Московский физико-технический институт
Факультет инноваций и высоких технологий
Математическая логика и теория алгоритмов, осень 2015
Лекция 2: Сложностные классы **P**, **NP** и **coNP**

Краткое содержание

Что называть быстрым алгоритмом? Полиномиальные алгоритмы. Определение класса **P**. Примеры задач из **P**. Замкнутость **P** относительно операций дополнения, объединения, пересечения. Другие классы для детерминированных вычислений: **E**, **EXP** и т.д. Идея недетерминированных вычислений. Недетерминированные машины Тьюринга. Класс $\mathbf{NTIME}(T(n))$. Вложение $\mathbf{NTIME}(T(n)) \subset \mathbf{DTIME}(2^{O(T(n))})$. Определения класса **NP** через недетерминированные машины и через сертификаты. Эквивалентность двух определений. Примеры задач из **NP**. Цепочка вложений $\mathbf{P} \subset \mathbf{NP} \subset \mathbf{EXP}$. Замкнутость **NP** относительно объединения и пересечения. Проблема перебора ($\mathbf{P} \stackrel{?}{=} \mathbf{NP}$): формулировка и известные обстоятельства, препятствующие её решению. Классы **coNP** и $\mathbf{NP} \cap \mathbf{coNP}$. Если $\mathbf{P} = \mathbf{NP}$, то $\mathbf{NP} = \mathbf{coNP}$. Классы **NE** и **NEXP**. Если $\mathbf{P} = \mathbf{NP}$, то $\mathbf{EXP} = \mathbf{NEXP}$.

1 Что называть быстрым алгоритмом?

На прошлой лекции мы условились измерять время работы программы как число её шагов в худшем случае. Но какое время считать хорошим, доступным для реализации на настоящих машинах? Точный ответ зависит от приложений. Например, реализуемая программа, анализирующая граф всего интернета, может иметь практически только линейное время работы. (Возможно, подойдёт $O(n \log n)$, но никак не $O(n^2)$: такая программа будет попросту не успевать за изменением графа). Однако, для общей теории хотелось бы иметь более прочный фундамент. Желательно, чтобы класс быстрых алгоритмов не зависел от деталей вычислительной модели. В качестве такого класса был выбран класс алгоритмов с полиномиальным временем работы. Хотя для больших степеней полинома такие алгоритмы не реализуемы на практике, большие степени полинома в реальных алгоритмах обычно и не встречаются. Более того, если открыт алгоритм с большим полиномиальным временем работы, то обычно в последующих работах степень уменьшается до приемлемых значений. (Например, знаменитый AKS-алгоритм проверки простоты изначально имел сложность $O(n^{12})$, а затем был улучшен до $O(n^6)$). Также не встречаются и алгоритмы со временем работы вида $n^{\log n}$ или даже $n^{\log \log n}$, что асимптотически быстрее полинома, но при реальных значениях n вполне доступно. Полиномиальность времени работы хороша тем, что не зависит от деталей вычислительной модели. Например, при переходе от многоленточной машины к одноленточной время возводится в квадрат, но квадрат полинома остаётся полиномом.

2 Класс P

Формально класс **P** определяется так:

Определение 1. $\mathbf{P} = \cup_{c=1}^{\infty} \mathbf{DTIME}(n^c)$.

Можно упрощённо записать это определение как $\mathbf{P} = \mathbf{DTIME}(\mathrm{poly}(n))$. Примерами языков из $\mathbf{P}$ являются такие множества:

- $\mathsf{PATH} = \{(G, s, t) \mid$ в графе G есть путь из s в $t\}$. Наличие пути в графе проверяется быстро, например обходом в ширину.
- $\mathsf{CONNECTED} = \{G \mid G$ — связный граф$\}$. Например, можно проверить граф на связность, проверив наличие пути из некоторой фиксированной вершины во все остальные, или использовать обход графа единовременно.
- $\mathsf{EULERCYCLE} = \{G \mid$ в графе G есть эйлеров цикл$\}$. По критерию существования эйлерова цикла достаточно проверить связность, а также посчитать степени всех вершин и проверить, что все они чётные.
- $\mathsf{BIPARTITE} = \{G \mid$ граф G двудольный$\}$. Иными словами, вершины графа G можно покрасить в 2 цвета, так чтобы вершины одного цвета не были соединены ребром. В данном случае подойдёт «жадный» алгоритм: покрасим одну вершину произвольно, дальше будем красить соседние с уже покрашенными согласно условию, пока не придём к противоречию или не покрасим всю компоненту связности. В последнем случае продолжим тем же алгоритмом, пока все компоненты не кончатся. Если в какой-то момент случилось противоречие, значит граф содержит нечётный цикл, а тогда он не является двудольным.

Обратите внимание, что если в постановку задачи входит числовой параметр n, то полином считается не от этого параметра, а от его логарифма, т.к. на запись числа n требуется $\log n$ битов. Если алгоритм работает полиномиальное время от самого n, то такой алгоритм называется *псевдополиномиальным*. Существуют задачи, для которых не известно полиномиального алгоритма, но известен псевдополиномиальный. Например, такими задачами являются задача о рюкзаке

$$\mathsf{KNAPSACK} = \{(n_1, \ldots, n_k, m_1, \ldots, m_k, N, M) \mid \exists \alpha \in \{0,1\}^k \sum \alpha_i n_i \leqslant N \text{ и } \sum \alpha_i m_i \geqslant M\}$$

и её частный случай

$$\mathsf{SUBSET\text{-}SUM} = \{(n_1, \ldots, n_k, N) \mid \exists \alpha \in \{0,1\}^k \sum \alpha_i n_i = N\}.$$

Интерпретация задачи о рюкзаке такая: имеются предметы с весами n_i и стоимостями m_i. Вопрос: можно ли взять в рюкзак несколько предметов, так чтобы их суммарный вес не превысил N (иначе рюкзак порвётся), а суммарная стоимость была не меньше M. В задаче $\mathsf{SUBSET\text{-}SUM}$ остаётся один параметр: ставится вопрос о том, можно ли заполнить рюкзак полностью.

Нетрудно заметить следующие соотношения для класса $\mathbf{P}$:

Теорема 2. *Если $L \in \mathbf{P}$ и $M \in \mathbf{P}$, то $\overline{L}$, $L \cup M$, $L \cap M$ также лежат в $\mathbf{P}$.*

Иначе говоря, класс $\mathbf{P}$ замкнут относительно операций дополнения, объединения, пересечения.

Доказательство. Действительно, имея ответ на вопрос, принадлежит ли слово x языкам L и M, простейшими логическими операциями (отрицание, дизъюнкция, конъюнкция) можно получить ответы на вопросы о принадлежности слова x языкам $\overline{L}$, $L \cup M$, $L \cap M$ соответственно. Эти операции займут совсем немного времени, поэтому время работы останется полиномиальным. Более точно, если $L \in \mathbf{DTIME}(T(n))$, то $\overline{L} \in \mathbf{DTIME}(T(n))$, а если $L \in \mathbf{DTIME}(T(n))$ и $M \in \mathbf{DTIME}(S(n))$, то $L \cup M$ и $L \cap M$ лежат в $\mathbf{DTIME}(\max\{T(n), S(n)\})$. □

Помимо класса **P** изучают другие классы, определяемые порядком роста времени работы на машине Тьюринга:

- $\mathbf{E} = \mathbf{DTIME}(2^{O(n)}) = \cup_{c=1}^{\infty} \mathbf{DTIME}(2^{cn})$;
- $\mathbf{EXP} = \mathbf{DTIME}(2^{\mathrm{poly}(n)}) = \cup_{c=1}^{\infty} \mathbf{DTIME}(2^{n^c})$ (иногда его также называют **EXPTIME**);
- $\mathbf{EE} = \mathbf{DTIME}(2^{2^{O(n)}}) = \cup_{c=1}^{\infty} \mathbf{DTIME}(2^{2^{cn}})$;
- $\mathbf{EEXP} = \mathbf{DTIME}(2^{2^{\mathrm{poly}(n)}}) = \cup_{c=1}^{\infty} \mathbf{DTIME}(2^{2^{n^c}})$;
- и т.д.

Очевидно, что $\mathbf{P} \subset \mathbf{E} \subset \mathbf{EXP} \subset \mathbf{EE} \subset \mathbf{EEXP}$. Из теоремы об иерархии следует, что все эти вложения строгие. Доказательство этой теоремы мы пока отложим.

3 Недетерминированные машины Тьюринга

Недетерминированные вычисления — важная теоретическая концепция, хотя по всей видимости не реализуемая на практике. Неформально говоря, недетерминизм позволяет проводить перебор экспоненциальных семейств слов за один шаг, сразу «угадывая» нужное слово. Формально недетерминированные вычисления проводятся на недетерминированной машине Тьюринга.

Определение 3. *Недетерминированной машиной Тьюринга* с k лентами называется кортеж $\langle \Sigma, \Gamma, Q, q_0, q_a, q_r, \delta \rangle$, где Σ, Γ и Q суть конечные непустые множества, причём $\Sigma \subset \Gamma$ и $\Gamma \cap Q = \varnothing$, $q_0, q_a, q_r \in Q$ и попарно различны, а $\delta\colon (Q \setminus \{q_a, q_r\}) \times \Gamma^k \rightrightarrows Q \times \Gamma^k \times \{L, N, R\}^k$ — всюду определённая многозначная функция.

Таким образом, единственным отличием от детерминированной машины является многозначность функции перехода δ. Гораздо сильнее отличие в определении того, что такое вычисление и что означает распознавание языка.

Определение 4. Пусть $C = (a_1\sigma_1, \ldots, a_k\sigma_k; \tau_1 b_1, \ldots, \tau_k b_k; q)$ и $C' = (a'_1, \ldots, a'_k; b'_1, \ldots, b'_k; r)$ — некоторые конфигурации, где $q \neq q_a$ и $q \neq q_r$. Тогда переход от C к C' *допусти́м*, если среди значений $\delta(q, \tau_1, \ldots, \tau_k)$ найдётся $(r, \rho_1, \ldots, \rho_k, D_1, \ldots, D_k)$, такое что:

- Если $D_i = L$, то $a'_i = a_i$ (или $a'_i = \#$, если $a_i = \varepsilon$), $b'_i = \sigma_i \rho_i b_i$;
- Если $D_i = N$, то $a'_i = a_i \sigma_i$, $b'_i = \rho_i b_i$;

- Если $D_i = R$, то $a'_i = a_i\sigma_i\rho_i$, $b'_i = b_i$ (или $b'_i = \#$, если $b_i = \varepsilon$).

Таким образом, из каждой конфигурации можно перейти не в одну, а в несколько, в зависимости от количества значений функции перехода.

Определение 5. Цепочка конфигураций $(C_1, \ldots, C_n)$ называется *вычислением*, если все переходы от C_i к C_{i+1} допустимы.

Удобно считать, что все конфигурации с одним и тем же началом образуют укоренённое дерево: корнем будет начальная конфигурация, а непосредственными потомками каждой вершины — те конфигурации, переходы в которые допустимы. Как обычно, будем считать, что машина, получившая на вход слово x, начинает вычисление в конфигурации $(\underbrace{\#, \ldots, \#}_{k \text{ раз}}; x, \underbrace{\#, \ldots, \#}_{k-1 \text{ раз}}; q_0)$. Ниже мы будем просто писать, что вычисление начинается с x.

Определение 6. Говорят, что недетерминированная машина M *распознаёт* язык L за время $T(n)$, если при всех $x \in \{0,1\}^*$ верно следующее:

- Любое вычисление, начинающееся с x, не длиннее $T(|x|)$ (иначе говоря, машина в любом случае останавливается не более чем за $T(|x|)$ шагов);
- Если $x \in L$, то найдётся вычисление, начинающееся с x и заканчивающееся в состоянии q_a;
- Если $x \notin L$, то любое вычисление, начинающееся с x, заканчивается в состоянии q_r.

Недетерминированные вычисления позволяют определить соответствующий сложностной класс:

Определение 7. Классом $\mathbf{NTIME}(T(n))$ называется множество языков, распознаваемых на недетерминированной машине Тьюринга за время $O(T(n))$.

Замечание 8. Некоторые авторы говорят не о многозначной, а о двузначной функции перехода δ, или о двух функциях δ_0 и δ_1. От такой вариации время работы изменится в константу раз, поскольку множество значений функции δ имеет константный (т.е. зависящий не от длины входа, а только от параметров машины) размер, поэтому выбор нужного значения можно смоделировать как константное число последовательных бинарных выборов. Точный размер множества значений — $|Q| \cdot |\Gamma|^k \cdot 3^k$.

Легко установить следующую связь между детерминированными и недетерминированными вычислениями:

Теорема 9. $\mathbf{NTIME}(T(n)) \subset \mathbf{DTIME}(2^{O(T(n))})$.

Как и раньше, запись в правой части толкуется как $\cup_{c=1}^{\infty} \mathbf{DTIME}(2^{cT(n)})$. В дальнейшем мы также будем пользоваться подобными сокращениями.

Доказательство. Как уже упоминалось, у функции перехода может быть не больше некоторой константы K значений. Значит, общее количество конфигураций во всех вычислениях, начинающихся с данного входа и имеющих длину не больше T, не превышает $K^T = 2^{O(T)}$. За такое же время можно все эти конфигурации перебрать (например, обходя дерево) и проверить, встречается ли среди них состояние q_a. Если встречается, то данный вход лежит в языке, иначе не лежит. □

4 Два определения класса NP и их эквивалентность

Класс **NP** важен как для теории, так и для практики. У него есть два эквивалентных определения: через недетерминированные машины и через сертификаты.

Определение 10. $\mathbf{NP} = \cup_{c=1}^{\infty} \mathbf{NTIME}(n^c)$.

Это определение объясняет название **NP**: N — недетерминированные вычисления, P — полиномиальное время.

Определение 11. Классом **NP** называется множество языков L, для которых существует функция $V(x, s)$ с булевыми значениями, вычислимая за полиномиальное время от длины первого аргумента, такая что:

- Если $x \in L$, то $\exists s\ V(x, s) = 1$;
- Если $x \notin L$, то $\forall s\ V(x, s) = 0$.

Второй вход s часто называют *сертификатом*, а функцию V — *верификатором*. Таким образом, сертификат удостоверяет, что $x \in L$, а верификатор проверяет верность сертификата. Для слов из языка подходящий сертификат должен существовать, а для слов не из языка все сертификаты должны отвергаться. Иначе говоря, класс **NP** — это класс языков, принадлежность к которым можно быстро доказать, а **P** — класс языков, принадлежность к которым можно быстро выяснить.

Замечание 12. Оговорка о том, что время работы V полиномиально именно от длины x, важна, иначе в сертификат можно было бы включить слишком много информации. Альтернативно можно потребовать, чтобы алгоритм V был полиномиальным от своего входа целиком, а вот длина сертификата была бы ограничена полиномом от длины x.

Теорема 13. *Определения 10 и 11 эквивалентны, т.е. задают один и тот же класс языков.*

Доказательство. Пусть L распознаётся недетерминированной машиной M, которая работает $O(n^c)$ шагов. Тогда в качестве сертификата можно взять последовательность значений функции перехода, а верификатор будет моделировать работу машины M, используя данные сертификата для выбора одной из ветвей алгоритма. Можно действовать и по-другому: в качестве сертификата взять всё вычисление, а верификатором проверять его соответствие программе.

Пусть, напротив, для L верно сертификатное определение. Тогда сертификат не может быть длиннее, чем время работы V, т.е. чем некоторый полином $p(|x|)$. Недетерминированная машина будет работать следующим образом: сначала она недетерминированно напишет сертификат s длины не больше $p(|x|)$, а затем запустит $V(x, s)$. □

Примерами языков из **NP** служат такие задачи:

- $\mathsf{SAT} = \{\varphi \mid \varphi$ — выполнимая булева формула$\}$. (Выполнимость означает, что формула равна 1 на некотором входе). В данном случае сертификатом будет выполняющий набор, а верификатор проверит, что значение формулы на этом наборе действительно равно 1. Важным частным случаем является задача 3SAT, в которой про формулу φ дополнительно сказано, что она представлена в виде 3-КНФ, т.е. КНФ, в которой каждый дизъюнкт является дизъюнктом 3 литералов.
- $\mathsf{CLIQUE} = \{(G, k) \mid$ в графе G найдётся полный подграф хотя бы на k вершинах$\}$. Сертификатом будет сама клика.
- $\mathsf{3COL} = \{G \mid$ вершины графа G можно правильно раскрасить в 3 цвета$\}$. Сертификатом будет раскраска.
- $\mathsf{HAMCYCLE} = \{G \mid$ в графе G существует гамильтонов цикл$\}$. (Гамильтонов цикл проходит ровно 1 раз через каждую вершину). Сертификатом будет сам цикл.
- KNAPSACK, SUBSET-SUM (см. выше). Сертификатом будет набор α.
- $\mathsf{FACTORING} = \{(N, a, b) \mid$ у числа N существует простой делитель на отрезке $[a, b]\}$. Сертификатом будет этот делитель.

Задачи из класса **NP** встречаются очень часто в дискретной математике, алгебре, логике, дискретной оптимизации и других разделах «конечной математики».

Как и класс **P**, класс **NP** замкнут относительно объединения и пересечения языков (но не дополнения!). Более того, можно доказать такую теорему:

Теорема 14. *Если* $L \in \mathbf{NTIME}(T(n))$, *а* $M \in \mathbf{NTIME}(S(n))$, *то* $L \cup M$ *и* $L \cap M$ *лежат в* $\mathbf{NTIME}(\max\{T(n), S(n)\})$.

Доказательство. Недетерминированная машина должна сначала провести вычисление для L, затем для M, а затем взять дизъюнкцию или конъюнкцию результатов для $L \cup M$ и $L \cap M$ соответственно. В случае с дизъюнкцией для элементов $L \cup M$ хотя бы на одной ветви хотя бы одного из этапов ответ будет положительным, поэтому на соответствующей ветви общего алгоритма ответ будет положительный, а для слов не из $L \cup M$ на всех ветвях обоих этапов будет отрицательный ответ, поэтому и итоговый ответ будет отрицательным. В случае с конъюнкцией для элементов $L \cap M$ на какой-то ветви на каждом этапе ответ будет положительным, поэтому на составной ветви ответ также будет положительным. Для слов не из $L \cap M$ на каком-то из этапов ответы на всех ветвях будут отрицательными, поэтому и итоговый ответ будет отрицательным.

Время работы будет примерно равно сумме $O(T(n))$ и $O(S(n))$, что также представимо в виде $O(\max\{T(n), S(n)\})$. □

5 Проблема перебора ($\mathbf{P} \stackrel{?}{=} \mathbf{NP}$)

Легко понять, что $\mathbf{P} \subset \mathbf{NP}$ (и вообще, $\mathbf{DTIME}(T(n)) \subset \mathbf{NTIME}(T(n))$): детерминированные машины являются частным случаем недетерминированных. Также из

теоремы 9 следует, что $\mathbf{NP} \subset \mathbf{EXP}$. Вскоре мы докажем, что хотя бы одно из этих вложений строгое, т.к. $\mathbf{P} \neq \mathbf{EXP}$. Однако вопрос о том, являются ли строгими оба вложения, открыт. Наибольшее внимание привлекает вопрос о равенстве классов $\mathbf{P}$ и $\mathbf{NP}$. Он также называется проблемой перебора, потому что может быть сформулирован так: возможно ли смоделировать экспоненциальный перебор возможных сертификатов за полиномиальное время? Существенная часть теории сложности вычислений и подавляющая часть теоретической криптографии разработаны в предположении $\mathbf{P} \neq \mathbf{NP}$. Однако это не единственное недоказанное предположение. Существуют и другие: неколлапсирование полиномиальной иерархии, существование односторонней функции и др. Исходный вопрос поставлен в начале 1970-х Стивеном Куком и Леонидом Левиным, однако до сих пор все существенные продвижения лишь дополнительно показывали, что вопрос сложный:

- В оригинальных работах Кук и Левин независимо друг от друга открыли класс $\mathbf{NP}$-полных задач, «самых сложных» в классе $\mathbf{NP}$. Нахождение полиномиального алгоритма для любой из них привело бы к тому, что $\mathbf{P} = \mathbf{NP}$, однако многолетние усилия огромного числа учёных по поиску такого алгоритма успехом не увенчались. Подробно про $\mathbf{NP}$-полные задачи мы поговорим на следующей лекции.

- В 1975 году Бейкер, Джилл и Соловей доказали, что проблема $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ *не релятивизуется*, т.е. существует оракул A, такой что $\mathbf{P}^A = \mathbf{NP}^A$, и существует оракул B, такой что $\mathbf{P}^B \neq \mathbf{NP}^B$. (Подробнее о вычислениях с оракулом и этой теореме мы поговорим через лекцию). Это значит, что любое правильное доказательство должно быть нерелятивизуемым, а примеров таких доказательств не так много.

- В 1994 году Разборов и Рудич ввели понятие естественного доказательства и доказали следующий факт: если верно некоторое усиление $\mathbf{P} \neq \mathbf{NP}$, то само утверждение $\mathbf{P} \neq \mathbf{NP}$ невозможно доказать при помощи естественных доказательств.

- В 2008 году Аронсон и Вигдерсон доказали, что гипотетическое доказательство $\mathbf{P} \neq \mathbf{NP}$ не может использовать «алгебраические техники», что дополнительно исключило многие известные методы.

При этом из-за того, что за решение проблемы перебора назначена премия в миллион долларов, вопрос привлекает внимание многих непрофессионалов, в результате несколько раз в год появляются заявления о том, что проблема решена. Как правило, найти ошибку в рассуждениях не составляет большого труда, но случаются и более изощрённые попытки, например работа Виджая Деолаликара 2010 года, вызвавшая переполох в среде теоретиков. Сообщество быстро самоорганизовалось, пристально изучило стостраничную статью и нашло «дырку», которая пока не заделана.

Также многие обыватели, завороженные словосочетанием «квантовый компьютер», ошибочно полагают, что изобретение квантового компьютера позволит решать все задачи из $\mathbf{NP}$ за полиномиальное время. Однако на современном уровне знаний это не так: про задачу о разложении числа на множители (которую быстро решает квантовый компьютер) не доказана $\mathbf{NP}$-полнота, а для $\mathbf{NP}$-полных задач не известно квантовых алгоритмов.

Рис. 1: Кирпичная кладка на стене факультета компьютерных наук принстонского университета. Каждый отсутствующий кирпич обозначает единицу, присутствующий — ноль. Строки кодируют в семибитной ASCII-записи вопрос «P=NP?».

Вопрос о равенстве **P** и **NP** настолько важен, что в 1989 году его увековечили в камне на стене факультета компьютерных наук принстонского университета (см. рис. 1).

6 Класс coNP

В отличие от класса **P**, для класса **NP** скорее всего не верна замкнутость относительно дополнения. Действительно, непонятно, как можно быстро доказать отсутствие подходящего сертификата. Дополнения языков из **NP** объединяют в отдельный класс **coNP**:

Определение 15. $\mathbf{coNP} = \{L \mid \overline{L} \in \mathbf{NP}\}$.

Иначе говоря, если к **NP** относятся языки, принадлежность к которым легко доказать, то **coNP** — класс языков, принадлежность к которым легко опровергнуть. Самым естественным представителем класса **coNP** является язык тавтологий $\mathsf{TAUT} = \{\varphi \mid \varphi$ — тавтология$\}$. Действительно, опровергнуть, что φ тавтология, очень легко: нужно предъявить набор значений, на котором формула ложна. Доказать, что φ тавтология, можно, предъявив вывод в исчислении высказываний, однако такой вывод может быть

слишком длинным: длиннее, чем полином от длины φ. Коротких доказательств тавтологичности не известно.

Легко показать, что $\mathbf{P} \subset \mathbf{coNP} \subset \mathbf{EXP}$. Таким образом, $\mathbf{P} \subset \mathbf{NP} \cap \mathbf{coNP}$. Про это вложение тоже неизвестно, строго ли оно, однако большинство исследователей верят, что строго. В частности, задача FACTORING лежит в $\mathbf{NP} \cap \mathbf{coNP}$: доказать наличие делителя в нужном интервале можно, просто предъявив его, а опровергнуть — приведя полное разложение на простые множители. Имея полиномиальный алгоритм проверки простоты, это разложение можно проверить и таким образом убедиться в отсутствии множителей в заданном интервале. Однако полиномиального алгоритма для поиска разложения на множители, по всей видимости, не существует.

Также легко показать такую связь:

Теорема 16. *Если* $\mathbf{P} = \mathbf{NP}$, *то* $\mathbf{NP} = \mathbf{coNP} = \mathbf{P}$.

Доказательство. Действительно, пусть $L \in \mathbf{coNP}$. Значит, $\overline{L} \in \mathbf{NP}$. Поскольку мы предположили, что $\mathbf{P} = \mathbf{NP}$, то $\overline{L} \in \mathbf{P}$. Поскольку класс $\mathbf{P}$ замкнут относительно дополнения, то и $L \in \mathbf{P}$. Значит, $\mathbf{coNP} \subset \mathbf{P}$. Поскольку $\mathbf{P} \subset \mathbf{coNP}$, имеем $\mathbf{P} = \mathbf{coNP}$, а значит и $\mathbf{NP} = \mathbf{coNP}$ в предположении $\mathbf{P} = \mathbf{NP}$. □

В частности, если $\mathbf{P} = \mathbf{NP}$, то $\mathbf{NP}$ всё-таки замкнут относительно дополнения. Об истинности обратной теоремы ничего не известно: из $\mathbf{NP} = \mathbf{coNP}$ автоматически не следует $\mathbf{P} = \mathbf{NP}$, различие $\mathbf{NP}$ и $\mathbf{coNP}$ это отдельное недоказанное предположение.

7 Класс NEXP

Для недетерминированных вычислений также можно определять высшие классы, например $\mathbf{NE} = \mathbf{NTIME}(2^{O(n)})$ и $\mathbf{NEXP} = \mathbf{NTIME}(2^{\mathrm{poly}(n)})$. Вопрос $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ можно перемасштабировать в вопрос $\mathbf{EXP} \stackrel{?}{=} \mathbf{NEXP}$. Между ответами можно установить следующую связь:

Теорема 17. *Если* $\mathbf{P} = \mathbf{NP}$, *то* $\mathbf{EXP} = \mathbf{NEXP}$.

Доказательство. Поскольку $\mathbf{EXP} \subset \mathbf{NEXP}$, достаточно доказать $\mathbf{NEXP} \subset \mathbf{EXP}$. В предположении $\mathbf{NP} \subset \mathbf{P}$ это доказывается при помощи метода «раздутия» (padding argument). Интуитивно, экспоненциальное время работы можно превратить в полиномиальное, если добавить в аргумент экспоненциальное количество ничего не значащего «мусора».

Формально, пусть $L \in \mathbf{NEXP}$, причём L распознаётся некоторой недетерминированной машиной M за время 2^{n^c}. Тогда язык $\tilde{L} = \{x01^{2^{n^c}} \mid n = |x|,\ x \in L\}$ лежит в $\mathbf{NP}$. Действительно, можно сначала легко проверить, что аргумент имеет вид $x01^{2^{n^c}}$, а затем применить к x машину M. Время работы машины M составит 2^{n^c}, что будет полиномом (и даже линейной функцией) от длины аргумента $\left|x01^{2^{n^c}}\right| = n + 1 + 2^{n^c}$. В предположении $\mathbf{NP} \subset \mathbf{P}$ найдётся детерминированная машина M', распознающая $\tilde{L}$ за время $\mathrm{poly}(n + 2^{n^c}) = 2^{O(n^c)}$. Тогда детерминированная машина, приписывающая к

слову x единицы в количестве 2^{n^c} и запускающая M' на полученном слове, будет распознавать L за время $2^{O(n^c)}$, т.е. за экспоненциальное от длины входа время. Значит, $L \in \mathbf{EXP}$, что и требовалось. $\square$

Про истинность обратного утверждения (если $\mathbf{EXP} = \mathbf{NEXP}$, то $\mathbf{P} = \mathbf{NP}$) ничего не известно. В частности, существует оракул A, для которого $\mathbf{EXP}^A = \mathbf{NEXP}^A$, но $\mathbf{P}^A \neq \mathbf{NP}^A$.