярус отражает ход чёрного, то он принимает значение w, если все дети (ходы) имеют пометку w.

Такой рекурсией мы однозначно получим значение у I, то есть восстановим исход игры на входе I.

Значит, для решения задачи Z нужно построить МТ, которая будет вычислять результат игры, воспроизводя описанный выше процесс. Число ярусов в дереве полиномиально, время проверки значения предиката также полиномиально $p(n)$, значит все слова имеют полиномиальную длину. После проверки мы будем их стирать. Получается, Машина Тьюринга имеет полиномиальную память $O(p(n)q(n))$.

($\Longrightarrow$). Пусть есть задача Z в классе PSPACE, покажем, что существует требуемая игра.

У нас есть МТ, которая с полиномиальной памятью распознаёт вхождение языка в L_Z, пусть используемый размер памяти не превышает n^k, построим таблицу выполнимости такой машины. Если Машина Тьюринга работает на алфавите A с множеством

состояний S, $|A \cup S| = m$, то строк в таблице не более $(m)^{n^k}$. Для простоты мажорируем это число при помощи 2^q

Суть игры будет состоять в следующем: белые утверждают, что для I ответ да, чёрные хотят это проверить. Понятно, что если для I ответ да, то таблица выполнимости будет существовать и будет без ошибок, если нет, то где-то в таблице выполнимости будет совершён недопустимый переход.

Белые своим ходом выписывают строку 2^{q-1}, чёрные выбирают, в какую из двух половин они двинутся, белые выписывают середину выбранного диапазона, чёрные снова выбирают. В какой-то момент диапазон будет состоять из двух подряд идущих строк. Если состояние одной строки можно получить из другой при помощи корректного перехода Машины Тьюринга (они отличаются только окошком 2×3, то значит таблица выполнимости корректна, белые победили, если нет, то таблица выполнимости некорректна, а значит для I ответ нет и победили чёрные. Так как и белые и чёрные играют по выигрышной стратегии, то в случае наличия некорректности мы к ней придём. □

Theorem 4.3.2 Задача проверки формулы исчисления предикатов на выполнимость PSPACE-полная.

Доказательство. Любой задаче из PSPACE мы ставим в соответствие игру двух лиц (Theorem 4.3.1), ответом да или нет на которую определяется по выполнимости соответствующего предиката. □

4.4 Отношение PSPACE и NPSPACE

Theorem 4.4.1 Задача Z лежит в классе $NPSPACE$ тогда и только тогда, когда существует игра с полиномиальным от длины входа числом ходов и полиномиально вычислимым результатом такая, что $L_z = L_w$

Доказательство. ($\Longleftarrow$). Если существует требуемая игра, то по Theorem 4.3.1 $Z \in PSPACE \implies Z \in NPSPACE$.

($\Longrightarrow$). Если $Z \in NPSPACE$, то рассмотрим игру аналогичную доказательству в Theorem 4.3.1 только теперь таблица выполнимости будет содержать помимо I ещё и отгадку. От увеличения длины рассматриваемых строк на полином используемая память также останется полиномиальной. □

Theorem 4.4.2

$$PSPACE = NPSPACE$$

Доказательство. Из Theorem 4.3.1 и Theorem 4.4.1 получаем в силу транзитивности, что $PSPACE = NPSPACE$. □

Theorem 4.4.3 (без доказательства)

$$NSPACE(f(n)) \leq SPACE(f^2(n))$$