Классические и квантовые вычисления

А. Шень (вр.и.о. А. Китаева); написал М. Вялый

2 лекция (19/02/98)

Краткое содержание предыдущей лекции

На первой лекции были даны определения машины Тьюринга (МТ), вычислимой функции, разрешимого предиката[1)], полиномиально вычислимой функции и предиката, распознаваемого за полиномиальное время.

Были введены классы $\mathcal{P}$ — функций, вычислимых за полиномиальное время, и $\mathcal{PSPACE}$ — функций, вычислимых с использованием полиномиально ограниченной памяти.

Была также определена ещё одна модель вычислений — схемы. *Схема* — это способ вычислить булеву (т. е. принимающую значения 0 или 1) функцию от конечного числа булевых аргументов. Помимо значений аргументов $x_1, \ldots, x_n$ схема использует некоторое количество вспомогательных переменных $y_1, \ldots, y_s$ и некоторый набор (*базис*) булевых функций. Схема полностью определяется последовательностью *присваиваний* $Y_1, \ldots, Y_s$. Каждое присваивание Y_i имеет вид $y_i := f_j(u_{k_1}, \ldots, u_{k_r})$, где $f_j(\cdot)$ — одна из базисных функций, а переменная u_{k_m} либо равна некоторому аргументу исходной функции $u_{k_m} = x_l$, либо одной из вспомогательных переменных с меньшим номером $u_{k_m} = y_l$, $l < i$. Если присвоить значения всем аргументам, то, выполняя последовательно все указанные присваивания, получим значения для всех вспомогательных переменных. Результатом вычисления считается значение последней переменной y_s. По определению *схема вычисляет функцию* f, если для любых значений аргументов выполняется равенство $y_s = f(x_1, \ldots, x_n)$.

$$
\begin{aligned}
&\cdots \quad \cdots\\
y_i &:= x_5 \vee x_7\\
y_{i+1} &:= y_{i-1} \wedge x_3\\
y_{i+2} &:= \neg y_2\\
&\cdots \quad \cdots
\end{aligned}
$$

На врезке слева приведен фрагмент схемы в базисе $\{\neg, \vee, \wedge\}$. Было доказано, что в этом базисе существует схема вычисления для любой булевой функции.

Число s (количество присваиваний в схеме) называется *размером* схемы. Размер схемы имеет тот же смысл, что и количество шагов вычислений на машине Тьюринга. Минимальный размер схемы, вычисляющей функцию f, называется (схемной) *сложностью* функции f и обозначается $c(f)$.

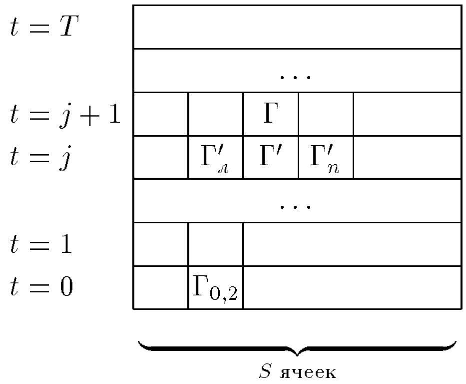
Каждый предикат f определяет последовательность булевых функций f_n, $f_n : \mathbb{B}^n \to \mathbb{B}$ следующим образом: $f_n(x_1, x_2, \ldots, x_n) = f(\overline{x_1 x_2 \ldots x_n})$. Предикат принадлежит классу ${}_{non-uniform}\mathcal{P}$ если, и только если, существует такой полином $p(x)$, что $c(f_n) \le p(n)$.

Переход от одного (полного) базиса к другому (полному) базису меняет схемную сложность функций, но это изменение ограничено некоторым фиксированным множителем (зависящим от рассматриваемой пары базисов). Поэтому класс ${}_{non-uniform}\mathcal{P}$ не зависит от выбора базиса.

Теорема. $\mathcal{P} \subset {}_{non-uniform}\mathcal{P}$.

1) Предикат рассматривается как функция, принимающая два значения 0 (ложь) и 1 (истина).

Доказательство. Если МТ работает полиномиально ограниченное время, то она в состоянии использовать лишь полиномиально ограниченное количество ячеек на предоставленной ей бесконечной ленте.



Поэтому весь процесс вычисления на входном слове x длины n можно представить таблицей $T \times S$, где $T, S \leq poly(n)$ (см. врезку слева). Символы $\Gamma_{j,k}$, записанные в таблице, задают состояние k-той ячейки в j-й момент времени (символ, записанный в ячейке, находится ли над этой ячейкой считывающая головка, и если да, то в каком состоянии пребывает в данный момент МТ). Для определённости считаем, что если вычисление заканчивается при некотором входе быстрее, чем за T шагов, то вышележащие строки строки таблицы повторяют последнее состояние ленты (и машины).

Чтобы объяснить, как построить схему, вычисляющую значения предиката на словах длины n, сделаем несколько замечаний. *Состояние каждой ячейки таблицы* можно закодировать конечным (не зависящим от n) числом булевых переменных. *Есть локальные правила согласования*, т. е. состояние каждой клетки Γ в строке выше первой однозначно определяется состояниями клеток в предыдущей строке, лежащих непосредственно под данной (Γ'), левее данной ($\Gamma'_л$) и правее данной ($\Gamma'_п$). Каждая переменная, кодирующая состояние клетки Γ, есть функция от переменных, кодирующих состояния клеток Γ'_*. Все эти функции могут быть вычислены схемами конечного размера. Объединяя эти схемы, получим схему, вычисляющую все переменные, кодирующие состояния клеток таблицы; размер этой схемы будет $O(ST) = O(n^{O(1)})$.

Осталось заметить, что переменные, кодирующие часть клеток 0-й строки определяются входным словом, а переменные, кодирующие остальные клетки являются константами. Чтобы узнать результат вычисления, нужно определить, в каком из двух финальных состояний остановилась машина: принимающем или отвергающем. Без ограничения общности можно считать, что состояния кодируются так, что одна из кодирующих переменных равна 1 только в том случае, когда состояние машины — принимающее. В этом случае достаточно взять дизъюнкцию таких переменных по всем ячейкам строки $t = T$, она и будет результатом вычисления. Это увеличит размер схемы не более, чем на S. $\square$

Замечание 1. Класс $_{non-uniform}\mathcal{P}$ шире класса $\mathcal{P}$. Любой булевой функции от натурального аргумента $\varphi(n)$ можно сопоставить предикат f_φ по правилу $f_\varphi(x) = \varphi(|x|)$, где $|x|$ обозначает длину слова x. Ограничения такого предиката на слова длины n есть тождественный 0 или тождественная 1, сложность таких функций ограничена константой. Поэтому все такие предикаты по определению принадлежат $_{non-uniform}\mathcal{P}$, хотя среди них есть и неразрешимые предикаты.

Справедливо следующее усиление теоремы.

Теорема. *f принадлежит $\mathcal{P}$ тогда, и только тогда, когда выполнены два условия:*
1) $f \in {}_{non-uniform}\mathcal{P}$;
2) существует МТ, которая по числу n за время $poly(n)$ строит схему вычисления f_n.

Доказательство. $\Longrightarrow$ То описание, которое дано в доказательстве предыдущей теоремы, нетрудно превратить в МТ, работающую полиномиальное по n время (почти все действия состояли в многократном, но полиномиально ограниченном, применении локальных правил согласования).

$\Longleftarrow$ Так же просто. Искомая машина должна вычислить размер слова, использовать указанную в 2) машину для построения по этому размеру схемы вычисления $f_{|x|}$, а затем подать x и полученную схему $Y_{|x|}$ на вход машины, интерпретирующей схемы. $\square$

1. Класс $\mathcal{NP}$

Общеизвестно, что помимо уже рассмотренного класса $\mathcal{P}$ есть ещё класс $\mathcal{NP}$ — недетерминированных полиномиальных вычислений. Термин «недетерминированный» неудачный, но он уже стал стандартным.

Класс $\mathcal{NP}$ определён только для предикатов. Говорят, например, что «свойство графа “иметь гамильтонов цикл” принадлежит $\mathcal{NP}$».

Мы дадим несколько определений этого класса. Первое использует понятие *недетерминированной машины Тьюринга*. Устройство недетерминированной машины (НМТ) такое же, как и обычной (детерминированной), за одним исключением. У НМТ есть особые недетерминированные состояния. Из такого состояния q НМТ может перейти в одно из двух других состояний. Иначе можно сказать так: в таблице переходов НМТ для каждого состояния машины (внутреннее состояние и текущий символ на ленте) вместо единственно возможного варианта действий (изменение внутреннего состояния, запись символа на ленту, сдвиг читающей головки) имеется набор возможных действий.

Путь вычисления НМТ определяется выбором одного из возможных переходов на каждом шаге. Неопределённость приводит к тому, что путей вычисления для НМТ, работающей на данном входном слове, оказывается много.

Определение 1. Предикат L принадлежит классу $\mathcal{NP}$, если существуют НМТ M и полином $p(n)$ такие, что

$L(x) = 1$	$\Longrightarrow$	существует путь вычисления, дающий ответ “да” и требующий времени $\le p(\|x\|)$;
$L(x) = 0$	$\Longrightarrow$	(*1 вариант*) нет указанного выше пути;
		(*2 вариант*) на любом пути вычисления ответа “да” не получается.

Замечание 2. Варианты определения (1) и (2) эквивалентны. Чтобы исключить возможность ответов “да” на длинных путях вычисления, достаточно взять НМТ, имитирующую исходную НМТ и подсчитывающую количество сделанных исходной НМТ шагов. Когда число шагов превышает $p(|x|)$, машина останавливается.

Замечание 3. В предыдущем рассуждении допущена одна тонкая ошибка. В определении ничего не сказано о многочлене $p(n)$, кроме факта его существования. Если коэффициенты многочлена невычислимы, могут возникнуть проблемы с указанным выше способом сведения одного определения к другому (нужно вычислить значение многочлена). Чтобы в дальнейшем избегать этой сложности, будем считать, что выражение «существует многочлен» понимается конструктивно: предъявлены коэффициенты многочлена в десятичной записи.

Второе определение класса $\mathcal{NP}$ представляется более естественным. Оно использует понятие полиномиально вычислимого предиката от двух переменных. Предикат $R(x,y)$ принадлежит классу $\mathcal{P}$ если он вычислим за полиномиальное от размера входа время. Под размером входа можно понимать $|x|+|y|$ или $\max(|x|,|y|)$ — получатся эквивалентные определения.

Определение 2. *Класс $\mathcal{NP}$ образуют предикаты, представимые в форме*

$$Q(x) = \exists y\big((|y| < q(|x|)) \wedge (R(x,y))\big),$$

где $q(\cdot)$ — многочлен, а $R(\cdot,\cdot) \in \mathcal{P}$.

Пример. Пусть $R(x,y) =$ « y есть гамильтонов цикл в графе x» (более точное выражение: «y, x — двоичные коды гамильтонова цикла и графа, такие что ... »). Возьмём $q(n) = n$. Тогда $Q(x)$ будет в точности означать, что в графе найдётся гамильтонов цикл.

Теорема. *Данные выше определения 1 и 2 эквивалентны.*

Доказательство. (1) $\Longrightarrow$ (2) Пусть есть НМТ M и полином $p(\cdot)$ из первого определения. Рассмотрим предикат $R(x, y) =$ «y есть протокол работы M на возможном пути вычисления со входом x, длящегося не более $p(|x|)$, заканчивающегося ответом "да"». Длина такого протокола при разумном кодировании линейно зависит от длины вычисления (а если использовать в качестве протокола описанную выше таблицу вычисления, то квадратично), поэтому в качестве $q(n)$ для второго определения можно взять $p(n)$ $(p^2(n))$, умноженный на подходящую константу.

Чтобы закончить доказательство в этом случае, осталось проверить, что $R(x, y) \in \mathcal{P}$. Это почти очевидно. Мы должны проверить протокол работы некоторой НМТ за полиномиально ограниченное время. Это займёт нас примерно на то же время (совершенно незачем смотреть на одну ячейку экспоненциально долго).

(2) $\Longrightarrow$ (1) Пусть есть R, q из определения 2. Построим M для определения 1. Что она должна делать?

1) M недетерминированно пишет y (который существует для любого слова, для которого ответом должно быть "да"). Говорят ещё, что M «отгадывает» ("guessing") слово y. Более точно это означает, что на первом этапе своей работы M находится в таком (недетерминированном) состоянии, в котором она пишет один из символов на ленту, сдвигается вправо и выбирает между сохранением этого состояния или переходом в (уже детерминированное) состояние начала следующего этапа.

2) После завершения первого этапа на ленте помимо входного слова x записано ещё и слово y. Теперь M осталось распознать значение предиката $R(x, y)$ за полиномиальное время. Но это уже можно сделать совершенно бесхитростно — детерминированным алгоритмом, существующим в силу определения 2. $\square$

Можно дать ещё одно определение класса $\mathcal{NP}$. Хотя оно есть не более чем вариант определения 2, именно в такой форме можно обобщать определение $\mathcal{NP}$ и получать новые, скорее всего, более широкие классы.

Определение 3. Имеется два персонажа: король **A**rthur, умственные способности которого полиномиально ограничены, и волшебник **M**erlin, который интеллектуально всемогущ и знает правильные ответы на все вопросы. Король интересуется некоторым свойством $L(x)$ (например, «есть ли у графа гамильтонов цикл»), а волшебник **M**, состоящий у него на службе, берётся ему помочь. Однако **A** не доверяет своему волшебнику и хочет иметь возможность самостоятельно убедиться в верности предложенного **M** ответа.

Поэтому они действуют следующим образом. **A** и **M** оба смотрят на слово x, после чего **M** сообщает некоторую информацию (слово y). После этого **A** находит значение некоторого полиномиально вычислимого предиката $R(x, y)$, которое и считает правильным ответом.

В этой, быть может излишне беллетризованной, ситуации определение класса $\mathcal{NP}$ звучит так: $L(x)$ принадлежит $\mathcal{NP}$, если

$L(x) = 1 \implies$ у **M** есть способ убедить **A** в этом;

$L(x) = 0 \implies$ как бы ни изощрялся **M**, король **A** не поверит, что $L(x) = 1$.

Эквивалентность этого определения определению 2 не вызывает сомнений. Действительно, из-за полиномиальной ограниченности короля **A** сообщение волшебника **M** должно иметь полиномиальный размер. В остальном различия между определениями чисто внешние.

2. Сводимость и $\mathcal{NP}$-полнота

Хотя до сих пор (по крайней мере в этом конспекте) мы рассматривали предикаты как булевы функции, столь же естественно рассматривать их как множества слов ($x \in L \Leftrightarrow L(x) = 1$).

Трудность вычисления предикатов можно сравнивать пользуясь следующим определением.

Определение. Сводимость по Карпу. Предикат L_1 сводится к предикату L_2 (обозначение $L_1 \preceq L_2$), если существует такая $f \in \mathcal{P}$, что $\forall x\big(x \in L_1 \Leftrightarrow f(x) \in L_2\big)$.

Сводимость по Карпу также называют полиномиальной сводимостью, а часто — просто сводимостью.

Лемма. а) $\dfrac{L_2 \in \mathcal{P} \quad L_1 \preceq L_2}{L_1 \in \mathcal{P}}$ б) $\dfrac{L_1 \notin \mathcal{P} \quad L_1 \preceq L_2}{L_2 \notin \mathcal{P}}$ в) $\dfrac{L_2 \in \mathcal{NP} \quad L_1 \preceq L_2}{L_1 \in \mathcal{NP}}$

Доказательство. Пункт а) совершенно очевиден — чтобы вычислить значение предиката L_1 достаточно применить ко входу x функцию f, а к результату её работы — программу вычисления L_2.

Пункт б) столь же очевиден.

Пункт в) также прост. Его можно объяснять по-разному. Например, так: Мерлин сообщает Артуру $f(x)$ и слово y, по которому Артур может убедиться, что $f(x) \in L_2$. Артур также может проверить, что $f(x)$ найдено правильно. Используя определение 2, можно написать цепочку эквивалентностей

$$L_1(x) \Leftrightarrow L_2(f(x)) \Leftrightarrow \exists y\big((|y| < q(|f(x)|)) \wedge (R(f(x), y)\big) \Leftrightarrow$$
$$\Leftrightarrow \exists y\big(|y| < r(x)\big) \wedge \big[|y| < q(|f(x)|) \wedge R(f(x), y)\big].$$

□

Если некоторый предикат таков, что к нему сводятся все предикаты из $\mathcal{NP}$, он называется $\mathcal{NP}$-*полным.* Вычисление полного предиката с точностью до полиномиальной потери эффективности (которая в этой науке считается несущественной) позволяет вычислять все предикаты из $\mathcal{NP}$. Такие полные предикаты существуют.

Задача выполнимости. $SAT(x) \Leftrightarrow$ x есть формула с булевыми переменными и символами $(\neg, \vee, \wedge)$, которая ***иногда*** истинна.

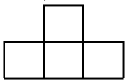
Теорема. *1)* $SAT \in \mathcal{NP}$; *2) любое* $\mathcal{NP}$-*свойство сводится к* SAT.

Следствие. *Если* $SAT \in \mathcal{P}$, *то* $\mathcal{P} = \mathcal{NP}$.

Доказательство. 1) Мерлину достаточно сообщить Артуру значения переменных, входящих в формулу, при которых она истинна. Артур справится с проверкой истинности полученного высказывания.

2) Пусть $\mathcal{NP}$-предикат, который нужно свести к SAT имеет вид $L(x) = \exists y\big((|y| < q(|x|)) \wedge R(x, y)\big)$. Сведение состоит в том, строится такая формула φ_x, которая истинна $\Leftrightarrow L(x)$.

Для доказательства воспользуемся таблицей вычисления, введённой в доказательстве первой теоремы. Напишем таблицу вычисления для МТ, вычисляющей $R(x, y)$ (входом является пара $x\#y$).

Переменные, которые мы будем рассматривать, имеют тот же смысл, что и в доказательстве первой теоремы (это коды состояний клеток таблицы вычисления). Чтобы таблица вычисления соответствовала правильно проведенному успешному (с ответом "да") вычислению, должны выполняться локальные условия корректности для каждой четверки клеток вида , и конечный ответ должен быть "да", что записывается как дизъюнкция по некоторому множеству переменных, отвечающих самой верхней строке таблицы. Формула φ_x будет конъюнкцией всех этих условий, в которую подставлены известные значения переменных, кодирующих вход $x\#y$. $\square$

Примеры других $\mathcal{NP}$-полных задач можно получать с помощью следующей леммы.

Лемма. *Если $SAT \preceq L$ и $L \in \mathcal{NP}$, то L — $\mathcal{NP}$-полная.*

И вообще, если L_1 — $\mathcal{NP}$-полная, $L_1 \preceq L_2$ и $L_2 \in \mathcal{NP}$, то L_2 — $\mathcal{NP}$-полная.

Для доказательства этой леммы достаточно проверить транзитивность сводимости: если $L_1 \preceq L_2$, $L_2 \preceq L_3$, то $L_1 \preceq L_3$. Эта транзитивность следует из того, что композиция двух полиномиально вычислимых функций полиномиально вычислима.

3-выполнимость. 3-$SAT(x)$ $\Leftrightarrow$ x есть конъюнкция дизъюнкций с тремя булевыми переменными или отрицаниями переменных (3-КНФ), которая ***иногда*** истинна.

3-SAT также $\mathcal{NP}$-полна. Это устанавливается сведением её к SAT.

Теорема. $SAT \preceq 3 - SAT$.

Доказательство. Построим по любой схеме, необязательно формуле, 3-КНФ, которая выполнима тогда, и только тогда, когда функция, вычисляемая схемой, не равна тождественно 0. Для этого построим вначале конъюнкцию K'' условий, означающих, что каждая из вспомогательных переменных y_j имеет значение, соответствующее правой части присваивания Y_j. Для примера напишем ту часть этой конъюнкции, которая соответствует фрагменту, приведённому на первой странице

$$K'' = \ldots \wedge \big(y_i \Leftrightarrow (x_5 \vee x_7)\big) \wedge \big(y_{i+1} \Leftrightarrow (y_{i-1} \wedge x_3)\big) \wedge \big(y_{i+2} \Leftrightarrow \neg y_2\big) \wedge \ldots$$

Каждый член этой конъюнкции есть формула от не более чем трех булевых переменных, поэтому его можно записать как 3-КНФ (проверьте!). В результате подстановки этих 3-КНФ в K'' получим 3-КНФ K'. Искомая 3-КНФ имеет вид $K = K' \wedge y_s$. Действительно, если при каких-то значениях переменных x_i схема выдает 1, это означает, что в результате присваиваний схемы получается $y_s = 1$, а тогда и $K(x_1, \ldots, x_n) = 1$. $\square$