ЛЕКЦИЯ 5

ВЫЧИСЛЕНИЯ С ОРАКУЛОМ

Вычисление с оракулом означает, что есть некоторое множество, принадлежность к которому можно установить за один шаг. Если множество разрешимо за длительное время, то наличие оракула существенно ускоряет процесс. Соответственно, если есть оракул, то всю теорию можно релятивизировать. С использованием оракула можно получить разрешимые перечислимые множества, множества, разрешимые за полиномиальное время, и аналог класса NP.

С помощью оракула можно определить верификаторы, которые будут определять сертификаты. При проверке верификаторы могут обращаться к оракулу. Таким образом, получается релятивизированная версия **задачи перебора**.

Теорема 5 (Бейкер, Джилл, Соловэй) Существует оракул A, при котором $\mathrm{P}^A = \mathrm{NP}^A$, а также существует оракул B, при котором $\mathrm{P}^B \neq \mathrm{NP}^B$. $*$

Док-во: В качестве оракула A, при котором $\mathrm{P}^A = \mathrm{NP}^A$, можно выбрать оракул экспоненциальных вычислений.

$$A = \mathrm{EXPCOUNT} = \{(M, x, 1^t) : M(x) = 1 \text{ и } M(x) \text{ работает не больше чем } 2^t \text{ шагов}\}.$$

$\mathrm{EXP} \subset \mathrm{P}^A$. Пусть язык $L \in \mathrm{EXP}$. Следовательно L распознается машиной M, которая работает не больше чем 2^{n^c} шагов. Тогда можно спросить оракул про тройку $(M, x, 1^{|x|^c})$. Если ответ оракула «да», то это означает, что $x \in L$. Если оракул отвечает «нет», то $M(x) = 0$.

Для любого оракула выполняется:

$$\mathrm{EXP} \subset \mathrm{P}^A \subset \mathrm{NP}^A \subset \mathrm{EXP}.$$

Докажем последний переход. Будем считать, что

$$L \in \mathrm{NP}^A \text{ значит, что } \exists\, V : x \in L \Leftrightarrow \exists\, s : V^A(x, s) = 1.$$

V работает не более чем $p(|x|)$ шагов. Следовательно, его запросы к оракулу не длиннее чем $p(|x|)$. За время $2^{p(|x|)}$ оракул может ответить на запрос. Всего запросов не более чем $p(|x|)$. Следовательно, общее время ответа не более чем $p(|x|) \cdot 2^{p(|x|)}$. Перебор всех

сертификатов — умножение на $2^{p(|x|)}$, откуда следует, что алгоритм завершится за экспоненциальное время. Значит, справедливо вложение:

$$\mathrm{EXP} \subset \mathrm{P}^A \subset \mathrm{NP}^A \subset \mathrm{EXP}.$$

Отсюда следует, что $\mathrm{P}^A = \mathrm{NP}^A$.

Докажем вторую часть теоремы. Построим оракул B так, что $\forall\, n |B \cap \{0,1\}^n| \leqslant 1$. Рассмотрим язык $L_B = \{1^n | B \cap \{0,1\}^n| = 1\}$. Тогда при всех B, $L_B \in \mathrm{NP}^B$. Можно построить B так, чтобы $L_B \notin \mathrm{P}^B$.

Пусть $M_1, M_2, \ldots$ — полиномиальные машины. B строится постепенно. К моменту рассмотрения каждой машины M_i значение B известно только в конечном числе точек. Рассмотрим работу машины $M_i(1^n)$, такой что:

1. В длине n неизвестна принадлежность ни одного слова.
2. Время работы $M_i(1^n)$ меньше чем 2^{n-1}.

Машина M_i делает запрос к B. Возможны два варианта.

1. Значение определено $\Rightarrow$ так и отвечаем.
2. Значение не определено $\Rightarrow$ отвечаем «нет».

Если M_i отвечает «да» $\Rightarrow$ все слова длины $n \notin B$. Значит, машина ошиблась. Если M_i отвечает «нет» $\Rightarrow$ про какое-то слово длины n машина M_i не спросила. Сделаем так, что это слово принадлежит B. Значит, машина снова ошиблась. Таким образом, можно сделать так, что все машины ошибутся.

Для каждой машины достаточно сделать один вход, на котором она ошибется. Процедура перебирает все машины, которые работают за полиномиальное время.

1. Пространственная сложность

Помимо времени работы важным ресурсом является использованная память. В последнее время стала популярна тема Big Data. Если нужно проанализировать большой объем данных, то не всегда возможно даже переписать весь вход на рабочую ленту. В этом случае приходится анализировать информацию частями. Например, оперативную память компьютера можно рассматривать как аналог рабочей ленты машины Тьюринга.

Когда измеряют пространственную сложность, не учитывают место, которое занимает вход. **Модуль вычислений** состоит из двух частей:

1. Лента, предназначенная только для чтения.
2. k рабочих лент.

Использованная память — количество измененных ячеек на рабочих лентах.

Рассмотрим базовые сложностные классы:

DSPACE($s(n)$) — множество языков, которые можно распознать на детерминированной машине, используя память $s(n)$.

DSPACE($s(n)$) — множество языков, которые можно распознать на недетерминированной машине, используя память $s(n)$.

$$\text{PSPACE} = \bigcup_{c=1}^{\infty} \text{DSPACE}(n^c).$$

$$\text{NPSPACE} = \bigcup_{c=1}^{\infty} \text{NSPACE}(n^c).$$

$$\text{L} = \text{DSPACE}(\log n).$$

$$\text{NL} = \text{NSPACE}(\log n).$$

Классы определяются с точностью до константы. В отличие от времени, где неизвестно, равны ли классы N и NP, классы PSPACE и NSPACE совпадают. Однако про классы L и NL это неизвестно.

Утверждение 3 PSPACE $\subset$ EXP. $*$

Док-во: Конфигурация — это состояние ленты, состояние машины и положение указателя. Состояний ленты всего $2^{s(n)}$, состояний машины — конечное число. Положений указателя порядка $s(n)$. Значит, всего возможных конфигураций будет $c \cdot (n) \cdot 2^{s(n)}$. Если $s(n)$ — полином, то количество всевозможных конфигураций — экспонента. Конфигурации не могут повторяться. Значит, машина не может работать более чем полученное время. Отсюда следует, что PSPACE $\subset$ EXP.

Утверждение 4 L $\subset$ P. $*$

Утверждение доказывается аналогично.

Теорема 6 (Сэвича) NSPACE$(s(n)) \subset$ DSPACE$(s(n)^2)$. $*$

Следствия: PSPACE = NSPACE. ■

Равенство PSPACE = NSPACE следует из того, что квадрат полинома есть полином. Однако про L и NL ничего нельзя сказать, т. к. квадрат логарифма не есть логарифм.

Докажем теорему.

Док-во: Рассмотрим конфигурационный граф. Вершины графа содержат конфигурации машины Тьюринга, т. е. содержимое ленты, состояние машины и положение указателя. Соединим две конфигурации, если из первой можно перейти во вторую в соответствии с командами машины. Заметим, что машина недетерминированная. Таким образом, вершины графа — конфигурации машины Тьюринга, ребра графа — ее переходы.

Обозначим C_{start} — начальная конфигурация, C_{accept} — принимающая конфигурация. Если машина детерминированная, то из каждой вершины графа выходит только одно ребро. Если машина недетерминированная, то из любой вершины могут выходить несколько ребер. Вопрос о том, лежит ли x в языке L, заключается в том, что существует ли путь из начальной конфигурации C_{start} в принимающую C_{accept}. Причем длина этого пути не больше чем количество вершин, т. е. не больше чем число конфигураций, равное $O(s(n) \cdot 2^{s(n)}$.

Рассмотрим **алгоритм проверки существования пути (рекурсивный)**. Обозначим алгоритм $R(u, v, i)$, где u, v — вершины графа, i — количество шагов. $R(u, v, i) = 1$, если существует путь из u в v длины не более чем 2^i.

Если $i = 0$, то нужно проверить, есть ли ребро из u в v. Иначе:

$$R(u, v, i) = \bigcup_w (R(u, v, i-1) \cap R(w, v, i-1)).$$

Т. е. нужно перебрать все значения w и для каждого значения запустить алгоритмы ($R(u, v, i-1)$ и $R(w, v, i-1)$).

Главное преимущество памяти заключается в том, что ее можно использовать вторично. Если на памяти закончилось вычисление, то можно на этой же памяти провести новое вычисление.

Глубина рекурсии будет равна $O(s(n))$. Вычисление на каждом уровне рекурсии будут тоже размера $O(s(n))$. Кроме того, на каждом шаге нужно хранить номер w. Значит, всего потребуется памяти $O(s(n)^2)$.