1. Универсальная машина Тьюринга

Классическая теорема теории алгоритмов утверждает, что существует универсальная вычислимая функция для класса частичных вычислимых функций типа $\{0,1\}^* \to \{0,1\}^*$. Функция U, отображающая пары двоичных слов в двоичные слова называется универсальной, если выполнено следующее. Для любой вычислимой частичной функции f, аргументами и значениями которой являются двоичные слова, существует $p \in \{0,1\}^*$, для которого

$$f(x) = U(p,x)$$

для всех $x \in \{0,1\}^*$. Двухленточную машину, вычисляющую универсальную функцию (вход p дается на первой ленте, вход x — на второй) мы будем называть *универсальной* машиной Тьюринга. Нас интересует такой вопрос: сколько времени требуется универсальной машине для вычисления $U(p,x)$ по сравнению со временем вычисления $f(x)$? Оказывается, существуют универсальные машины, для которых замедление линейно: для каждого p время работы универсальной машины на паре p,x не более чем в константу раз больше времени вычисления $f(x)$. При этом константа линейно зависит от длины p.

Будем в дальнейшем рассматривать только машины Тьюринга, алфавит которых содержит пробел и символы 0,1 (и, возможно, иные символы). Только такие машины могут получать на вход двоичные слова.

Теорема 1. *Существует двухленточная машина Тьюринга U такая, что для любой одноленточной машины Тьюринга M найдется $p \in \{0,1\}^*$, для которого*

$$M(x) = U(p,x)$$

для всех $x \in \{0,1\}^*$. При этом

$$t_U(p,x) = O(|p| t_M(x) + |p||x|)$$

(константы в "О-большом" абсолютные). (Время работы U на паре (p,x) по существу ограничено линейной функцией от произведения $|p|$ и времени работы M на входе x.)

Доказательство. Идея построения машины U в следующем. Универсальная машина будет моделировать работу M на входе x шаг за шагом.

Слово p будет записью программы машины M в удобном для моделирования формате. Формат записи программ должен позволять моделировать один шаг работы M за время $O(|p|)$. Слагаемое $O(|p||x|)$ необходимо, чтобы преобразовать исходное данное x в удобный для моделирования формат.

Пусть M — некоторая одноленточная машина Тьюринга. Закодируем ее программу нулями и единицами следующим образом. Пусть $\{a_1, \ldots, a_l\}$ алфавит машины M, причем символы перенумерованы так, что a_1 — это пробел, $a_2 = 0$ и $a_3 = 1$. Пусть $\{q_1, \ldots, q_k\}$ — множество состояний, причем q_1 — начальное состояние. Каждая команда машины $q_i a_j \to q_s a_t \Delta$ является словом в алфавите, состоящем из символов $a_1, \ldots, a_m, q_1, \ldots, q_k, \to, R, L, S$. Мы хотим представлять команды словами в фиксированном алфавите. Для этого договоримся записывать q_i и a_j последовательностями q(двоичная запись i) и a(двоичная запись j), соответственно. Тогда каждая команда станет словом длины $O(\log k + \log l)$ в восьмибуквенном алфавите $\{0, 1, a, q, \to, R, L, S\}$. Закодируем буквы этого алфавита трехбитовыми строками, например, в лексикографическом порядке. Двоичные слова, в которые при выбранном кодировании перейдут q_i, a_j, и символы $\to, R, L, S$ будем называть их кодами и обозначать $\overline{q_i}$, $\overline{a_j}$ и т.д.

Заменив в записи команды каждый символ на его код, мы получим двоичное слово, называемое кодом команды. Перепишем коды всех команд подряд в лексикографическом порядке и получим двоичное слово p, кодирующее программу машины M в удобном формате. Длина слова p имеет порядок $kl(\log k + \log l)$, поскольку количество команд равно kl и на запись одной команды нужно $O(\log k + \log l)$ битов.

Первая лента машины U будет содержать последовательность кодов всех букв, написанных на ленте моделируемой одноленточной машины M (в том же порядке), причем головка будет обозревать первый символ кода буквы, обозреваемой машиной M. На второй ленте будет записан код программы p, а затем код состояния q_i машины M. Как смоделировать один шаг машины M? Нужно скопировать на вторую ленту код обозреваемой буквы a_j. Затем просмотрев программу p, нужно найти вхождение последовательности $\overline{q_i a_j \to}$. Для этого понадобится $O(kl(\log k + \log l)) = O(|p|)$ шагов (мы просматриваем все kl вхождений кода символа $\to$, и для каждого вхождения сравниваем слово $\overline{q_i a_j}$ со словом, написанным слева от этого вхождения). Когда мы найдем вхождение последовательности $\overline{q_i a_j \to}$, надо заменить на второй ленте $\overline{q_i}$ на

последовательность $\overline{q_s a_t \Delta}$, стоящую справа от символа $\rightarrow$. Затем заменяем код a_j на код a_s на первой ленте. Чтобы для этого не понадобилось сдвигать все содержимое ленты, нужно, чтобы коды всех символов имели одну длину (равную $1 + \lceil \log l \rceil$). Затем перемещаем головку на первой ленте влево или вправо, в зависимости от Δ и стираем код $\overline{a_t \Delta}$ со второй ленты. Моделирование одного шага закончено. Оно выполняется за $O(|p|)$ шагов.

Осталось только заметить, что преобразование исходного данного x в его код $\overline{x}$ можно выполнить за время $O(|x| \log l) = O(|p||x|)$ с помощью второй ленты: каждый бит слова x надо заменить на код нуля или код единицы. Обратное преобразование кода результата работы y в двоичное слово также требует времени $O(|y| \log l)$. Поскольку длина y ограничена максимумом из длины x и времени работы M на x, слагаемое $O(\log l|y|)$ поглощается слагаемыми $O(|p||x|)$ и $O(|p|t_M(x))$. $\square$

2. Теоремы о иерархии

Пусть имеется некоторая функция $t : \mathbb{N} \rightarrow \mathbb{N}$. Существует ли язык $A \subset \{0,1\}^*$, который не распознается никакой одноленточной машиной Тьюринга за время $t(n)$? (Напомним, что машина M распознает язык A за время $t(n)$, если на любом входе $x \in \{0,1\}^*$ она останавливается, проработав не более $t(|x|)$ шагов и напечатав 0 или 1 в зависимости от того, принадлежит ли x к A или нет.) Разумеется, такой язык существует для любой функции $t(n)$: можно взять в качестве A любой неразрешимый язык, то есть, язык для которого вообще нет распознающей машины Тьюринга.

Для понимания дальнейшего полезно вспомнить, как доказывается существование неразрешимого языка. Вот наиболее простое доказательство этого факта. Перенумеруем все одноленточные машины Тьюринга, алфавит которых содержит символы $0, 1$ и пробел:

$$M_0, M_1, \ldots, M_n, \ldots.$$

Каждая машина должна встречаться в этой последовательности хотя бы однажды, но может встречаться и много раз. Рассмотрим язык

$$A = \{1^n \mid M_n(1^n) = 0\}. \tag{1}$$

Через 1^n здесь обозначено слово, состоящее из n единиц, фактически A это множество слов в алфавите $\{1\}$. Докажем, что какую бы нумерацию машин Тьюринга мы ни выбрали, построенный язык A неразрешим. Действительно, пусть машина с номером k разрешает A, то есть, $M_k(x) \in \{0, 1\}$ для любого x и при этом

$$M_k(x) = 1 \Leftrightarrow x \in A.$$

По построению A из этого следует, что для любого n

$$M_k(1^n) = 1 \Leftrightarrow M_n(1^n) = 0.$$

При n равном k мы получаем противоречие, которое доказывает, что язык A неразрешим.

Этот метод доказательства принято называть диагональным по следующей причине. Расположим результаты работы всевозможных одноленточных машин Тьюринга на всевозможных входах вида 1^n в бесконечную таблицу, строки которой нумеруются номерами машин, а столбцы их входами вида 1^n. На диагонали полученной матрицы стоит последовательность $M_n(1^n)$.

Более содержательный вопрос — для каких функций $t(n)$ существует *разрешимый* язык, который невозможно распознать за время $t(n)$? Оказывается, такой язык существует для всех вычислимых функций $t(n)$ и его можно построить с помощью того же метода. А именно, рассмотрим язык

$$\begin{gathered}\{1^n \mid \text{машина } M_n \text{ останавливается на входе } 1^n \\ \text{с результатом 0 за } t(n) \text{ шагов}\}.\end{gathered} \tag{2}$$

Теорема 2. *(1) Какова бы ни была выбранная нумерация одноленточных машин Тьюринга, не существует одноленточной машины Тьюринга, разрешающей язык* (2) *за время* $t(n)$. *(2) Если выбранная нумерация машин Тьюринга вычислима, то язык* (2) *разрешим. (Вычислимость нумерации означает существование алгоритма, который по любому* n *находит программу машины* M_n.*)*

Доказательство. (1) Допустим, машина Тьюринга с номером k разрешает язык (2) за время $t(n)$. Тогда

$$M_k(1^n) = 1 \Leftrightarrow (M_n(1^n) = 0 \text{ за время } t(n)).$$

При n равном k это означает, что

$$M_k(1^k) = 1 \Leftrightarrow (M_k(1^k) = 0 \text{ за время } t(k)).$$

Но поскольку машина M_k на входе 1^k заведомо остановится за время $t(k)$, мы получаем противоречие:

$$M_k(1^k) = 1 \Leftrightarrow M_k(1^k) = 0.$$

(2) Алгоритм разрешения (2) заключается в моделировании: по данному натуральному n мы находим программу машины M_n (по условию это возможно — у нас есть алгоритм нахождения программы M_n по n) и затем применяем ее шаг за шагом к исходному данному 1^n. Мы обрываем моделирование после шага номер $t(n)$ и выдаем 0, если машина M_n не остановилась за это время. Иначе мы выдаем 1, если машина M_n остановилась с результатом 0 и выдаем 0 во всех остальных случаях.

В этом рассуждении важно, что функция $t(n)$ вычислима, иначе мы не знали бы когда нам можно оборвать моделирование. Кстати, по этой же причине не проходит попытка разрешить язык A с помощью моделирования — мы можем начать пошаговое моделирование работы M_n на входе 1^n, но мы не знаем, когда его закончить. □

1 Пусть фиксирована вычислимая нумерация одноленточных машин Тьюринга. Приведите пример невычислимой функции $t(n)$, для которой язык (2) неразрешим.

Теперь у нас все готово для доказательства теорем о иерархии. В наиболее простой форме теорема о иерархии говорит, что для данных двух функций $t(n) \ll t'(n)$ существует язык, распознаваемый за время $t'(n)$, но не распознаваемый за время $t(n)$ (на одноленточных машинах Тьюринга). В качестве такого языка можно взять диагональный язык из теоремы 2. Вспомним, как доказывалась разрешимость языка (2). Мы сначала вычисляли значение $t(n)$, а затем моделировали $t(n)$ шагов работы машины M_n на входе 1^n. Чтобы доказать теорему о ирерархии нам нужно подсчитать, сколько шагов на это требуется. Самый простой способ подсчета такой: сначала оценить, сколько шагов требуется многоленточной машине Тьюринга, а потом использовать моделирование многоленточной машины с помощью одноленточной с квадратичным замедлением. На многоленточной машине можно организовать вычисление так. (1) Сначала скопировать исходное данное 1^n на вторую ленту (мы предполагаем, что изначально оно записано на первой ленте) и

вычислить значение $t(n)$ в унарной записи, то есть $1^{t(n)}$, результат записать на вторую ленту. (2) Затем скопировать 1^n на третью ленту и преобразовать его в программу машины M_n. (3) Затем смоделировать $t(n)$ шагов работы M на входе $1^{t(n)}$. После моделирования каждого шага уменьшать на 1 счетчик количества шагов, записанный не второй ленте.

Общее время работы этой многоленточной машины будет равно (время вычисления $1^{t(n)}$)+ (время преобразования 1^n в программу M_n) + $t(n) \times$ (1+ (время моделирования одного шага)). Первое слагаемое зависит от трудности вычисления $t(n)$, а остальные от выбора способа записи программ машин и выбора нумерации машин Тьюринга. Обычно теорема о ирерархии применяется только к просто вычислимым функциям $t(n)$ (полиномам, экспонентам и т.д.). Мы будем предполагать, что существует многоленточная машина Тьюринга, вычисляющая $1^{t(n)}$ по 1^n за $O(t(n)+n)$ шагов. Такие функции называются *конструируемыми по времени.*

Чтобы последнее слагаемое было мало нужно, чтобы номер машины был существенно больше длины записи ее программы. Для этого годится самая, например, следующая нумерация. Сопоставим каждой одноленточной машине M двоичное слово p (двоичный код ее программы), как в теореме 1. Номером машины M будет натуральное число, двоичная запись которого равна $1p$ (мы приписываем слева 1, поскольку двоичные записи натуральных чисел начинаются на 1). Если двоичная запись натурального числа n не имеет такого вида, то будем считать n номером тривиальной машины, которая ничего не делает (стоит на одном месте). По двоичному слову p на двухленточной машине за время $O(|p|) = O(\log n)$ можно узнать, является ли оно кодом какой-нибудь программы машины Тьюринга. Для этого, как это делалось при моделировании (см. доказательство теоремы 1), надо проверить, что оно состоит из кодов команд, записанных в лексикографическом порядке. Если p не является кодом программы никакой машины, то мы не выполняем моделирования, а останавливаемся сразу с результатом 0.

Итак, первое слагаемое по условию есть $O(t(n)+n)$ шагов. Второе слагаемое (время преобразования n из унарной в двоичную запись и последующая проверка, является ли слово p программой некоторой машины) есть $O(n)$ (преобразование) плюс $O(|p|) = O(\log n)$ (проверка). Наконец, последнее слагаемое, как доказано в теореме 1, есть $O(t(n)|p|) = O(t(n)\log n)$.

Кроме того, нужно еще учесть следующее обстоятельство. Исходное

данное 1^n записано на ленте не в том формате, который требуется для начала пошагового моделирования. Необходимо заменить каждую из n единиц на ее код. Для этого достаточно времени $O(n \log n)$.

Теорема 3. *Существует нумерация одноленточных машин Тьюринга такая, что для любой конструируемой функции $t(n) \geqslant n$ язык (2) можно распознать за время $O((t(n) \log n)^2)$ на одноленточной машине Тьюринга.*

2 Докажите, что функции n и 2^n являются конструируемыми. Докажите, что для произвольного натурального c функция $t(n) \equiv c$ конструируема.

3 Докажите, что сумма и произведение конструируемых функций конструируемы. Докажите, что если $t(n)$ конструируемая функция, то и функция $2^{t(n)}$ конструируема. Следовательно, все полиномы с натуральными коэффициентами, а также экспоненты от полиномов конструируемы.

4 Докажите, что в формулировке теоремы требование конструируемости функции t можно ослабить. Достаточно потребовать, чтобы функция $1^n \to 1^{t(n)}$ вычислялась на многоленточной машине Тьюринга за время $O(t(n) \log n)$.

Замечание. Для построения языка, распознаваемого за время $O((t(n) \log n)^2)$, но не распознаваемого за время $t(n)$ (для конструируемой функции $t(n)$) можно и не прибегать к нумерации машин Тьюринга. Рассмотрим следующий язык

$$\{1^{2^{|p|}}0p \mid p \text{ есть двоичный код программы машины Тьюринга,}$$
$$\text{останавливающейся на входе } 1^{2^{|p|}}0p$$
$$\text{за } t(2^{|p|} + 1 + |p|) \text{ шагов с результатом } 0\}.$$

Этот язык отличается от языка (2) тем, что программа указывается явно, а не своим номером, но при этом исходное данное искусственно удлиняется, чтобы один шаг машины M можно было смоделировать за время $O(\log n)$. Прежде чем начинать моделирование надо проверить, что вход имеет вид $1^{2^{|x|}}0x$ для некоторого слова x, то есть, понадобится опять переводить число $2^{|x|}$ из унарной записи в двоичную. Так что доказательство того, что этот язык можно распознать за время $O((t(n) \log n)^2)$ не

будет проще. Теорема 2 для этого языка доказывается точно так же, как и раньше.

Теорему 3 можно усилить, избавившись от возведения $t(n)$ в квадрат. Точнее, для той же самой нумерации машин можно доказать, что язык (2) распознается за время $O(t(n)\log t(n)+n^2)$ на одноленточной машине Тьюринга (для просто вычислимых функций $t(n)$). Для этого нужно записывать счетчик числа шагов в двоичной форме, а не в унарной. При этом вычитание единицы из счетчика уже будет выполняться не за 1 шаг, как раньше, а за $O(\log t(n))$ шагов. В результате многоленточная машина будет распознавать (2) за время $O(t(n)\log t(n))$. Это больше, чем было раньше, но зато теперь содержимое всех лент, кроме первой (на которой записано содержимое ленты моделируемой машины) занимает мало места. А именно, на всех лентах, кроме первой используется не более $O(\log t(n))$ ячеек. Это дает возможность сэкономить на переходе от многоленточной машины к одноленточной. А именно, при моделировании многоленточной машины содержимое всех лент кроме первой, надо сдвигать вслед за положением головки на первой ленте. Тогда моделирование одного шага многоленточной машины и уменьшение счечтика обойдется $O(\log t(n))$ шагов. Кроме этого надо скопировать 1^n на "вторую дорожку" и разложить n в двоичную запись. На это уйдет $O(n^2)$ шагов. Затем надо проверить, является ли полученное слово p программой машины Тьюринга (в $O(|p|)=O(\log n)$ шагов). Наконец, надо в исходном данном заменить каждую единицу на ее код (в $O(n^2)$ шагов).

Итак, мы установили следующий факт. Назовем функцию $t(n)$ доступной, если функция $1^n \mapsto$ (двоичная запись $t(n)$) вычислима за время $O(t(n)\log t(n)+n^2)$ на одноленточной машине Тьюринга,

Теорема 4. *Для некоторой нумерации одноленточных машин Тьюринга для любой доступной функции $t(n) \geqslant n$ язык (2) можно распознать за время $O(t(n)\log t(n)+n^2)$ на одноленточной машине Тьюринга.*

5 Докажите, что класс доступных функций удовлетворяет утверждениям задач 2 и 3.

Обычно требуется доказать, что существует язык не распознаваемый за время $t(n)$ ни для одной функции $t(n)$ из некоторого семейства функций и распознаваемый за время $O(t'(n))$ для некоторой функции $t'(n)$, значительно большей всех функций из этого семейства. Например, если мы хотим доказать, что данный язык не принадлежит классу P, то

в качестве семейства можно взять класс всех полиномов с натуральными коэффициентами (можно ограничиться классом всех полиномов вида $n^k + k$, где k — натуральное число, поскольку любой полином ограничен сверху некоторым полиномом такого вида). Другой пример — если мы хотим доказать, что данный язык нельзя разрешить за время $O(2^n)$, в качестве класса семейства можно взять класс всех функций вида $k2^n$, $k = 0, 1, 2, \ldots$. Утверждения такого сорта можно доказывать, применяя теорему 4 к подходящей функции $t(n)$.

6 Докажите, что для любой доступной функции $t(n) > n$ существует язык, распознаваемый за время $O(t(n) \log t(n) + n^2)$, но не распознаваемый за время $o(t(n))$ (на одноленточных машинах Тьюринга). (Указание. Этим требованиям удовлетворяет язык из теоремы 4. Любую машину, распознающую его за время $o(t(n))$ можно преобразовать в машину, распознающую его за время $t(n)$. Заметьте, что исходная машина на коротких входах может работать более $t(n)$ шагов.)

7 Докажите, что существет язык, распознаваемый за время $O(2^n)$ и не принадлежащий классу P. (Указание. Примените задачу 6 к функции $2^{n/2}$.)

8 Докажите, что для любой доступной функции $t(n) > n$ существует язык, распознаваемый за время $O(t(n) \log^2 t(n) + n^2)$, но не распознаваемый за время $O(t(n))$ (на одноленточных машинах Тьюринга). (Указание. Примените задачу 6 к функции $t(n) \log t(n)$.)

9 Докажите, что для каждого натурального $c \geqslant 2$ существует язык, распознаваемый за время $O(n^c \log^2 n)$ на одноленточной машине Тьюринга, но не распознаваемый за время $O(n^c)$ на одноленточных машинах Тьюринга. (Указание. Примените задачу 8 к функци n^c.)

3. Нижние оценки времени распознавания других языков

Трудность распознавания других языков обычно доказывают с помощью сведéний к языкам (1) и (2). Вот как это делается.

Докажем, например, неразрешимость классической проблемы применимости машины Тьюринга, то есть неразрешимость языка

$$H = \{1^n 0x \mid x \in \{0,1\}^*,\ M_n \text{ останавливается на входе } x\}.$$

(Мы предполагаем, что нумерация машин вычислима.) Для доказательства неразрешимости языка H мы сведем к проблеме его разрешения проблему разрешения языка (1). Для этого рассмотрим следующее преобразование на машинах Тьюринга: преобразуем машину M в машину M', которая работает так же, как и M, за одним исключением: машина M' останавливается только если M останавливается с результатом 0. То есть, машина M' останавливается на входе x тогда и только тогда, когда $M(x) = 0$. Существует вычислимое отображение $k \mapsto k'$ такое, что $(M_k)' = M_{k'}$. Действительно, по натуральному числу k мы можем найти программу машины M_k, преобразовать ее в программу машины $(M_k)'$ и затем найти некоторый номер получившейся программы в данной нумерации машин.

Слово 1^n принадлежит языку (1) тогда и только тогда, когда машина $(M_n)'$ останавливается на входе 1^n, то есть слово $1^{n'}01^{n'}$ принадлежит языку H. Значит, если бы существовал алгоритм, распознающий принадлежность данного слова к H, то применив его к слову $1^{n'}01^{n'}$, мы получили бы ответ на вопрос, принадлежит ли слово 1^n к языку (1).

Применение метода сведéния к языку (2) немного более хлопотно, поскольку требуется оценить время, нужное для вычисления сводящего отображения, а также оценить, во сколько раз это отображение увеличивает длину слова. Для примера рассмотрим язык, универсальный для класса всех языков, распознаваемых за время $t(n)$:

$$U = \{1^k 0x \mid \text{машина } M_k \text{ останавливается на входе } x \text{ с результатом 1 за } t(|x|) \text{ шагов}\} \tag{3}$$

К языку U сводится любой язык L, распознаваемый за время $t(n)$: для подходящего k (зависящего от L):

$$x \in L \Leftrightarrow 1^k 0x \in U.$$

Язык U удовлетворяет следующему немного ослабленному варианту теоремы 4:

Теорема 5. *(1) Для некоторого квадратичного полинома $p(n)$ для любой функции t и любой нумерации машин язык неразрешим за время $t(\lfloor (n-1)/2 \rfloor) - p(n)$ на одноленточных машинах Тьюринга. (2) Для некоторой нумерации одноленточных машин Тьюринга для любой функции $t(n) \geqslant n$ такой, что функция $1^n \mapsto$ (двоичная запись $t(n)$) вычислима*

за время $O(t(n)\log t(n)+n^2)$ на одноленточной машине Тьюринга, язык (3) *можно распознать за время $O(t(n)\log t(n)+n^2)$ на одноленточной машине Тьюринга.*

Доказательство. Первое утверждение доказывается сведением задачи разрешения (2) к задаче разрешения U. А именно слово 1^k принадлежит (2) тогда и только тогда, когда слово 1^k01^k принадлежит U. Поэтому если машина Тьюринга M распознает язык U за время $t'(n)$, то применив эту машину к слову 1^k01^k мы за время $t'(2k+1)$ узнаем, принадлежит ли слово 1^k языку (2). Нужно еще учесть время, необходимое для преобразования слова 1^k в слово 1^k01^k. Это можно сделать за время $O(k^2)$. Поэтому общее время работы машины Тьюринга для распознавания (2) будет равно $t'(2k+1)+O(k^2)$. Если $t'(n)=t(\lfloor(n-1)/2\rfloor)-p(n)$, где $p(n)$ подходящий квадратичный полином, то $t'(2k+1)+O(k^2)$ будет меньше $t(k)$ и мы получим противоречие.

Второе утверждение доказывается точно так же, как и теорема 4. □

Если функция $t(\lfloor(n-1)/2\rfloor)$ существенно меньше, чем $t(n)$, то зазор между верхней и нижней оценками для языка (3) хуже, чем для языка (2). Например, если $t(n)=2^{2n+1}$, то язык (3) нельзя распознать за время 2^n, и можно распознать за время $O(n2^{2n})$, в то время как язык (2) нельзя распознать за время 2^{2n+1} (и можно распознать за время $O(n2^{2n})$).

С другой стороны, если функция $t(n)$ ограничена полиномом, то зазоры между верхней и нижней оценками для языков (3) и (2) одинаковы.

4. Теорема Фишера–Рабина

Классическая теорема Тарского утверждает, что элементарная теория упорядоченного поля действительных чисел разрешима. Это означает, что имеется алгоритм, который по замкнутной формуле сигнатуры

$$\{0,1,+,\times,=,<\}$$

выясняет истинна ли она в упорядоченном поле действительных чисел. Теорема Фишера-Рабина утверждает, что такой алгоритм не может быть быстрым: разрешить эту теорию невозможно за время $2^{o(n)}$ (где n длина записи исходной формулы).

Под длиной формулы мы понимаем количество символов в ней. При этом каждая переменная считается одним символом. Чтобы алфавит формул оставался конечным (ведь каждая машина Тьюринга может получать на вход только слова над конечным алфавитом), мы зафиксируем некоторый конечный список индивидных переменных V и будем предполагать, что в формулах не используются другие переменные. Существует такой конечный список, что ни одна машина Тьюринга не разрешает элементарную теорию упорядоченного поля действительных чисел за время $2^{o(n)}$. Более того, это верно для элементарной теории действительных чисел в обедненной сигнатуре $\{0, 1, +, =\}$.

Теорема 6 (Фишера–Рабина). *Не существует одноленточной машины Тьюринга, разпознающей за время $2^{o(n)}$ множество истинных в аддитивной группе действительных чисел замкнутых формул сигнатуры $\{0, 1, +, =\}$, содержащих только переменные из некоторого конечного списка V.*

Замечание. По теореме о моделировани многоленточноых машин одноленточными теорема Фишера–Рабина верна и для многолетночных машин.

Доказательство. Воспользуемся теоремой 3 для $t(n) = 2^n$. Язык L, о котором в ней идет речь, можно распознать за время $O(n^2 2^{2n}) = 2^{O(n)}$ и нельзя (по теореме 2) распознать за время 2^n, а значит и за время $2^{o(n)}$.

Предоположим, теорема Фишера–Рабина неверна, то есть истинность формул можно было распознавать за время $2^{o(n)}$. Мы получим противоречие, доказав, что тогда и язык L можно распознать за время $2^{o(n)}$. Это делается с помощью сведения. Ключевую роль здесь играет следующая лемма.

Основная лемма. Пусть дана зафиксирована произвольная машина Тьюринга M, время работы которой есть $2^{O(n)}$. По любому двоичному слову w за время ограниченное неоторым полиномом от длины w можно найти замкнутую формулу Φ_w длины $O(|w|)$ с $O(1)$ переменными, такую, что

$$M(w) = 1 \Leftrightarrow \Phi_w \text{ истинна.}$$

Допустим лемма уже доказана. Зафиксируем машину Тьюринга M, распознающую L за время $2^{O(n)}$. Рассмотрим следующий алгоритм распознавания L: по данному w находим формулу Φ_w, удовлетворяющую

Основной лемме и применяем алгоритм разпознавания истинности формул за время $2^{o(n)}$. Общее время работы построенной машины Тьюринга равно

$$\mathrm{poly}(|w|) + 2^{o(|\Phi_w|)} = \mathrm{poly}(|w|) + 2^{o(O(|w|))} = \mathrm{poly}(|w|) + 2^{o(|w|)} = 2^{o(|w|)}.$$

По условию, не существует машины Тьюринга, распознающей L за время $2^{o(n)}$, следовательно, не существует и машины Тьюринга, распознающей истинность формул за время $2^{o(n)}$.

Итак, осталось доказать Основную лемму. Она следует из следующей вспомогательной леммы.

Вспомогательная лемма. (1) По любому натуральному n за время $\mathrm{poly}(n)$ можно построить формулы $Z_n(x)$, $N_n(x)$, $P_n(x,y,z)$, $E_n(x,y)$, длина каждой из которых есть $O(n)$, и такие что

$$\begin{aligned} Z_n(x) \text{ истинна} &\Leftrightarrow x = 2^n \\ N_n(x) \text{ истинна} &\Leftrightarrow x \in \mathbb{N},\ x < 2^{2^n} \\ P_n(x,y,z) \text{ истинна} &\Leftrightarrow x \in \mathbb{N},\ x < 2^{2^n},\ xy = z \\ E_n(x,y) \text{ истинна} &\Leftrightarrow x \in \mathbb{N},\ x \leqslant 2^n,\ y = 2^x. \end{aligned}$$

(2) По двоичному слову u можно за время за $\mathrm{poly}(n)$ построить формулу $\Psi_u(x)$ длины $O(|u|)$ такую, что

$$\Psi_u(x) \text{ истинна} \Leftrightarrow (\text{двоичная запись } x) = u.$$

(Имеется в виду, что двоичная запись x дополняется старшими нулями до слова длины $|u|$.) Число переменных во всех указанных формулах ограничено константой.

Допустим, эта лемма уже доказана. Тогда доказательство основной леммы завершается следующим образом. Рассмотрим протокол работы M на входе x. Протокол естественным образом представляется в виде матрицы размера $2^{O(n)} \times 2^{O(n)}$ (где $n = |w|$), элементы которой являются символами из некоторого фиксированного алфавита Γ (зависящего от M). В этой матрице каждый элемент (кроме крайних) в любой строчке, кроме первой, является фиксированной функцией f от трех своих соседей из предыдущей строчки. Элементы первого столбца также являются некоторой функцией g от двух соседей из предыдщуей строки, и то же самое верно для элементов последнего столбца и некоторой функции h.

Запишем строчки этой матрицы подряд, начиная с первой. Получим слово v над алфавитом Г, которое также будем называть протоколом. Оно состоит из 2^{cn} блоков длины 2^{cn}; каждый блок соответствует одной строке матрицы. Элемент матрицы, стоящий в i-ом столбце и j-ой строке, стоит на $(j2^{cn}+i)$-ому символу слова v (нумерация символов начинается с нуля). Заменив каждую букву этого слова его двоичным кодом фиксированной длины l (зависящей только от машины M), мы получим двоичную строку длины $2^{2cn}l$. Эту строку будем называть двоичным кодом протокола.

Будем называть протокол вычисления M на w допускающим, если $M(w)=1$, то есть первая ячейка ленты в момент остановки содержит 1. Формула Φ_w будет утверждать, что существует натуральное число p, двоичная запись которого является кодом допускающего протокола вычисления M на w. Чтобы понять, как это свойство p можно коротко выразить на нашем языке, полезно переформулировать его в терминах самого протокола (а не его кода). Слово v в алфавите Г длины 2^{2cn} является допускающим протоколом вычисления M на w, если и только если
(1) начальный символ $v[0]$ слова v равен паре $\langle w[1]$, начальное состояние $M\rangle$, последующие $n-1$ символов v равны $w[2],\ldots,w[n]$, соответственно, а все оставшиеся символы первого блока v, то есть $v[n],\ldots,v[2^{cn}-1]$, являются пробелами,
(2) для всех $j<2^{cn}-1$ символ номер $(j+1)2^{cn}$ слова v (то есть первый символ $j+1$-ого блока v) равен значению функции g на паре символов $v[j2^{cn}]$ и $v[j2^{cn}+1]$,
(3) для всех $j<2^{cn}-1$ и всех $0<i<2^{cn}-1$ символ номер $(j+1)2^{cn}+i$ слова v (то есть i-ый символ $j+1$-ого блока v) равен значению функции f на тройке символов $v[j2^{cn}+i-1]$, $v[j2^{cn}+i]$ и $v[j2^{cn}+i+1]$;
(4) для всех $j<2^{cn}-1$ символ номер $(j+2)2^{cn}-1$ слова v (то есть последний символ $j+1$-ого блока v) равен значению функции h на паре символов $v[(j+1)2^{cn}-2]$ и $v[(j+1)2^{cn}-1]$,
(5) символ номер $(2^{cn}-1)2^{cn}$ слова v (то есть первый символ последнего блока v) равен 1.

Теперь нам нужно выразить каждое из этих свойств слова v формулой нашей сигнатуры длины $O(n)$, содержащей одну свободную переменную p. С помощью формулы N_{2cln} мы можем выразить свойство быть натуральным числом меньшим $2^{2^{2cln}}$. Больших натуральных чисел нам не понадобится и мы будем считать, что в нижеприведенных формулах все переменные пробегают натуральные числа этой величины.

Сначала выразим свойство (1). Будем считать, что l-битный код пробела состоит из l нулей. С помощью формулы $Z_{cn}(x)$ из леммы, мы можем выразить константу $2^{cn}l$ (длина одного блока в двоичном коде протокола). С помощью формулы $\Psi_u(x)$ из леммы для подходящего u, зависящего от w, можно выразить свойство "двоичная запись x равна $\langle w[1]$, начальное состояние $M\rangle w[2]\dots w[n]$". Формула

$$\exists x, y\ (\Phi_u(x)\ \wedge\ p = x + 2^{cln}y)$$

истинна тогда и только тогда, когда слово p является двоичным кодом слова v, удовлетворяющего, свойству (1).

Перейдем к свойству (2). Для этого нам понадобятся формулы N_n, P_n, E_n. С их помощью можно написать формулу $I_n(i,p)$ длины $O(n)$, которая утверждает, что $i < 2^{2cnl}$ и i-ый бит двоичной записи p равен 1:

$$\exists u, v\ (u < 2^i\ \wedge\ p = u + 2^i + 2^{i+1}v).$$

Здесь отношение $<$ (на натуральных числах) можно выразить с помощью сложения, а произведение и возведение в степень с помощью формул P_{2cln}, E_{2cln}. Получившаяся формула автоматически даст и верхнюю оценку на i. Используя формулу $I_n(i,p)$, можно формулой длины $O(n)$ со свободными переменнами p, i, j, k выразить отношение "двоичная запись p является двоичным кодом слова v в алфавите Γ, для которого $v[i] = g(v[j], v[k])$".

Кроме этого, для выражения свойста (2) нужно уметь выражать умножение на числах, не превосходящих длины двоичного кода протокола. По лемме мы можем это делать формулой длины $O(n)$ даже на числах значительно большей длины. Еще нужно выразить константу 2^{cn} формулой длины $O(n)$ (формула Z_{cn}).

Выразимость оставшихся свойств v формулами длины $O(n)$ доказывается аналогично. Таким образом, нам осталось доказать вспомогательную лемму. Начнем с формулы $Z_n(x)$. Напомним, что ее длина должна быть $O(n)$, она должна иметь ограниченное число переменных и выражать свойство $x = 2^n$. Из-за первого требования очевидная формула

$$x = 1 + 1 + \dots + 1 \quad (2^n \text{ раз})$$

не годится и надо изобретать что-то другое. Попробуем построить формулу Z_n по индукции: в качестве $Z_0(x)$ возьмем формулу $x = 1$ и положим

$$Z_{n+1}(y) = \exists x\ (Z_n(x)\ \wedge\ y = x + x).$$

При увеличении n на 1 длина формулы Z_n увеличивается на 10 символов, следовательно, длина Z_n равна $10n+3$, как нам и хотелось. Построение Z_n закончено? Почти закончено, надо только подсчитать количество различных переменных в Z_n. Если при увеличении n на единицу использовать какждый раз новую переменную, то количество различных переменных в Z_n будет равно $n+1$. Поэтому будем повторно использовать переменные, и нам хватит всего двух переменных. Для четных n мы будем строить формулу Z_n со свободной переменной x, а для нечентых — со свободной переменной y. Для нечетных n индуктивный переход имеет вид:

$$Z_{n+1}(x) = \exists y\ (Z_n(y)\ \wedge\ x = y + y).$$

Переменная x имеет одно свободное вхождение в формулу $Z_{n+1}(x)$, а также несколько связанных вхождений (в составе формулы $Z_n(y)$). Правилами построения формул это не запрещено.

Теперь можно перейти к формуле $\Psi_u(x)$. Ее тоже построим по индукции: если u пустое слово, то положим $\Psi_u(x) = (x = 0)$. Затем определим

$$\Psi_{0u}(y) = \exists x\ (\Psi_u(x)\ \wedge\ y = x+x), \qquad \Psi_{1u}(y) = \exists x\ (\Psi_u(x)\ \wedge\ y = x+x+1).$$

Построение формулы $N_n(x)$ немного сложнее, чем предыдущих двух. Сначала решим более простую задачу построения формулу $T_n(x)$, выражающей свойство "x натуральное число не превосходящее 2^n" (разница с N_n в том, что экспонента одноэтажная , а неравенство нестрогое). Это делается по индукции. В качестве $T_0(x)$ возьмем формулу $x = 0\ \vee\ x = 1$ и положим

$$T_{n+1}(y) = \exists x_1, x_2\ (T_n(x_1)\ \wedge\ T_n(x_2)\ \wedge\ y = x_1 + x_2).$$

Годится ли такая конструкция? Нет, не годится, посколку формула T_{n+1} содержит два вхождения формулы T_n, а следовательно длина T_n растет экспоненциально с ростом n. Применим следующий трюк, позволяющий заменить два вхождения на одно. Положим

$$T_{n+1}(y) = \exists x_1, x_2\ (y = x_1 + x_2\ \wedge\ \forall x\ (x = x_1 \vee\ x = x_2 \to T_n(x)).$$

Смысл формулы не изменился, но теперь она содержит одно вхождение T_n вместо двух. Поэтому длина T_n есть $O(n)$. При этом количество переменных в T_n будет ограничено константой, если при четных n использовать в качестве свободной переменной x, а при нечетных y (каждая из формул T_n будет содержать 4 переменных: x_1, x_2, y, x).

Теперь мы построим формулу $P_n(x, y, z)$ (а затем положим $Z_n(x) = P_n(x, 0, 0)$). Для ее построения по индукции полезно заметить, что x является натуральным числом, строго меньшим $2^{2^{n+1}}$, тогда и только тогда, когда его можно представить в виде $x_1 + x_2 + x_1x_3$, где x_1, x_2, x_3 — натуральные числа, строго меньшие 2^{2^n}. Действительно, пусть x имеет такой вид. Максимально возможное значение x_1, x_2, x_3 равно $a = 2^{2^n} - 1$, поэтому $1 + x \leqslant 1 + a + a + a^2 = (1 + a)^2 = 2^{2^{n+1}}$. Обратно, пусть $x < 2^{2^{n+1}}$. Положим $x_3 + 1 = 2^{2^n}$, а x_1 и x_2 равным, соответственно, частному и остатку от деления x на $x_3 + 1$.

Если x имеет указанный вид, то $xy = x_1y + x_2y + x_3(x_1y)$. В этом равенстве умножение выполняется только на натуральные числа меньшие 2^{2^n}. Поэтому можно определить по индукции

$$\begin{aligned} P_{n+1}(x, y, z) = &\exists x_1, x_2, x_3, t, r, s, w \\ &(x = x_1 + x_2 + t \ \wedge \ z = s + r + w \\ &\wedge \ P_n(x_1, x_3, t) \ \wedge \ P_n(x_1, y, s) \ \wedge \ P_n(x_2, y, r) \ \wedge \ P_n(x_3, s, w)) \end{aligned}$$

и

$$P_0(x, y, z) = (x = 0 \ \wedge \ z = 0) \ \vee (x = 1 \ \wedge \ z = y).$$

Уже известным приемом четыре вхождения P_n в P_{n+1} можно заменить на одно и получить формулу длины $O(n)$ с ограниченным количеством переменных.

Формулу E_n строим также по индукции: любое число натуральное x, не превосходящее 2^{n+1}, имеет вид $x_1 + x_2$ для некоторых натуральных $x_1, x_2 \leqslant 2^n$, причем $2^x = 2^{x_1}2^{x_2}$. Поэтому можно положить

$$E_{n+1}(x, y) = \exists x_1, x_2, t, r \ (x = x_1 + x_2 \ \wedge \ P_{n+1}(r, s, z) \ \wedge \ E_n(x_1, r) \ \wedge \ E_n(x_2, s))$$

(Мы вынуждены использовать P_{n+1}, а не P_n, поскольку x_1 может быть равно 2^n, а следовательно r окажется равным 2^{2^n}, а не строго меньшим 2^{2^n}.) К сожалению, эта формула содержит не только вхождения E_n, но и вхождение P_{n+1}, поэтому ее длина более чем на константу превосходит длину E_n (даже после замены двух вхождений E_n на одно).

Этот недостаток устраняется так: будем определять одну формулу C_n от пяти переменных x, y, z, r, s, которая утверждает то же, что и P_{n+1} о числах x, y, z и то же, что E_n о числах r, s. Такую формулу длины $O(n)$ нетрудно определить по индукции, совместив индуктивные определения P_{n+1} и E_n. Формулы P_n и E_n получатся как $P_n(x, y, z) = C_{n-1}(x, y, z, 0, 1)$

и $E_n(r,s) = C_n(0,0,0,r,s)$ (эти же выражения надо использовать и при индуктивном выражении C_{n+1} через C_n). $\square$