Категориальные грамматики

Теория алгоритмов и сложность вычислений.

Степан Львович Кузнецов, Мати Рейнович Пентус,
Алексей Андреевич Сорокин

МГУ им. М. В. Ломоносова, межфакультетский курс,
весенний семестр 2017–2018 учебного года

Алгоритмы и вычислимые функции

Определение

Алгоритм — набор инструкций, описывающий последовательность действий некоторого исполнителя в зависимости от входных данных.

Примеры алгоритмов:

- Программа на любом языке программирования,

Алгоритмы и вычислимые функции

Определение

Алгоритм — набор инструкций, описывающий последовательность действий некоторого исполнителя в зависимости от входных данных.

Примеры алгоритмов:

- Программа на любом языке программирования,
- Формальные модели вычислений:
  - Машина Тьюринга,
  - Равнодоступная адресная машина
  - Алгорифм Маркова, машина Поста и т.д.
  - Лямбда-исчисление (нетипизованное).

Все эти модели вычислений эквивалентны.

Свойства алгоритмов

- Алгоритм — конечный набор инструкций,
- Можно считать, что каждый алгоритм — это слово в некотором алфавите.
- Как следствие, алгоритмы можно занумеровать.

Свойства алгоритмов

- Алгоритм — конечный набор инструкций,
- Можно считать, что каждый алгоритм — это слово в некотором алфавите.
- Как следствие, алгоритмы можно занумеровать.
- Входы алгоритмов: конструктивные объекты (имеющие конечное описание).
- Примеры входов: натуральные числа, конечные последовательности натуральных чисел, слова в некотором алфавите...

Свойства алгоритмов

- Алгоритм — конечный набор инструкций,
- Можно считать, что каждый алгоритм — это слово в некотором алфавите.
- Как следствие, алгоритмы можно занумеровать.
- Входы алгоритмов: конструктивные объекты (имеющие конечное описание).
- Примеры входов: натуральные числа, конечные последовательности натуральных чисел, слова в некотором алфавите...
- Выходы алгоритмов — также конструктивные объекты.
- Можно считать, что выходы и входы — натуральные числа.

Вычислимые функции

Определение

Вычислимая функция — функция, для которой существует вычисляющий её алгоритм.

Вычислимые функции

Определение

Вычислимая функция — функция, для которой существует вычисляющий её алгоритм.

Теорема

Существуют невычислимые функции из натуральных чисел в множество $\{0, 1\}$.

Невычислимые функции

Теорема

Существуют невычислимые функции из натуральных чисел в множество $\{0, 1\}$.

Невычислимые функции

Теорема

Существуют невычислимые функции из натуральных чисел в множество $\{0, 1\}$.

Доказательство

- Занумеруем все вычислимые функции $h_0, h_1, \ldots$.

Невычислимые функции

Теорема

Существуют невычислимые функции из натуральных чисел в множество $\{0, 1\}$.

Доказательство

- Занумеруем все вычислимые функции $h_0, h_1, \ldots$.
- Определим новую функцию

$$g(i) = \begin{cases} 1 - h_i(i), & h_i(i) \text{ определена}, \\ 0, & \text{иначе} \end{cases}$$

Невычислимые функции

Теорема

Существуют невычислимые функции из натуральных чисел в множество $\{0, 1\}$.

Доказательство

- Занумеруем все вычислимые функции $h_0, h_1, \ldots$.
- Определим новую функцию

$$g(i) = \begin{cases} 1 - h_i(i), & h_i(i) \text{ определена}, \\ 0, & \text{иначе} \end{cases}$$

- Она не совпадает ни с одной вычислимой функцией.

Невычислимые функции

Теорема

Существуют невычислимые функции из натуральных чисел в множество $\{0, 1\}$.

Доказательство

- Занумеруем все вычислимые функции $h_0, h_1, \ldots$.
- Определим новую функцию

$$g(i) = \begin{cases} 1 - h_i(i), & h_i(i) \text{ определена}, \\ 0, & \text{иначе} \end{cases}$$

- Она не совпадает ни с одной вычислимой функцией.

Следствие

Не для всех множеств натуральных чисел существует алгоритм проверки принадлежности этому множеству.

Разрешимые множества

Определение

Множество разрешимо, если существует алгоритм проверки принадлежности этому множеству.

Разрешимые множества

Определение

Множество разрешимо, если существует алгоритм проверки принадлежности этому множеству.

- Существуют неразрешимые множества натуральных чисел.

Разрешимые множества

Определение

Множество разрешимо, если существует алгоритм проверки принадлежности этому множеству.

- Существуют неразрешимые множества натуральных чисел.
- В частности, неразрешимо множество пар (p, x), таких что программа p останавливается на входе x.

Разрешимые множества

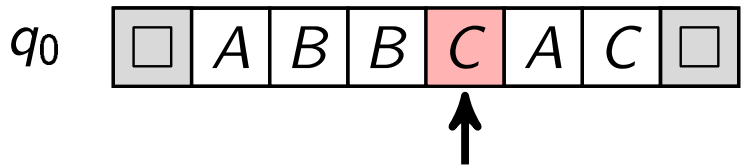
Определение

Множество разрешимо, если существует алгоритм проверки принадлежности этому множеству.

- Существуют неразрешимые множества натуральных чисел.
- В частности, неразрешимо множество пар (p, x), таких что программа p останавливается на входе x.
- Это обычно называют неразрешимостью проблемы остановки.

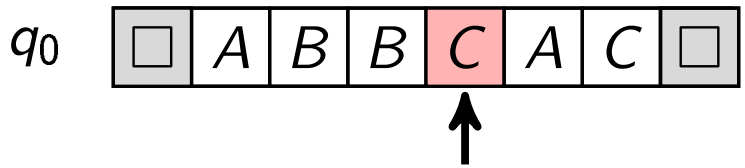
Машины Тьюринга

- Машина Тьюринга — базовая модель вычислений.
- Состоит из бесконечной ленты, указателя на текущую позицию (головки) и текущего состояния.
- На ленте написаны символы из некоторого алфавита, которые машина перезаписывает в соответствии со своей программой.



Машины Тьюринга

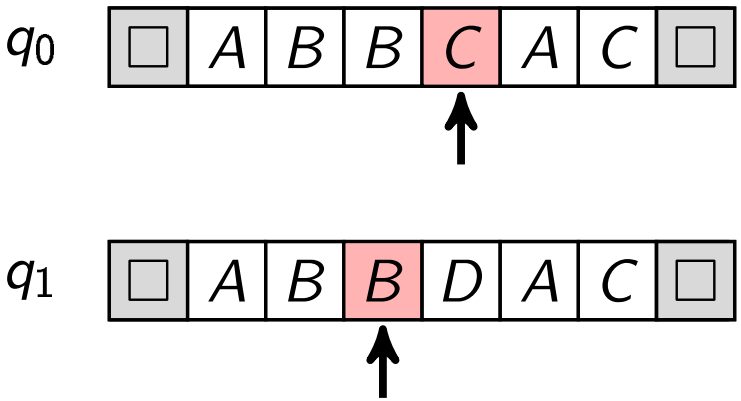
- Машина Тьюринга — базовая модель вычислений.
- Состоит из бесконечной ленты, указателя на текущую позицию (головки) и текущего состояния.
- На ленте написаны символы из некоторого алфавита, которые машина перезаписывает в соответствии со своей программой.



$$\langle q_0, C \rangle \to \langle q_1, D, L \rangle$$

Машины Тьюринга

- Машина Тьюринга — базовая модель вычислений.
- Состоит из бесконечной ленты, указателя на текущую позицию (головки) и текущего состояния.
- На ленте написаны символы из некоторого алфавита, которые машина перезаписывает в соответствии со своей программой.



$$\langle q_0, C \rangle \to \langle q_1, D, L \rangle$$

Программа машины Тьюринга

- Программа — конечный список команд вида

$$\langle q, a \rangle \to \langle q', b, L/R/N \rangle$$

Программа машины Тьюринга

- Программа — конечный список команд вида

$$\langle q, a \rangle \to \langle q', b, L/R/N \rangle$$

- Структура команды:
  - q — текущее состояние.
  - a — текущий символ.
  - q' — новое состояние.
  - b — новый символ на месте a.
  - $L/R/N$ — направление смещения головки (**L**eft/**R**ight/**N**o).

Работа машины Тьюринга

- Машины Тьюринга бывают двух типов:
  - Распознающая: ответ да/нет, на входе некоторое слово x.

Работа машины Тьюринга

- Машины Тьюринга бывают двух типов:
  - Распознающая: ответ да/нет, на входе некоторое слово x.
  - Вычисляющая: на входе некоторое слово x, на выходе некоторое слово $f(x)$.

Работа машины Тьюринга

- Машины Тьюринга бывают двух типов:
  - Распознающая: ответ да/нет, на входе некоторое слово x.
  - Вычисляющая: на входе некоторое слово x, на выходе некоторое слово $f(x)$.
- МТ начинает работу в выделенном состоянии q_S. В начале работы на ленте написано входное слово x и головка находится на его первом символе.
- В каждый момент времени МТ находит команду, соответствующую текущему символу и состоянию, после чего выполняет её.

Работа машины Тьюринга

- Машины Тьюринга бывают двух типов:
  - Распознающая: ответ да/нет, на входе некоторое слово x.
  - Вычисляющая: на входе некоторое слово x, на выходе некоторое слово $f(x)$.
- МТ начинает работу в выделенном состоянии q_S. В начале работы на ленте написано входное слово x и головка находится на его первом символе.
- В каждый момент времени МТ находит команду, соответствующую текущему символу и состоянию, после чего выполняет её.
- Распознающая МТ:
  - "да" выдаётся, когда МТ приходит в состояние q_A.
  - "нет" выдаётся, когда МТ приходит в состояние q_R.
  - МТ работает, пока одно из этих состояний не достигнуто.
- Вычисляющая МТ:
  - МТ останавливается, когда приходит в состояние q_A.
  - Ответ $f(x)$ — слово, написанное на ленте в этот момент.

Пример: прибавление 1 на машине Тьюринга

- На вход поступает двоичное число, записанное слева направо.
- В самом начале M стоит на крайнем левом бите.

Пример: прибавление 1 на машине Тьюринга

- На вход поступает двоичное число, записанное слева направо.
- В самом начале M стоит на крайнем левом бите.
- Алгоритм вычисления:
  - Дойти до правого конца числа (состояние $q_{\rightarrow}$), то есть до первого пробела.
  - Перейти в состояние q_1 (осталось добавить 1).

Пример: прибавление 1 на машине Тьюринга

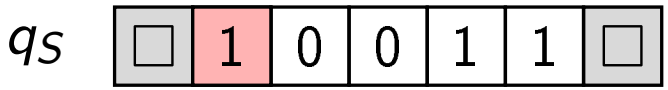
- На вход поступает двоичное число, записанное слева направо.
- В самом начале M стоит на крайнем левом бите.
- Алгоритм вычисления:
  - Дойти до правого конца числа (состояние $q_{\rightarrow}$), то есть до первого пробела.
  - Перейти в состояние q_1 (осталось добавить 1).
  - На каждом бите числа:
    - Если текущее состояние q_1, поменять текущий бит.
    - Если он был равен 0, перейти в состояние q_0, иначе q_1.

Пример: прибавление 1 на машине Тьюринга

- На вход поступает двоичное число, записанное слева направо.
- В самом начале M стоит на крайнем левом бите.
- Алгоритм вычисления:
  - Дойти до правого конца числа (состояние $q_{\rightarrow}$), то есть до первого пробела.
  - Перейти в состояние q_1 (осталось добавить 1).
  - На каждом бите числа:
    - Если текущее состояние q_1, поменять текущий бит.
    - Если он был равен 0, перейти в состояние q_0, иначе q_1.
  - Если МТ дошла до пробела, но бит переноса 1, заменить пробел на 1 и завершить работу.
  - Если текущее состояние q_0, сразу завершить работу (больше прибавлять нечего).

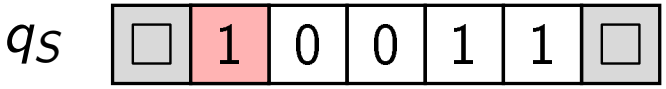
Пример: прибавление 1 на машине Тьюринга

- Входное число 10011, начальное состояние q_S.



Пример: прибавление 1 на машине Тьюринга

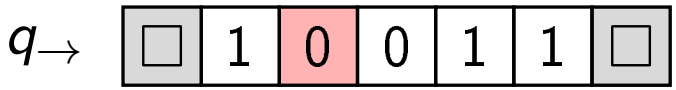
- Входное число 10011, начальное состояние q_S.



$\langle q_S, 0/1 \rangle \to \langle q_\to, 0/1, R \rangle$

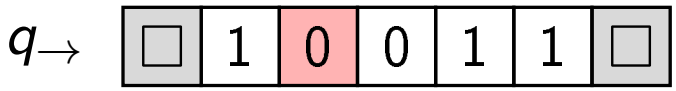
Пример: прибавление 1 на машине Тьюринга

- Входное число 10011, начальное состояние q_S.



Пример: прибавление 1 на машине Тьюринга

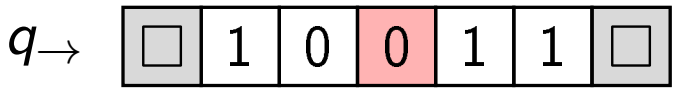
- Входное число 10011, начальное состояние q_S.



$\langle q_\rightarrow, 0/1 \rangle \rightarrow \langle q_\rightarrow, 0/1, R \rangle$

Пример: прибавление 1 на машине Тьюринга

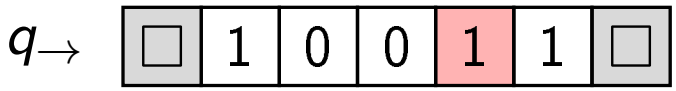
- Входное число 10011, начальное состояние q_S.



$\langle q_\rightarrow, 0/1 \rangle \rightarrow \langle q_\rightarrow, 0/1, R \rangle$

Пример: прибавление 1 на машине Тьюринга

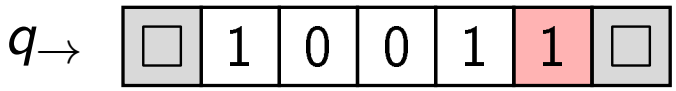
- Входное число 10011, начальное состояние q_S.



$\langle q_\rightarrow, 0/1 \rangle \rightarrow \langle q_\rightarrow, 0/1, R \rangle$

Пример: прибавление 1 на машине Тьюринга

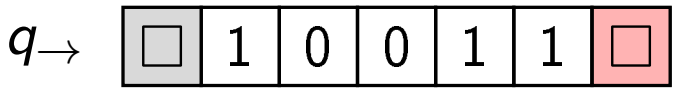
- Входное число 10011, начальное состояние q_S.



$\langle q_{\rightarrow}, 0/1 \rangle \rightarrow \langle q_{\rightarrow}, 0/1, R \rangle$

Пример: прибавление 1 на машине Тьюринга

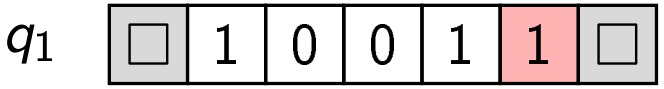
- Входное число 10011, начальное состояние q_S.



$$\langle q_\rightarrow, \square \rangle \rightarrow \langle q_1, \square, L \rangle$$

Пример: прибавление 1 на машине Тьюринга

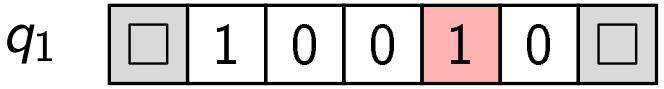
- Входное число 10011, начальное состояние q_S.



$$\langle q_1, 1 \rangle \to \langle q_1, 0, L \rangle$$

Пример: прибавление 1 на машине Тьюринга

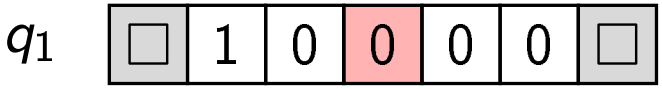
- Входное число 10011, начальное состояние q_S.



$$\langle q_1, 1 \rangle \to \langle q_1, 0, L \rangle$$

Пример: прибавление 1 на машине Тьюринга

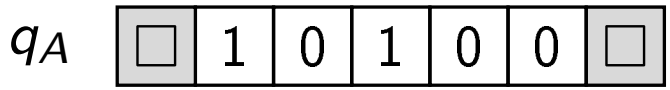
- Входное число 10011, начальное состояние q_S.



$$\langle q_1, 0 \rangle \to \langle q_A, 1, N \rangle$$

Пример: прибавление 1 на машине Тьюринга

- Входное число 10011, начальное состояние q_S.



- Финальный ответ 10100.

Алгоритмически неразрешимые задачи

- Проблема равенства слов в полугруппе неразрешима.

Алгоритмически неразрешимые задачи

- Проблема равенства слов в полугруппе неразрешима.

Неразрешимость проблемы равенства в полугруппах

Если дано конечное множество равенств вида $w_1 = w_2$, где w_1, w_2 — слова в некотором алфавите, то в общем случае невозможно проверить, можно ли множественными заменами левой части равенства на правую получить из одного слова другое.

Алгоритмически неразрешимые задачи

- Проблема равенства слов в полугруппе неразрешима.

Неразрешимость проблемы равенства в полугруппах

Если дано конечное множество равенств вида $w_1 = w_2$, где w_1, w_2 — слова в некотором алфавите, то в общем случае невозможно проверить, можно ли множественными заменами левой части равенства на правую получить из одного слова другое.

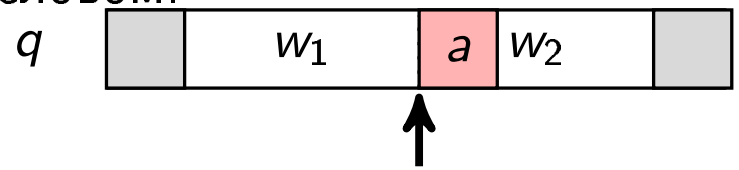
Переформулировка частного случая, Цейтин (1958)

Не существует способа проверить, можно ли из некоторого слова получить слово *aaa*, если разрешены замены:

$$
\begin{array}{rcl}
c & \leftrightarrow & ca \\
be & \leftrightarrow & cb \\
c & \leftrightarrow & ce \\
cdca & \leftrightarrow & edeae \\
daaa & \leftrightarrow & aaa
\end{array}
\qquad
\begin{array}{rcl}
ad & \leftrightarrow & da \\
bd & \leftrightarrow & db \\
edb & \leftrightarrow & de \\
caaa & \leftrightarrow & aaa
\end{array}
$$

Неразрешимость проблемы равенства в полугруппах: идея доказательства

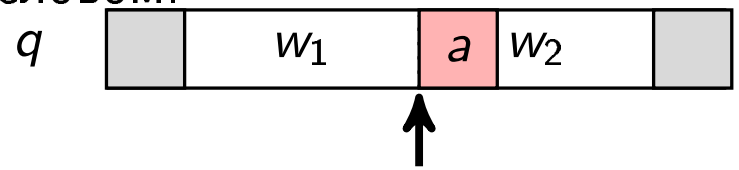
- Состояние ленты машины Тьюринга можно закодировать некоторым словом:



кодируется словом $[w_1 q a w_2]$.

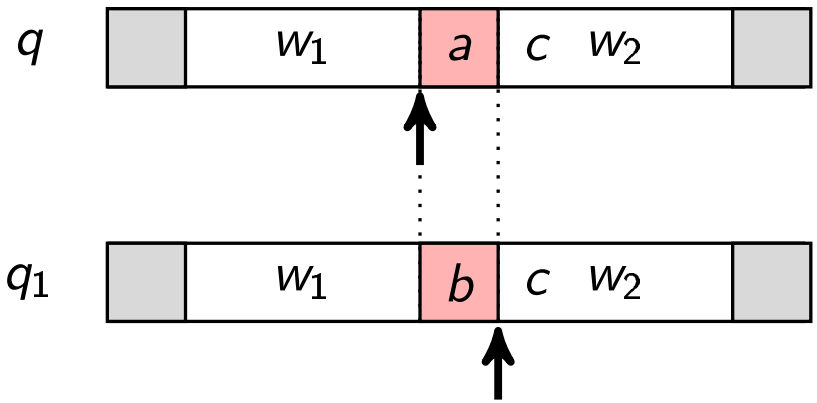
Неразрешимость проблемы равенства в полугруппах: идея доказательства

- Состояние ленты машины Тьюринга можно закодировать некоторым словом:



кодируется словом $[w_1 q a w_2]$.

- Изменения ленты можно закодировать с помощью правил:

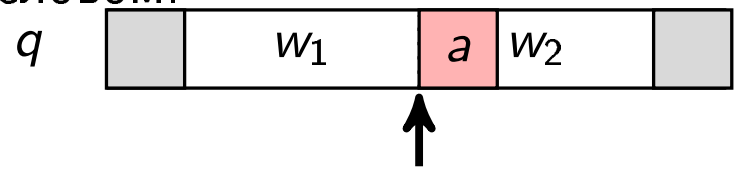


$\langle q, a\rangle \to \langle q_1, b, R\rangle$ $[w_1 q a c w_2]$

$[w_1 b q_1 c w_2]$

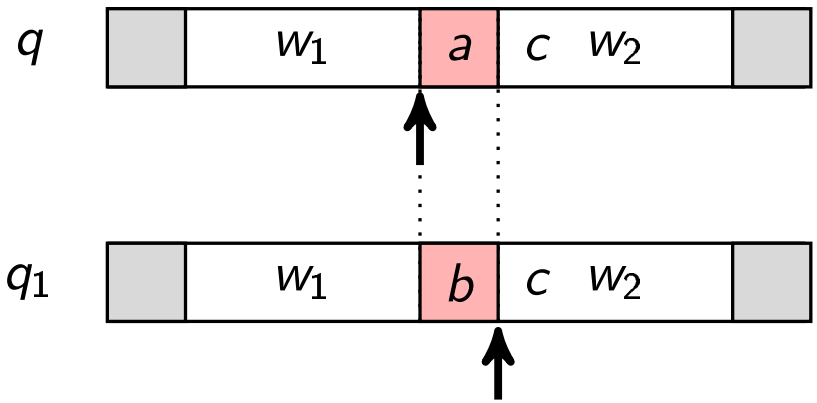
Неразрешимость проблемы равенства в полугруппах: идея доказательства

- Состояние ленты машины Тьюринга можно закодировать некоторым словом:



кодируется словом $[w_1 q a w_2]$.

- Изменения ленты можно закодировать с помощью правил:



$\langle q, a\rangle \to \langle q_1, b, R\rangle$ $[w_1 q a c w_2]$

$[w_1 b q_1 c w_2]$

- Команда $\langle q, a\rangle \to \langle q_1, b, R\rangle$ соответствует замене $qac \to bq_1c$ для любого символа c.

Неразрешимость проблемы равенства в полугруппах: идея доказательства

- С небольшими тонкостями (замены на краю ленты, двусторонние правила в односторонние) программу машины Тьюринга можно закодировать соотношениями полугруппы.

Неразрешимость проблемы равенства в полугруппах: идея доказательства

- С небольшими тонкостями (замены на краю ленты, двусторонние правила в односторонние) программу машины Тьюринга можно закодировать соотношениями полугруппы.
- Можно модифицировать распознающую МТ так, чтобы в конце своей работы она очищала ленту.
- Тогда если слово x принимается МТ тогда и только тогда, когда строку $[q_S x]$ равна $[q_A]$ по правилам полугруппы.

Неразрешимость проблемы равенства в полугруппах: идея доказательства

- С небольшими тонкостями (замены на краю ленты, двусторонние правила в односторонние) программу машины Тьюринга можно закодировать соотношениями полугруппы.
- Можно модифицировать распознающую МТ так, чтобы в конце своей работы она очищала ленту.
- Тогда если слово x принимается МТ тогда и только тогда, когда строку $[q_S x]$ равна $[q_A]$ по правилам полугруппы.
- Давайте возьмём МТ M, получающую на вход число k, и возвращающую 1, если k-ая МТ останавливается на числе k.
- Такая МТ существует (данный язык полуразрешим).

Неразрешимость проблемы равенства в полугруппах: идея доказательства

- С небольшими тонкостями (замены на краю ленты, двусторонние правила в односторонние) программу машины Тьюринга можно закодировать соотношениями полугруппы.
- Можно модифицировать распознающую МТ так, чтобы в конце своей работы она очищала ленту.
- Тогда если слово x принимается МТ тогда и только тогда, когда строку $[q_S x]$ равна $[q_A]$ по правилам полугруппы.
- Давайте возьмём МТ M, получающую на вход число k, и возвращающую 1, если k-ая МТ останавливается на числе k.
- Такая МТ существует (данный язык полуразрешим).
- Тогда равенство $[q_S x] = [q_A]$ в соответствующей ей полугруппе равносильно тому, что данная МТ принимает x.
- А эта проблема неразрешима.

Ещё неразрешимые проблемы

Неразрешимость равенства КС-грамматик

Не существует алгоритма, проверяющего, что две контекстно-свободные грамматики задают один и тот же язык.

(Канович, Кузнецов, Щедров (2017))

Не существует алгоритма, проверяющего выводимость в исчислениях $L(\backslash, !)$ и$L(/, \backslash, \cdot, !)$ (варианты исчисления Ламбека, обогащённые дополнительной связкой $!$).

Временная и пространственная сложность

- Наличие алгоритма недостаточно для практического решения задачи.
- Важны ещё затрачиваемые время и память.

Временная и пространственная сложность

- Наличие алгоритма недостаточно для практического решения задачи.
- Важны ещё затрачиваемые время и память.

Временная и пространственная сложность

- Наличие алгоритма недостаточно для практического решения задачи.
- Важны ещё затрачиваемые время и память.
- Временная сложность: число элементарных операций, затрачиваемых программой на данном входе.

Временная и пространственная сложность

- Наличие алгоритма недостаточно для практического решения задачи.
- Важны ещё затрачиваемые время и память.
- Временная сложность: число элементарных операций, затрачиваемых программой на данном входе.
- Нас интересует зависимость числа операций от длины входа:

$$T_A(n) = \max_{|x|=n} T(A, x)$$

- Здесь $T(A, x)$ – время, затрачиваемое алгоритмом A на входе x.

Элементарные операции

- Элементарные операции зависят от модели вычислений.

Элементарные операции

- Элементарные операции зависят от модели вычислений.
- Для машины Тьюринга это число тактов работы.

Элементарные операции

- Элементарные операции зависят от модели вычислений.
- Для машины Тьюринга это число тактов работы.
- Для большинства языков программирования:
  - Сравнение двух чисел.
  - Присваивание переменной значения.
  - Элементарные арифметические операции (сложение, умножение).

Элементарные операции

- Элементарные операции зависят от модели вычислений.
- Для машины Тьюринга это число тактов работы.
- Для большинства языков программирования:
  - Сравнение двух чисел.
  - Присваивание переменной значения.
  - Элементарные арифметические операции (сложение, умножение).
- При переходе от одной модели вычислений к другой сложность изменяется не более чем полиномиально (то есть число операций в одной модели не превышает многочлена от числа операций в другой)

Основные сложностные классы

Предполагается, что алгоритм вычисляет некоторую функцию $f : \Sigma^* \to \{0, 1\}$ (то есть даёт ответы "да/нет" для слов в некотором алфавите Σ).

Основные сложностные классы:

- **P** (задачи, решаемые за полиномиальное время): $T(A, x) \leqslant C|x|^k$ для некоторых констант C и k.
- **EXP** (экспоненциальное время): $T(A, x) \leqslant C2^{|x|^k}$ для некоторых констант C и k.
- **NP** (недетерминированное полиномиальное время): за полиномиальное время можно проверить, что $f(x) = y$, но не вычислить $f(x)$.

Основные сложностные классы

Предполагается, что алгоритм вычисляет некоторую функцию $f : \Sigma^* \to \{0, 1\}$ (то есть даёт ответы "да/нет" для слов в некотором алфавите Σ).

Основные сложностные классы:

- **P** (задачи, решаемые за полиномиальное время): $T(A, x) \leqslant C|x|^k$ для некоторых констант C и k.
- **EXP** (экспоненциальное время): $T(A, x) \leqslant C2^{|x|^k}$ для некоторых констант C и k.
- **NP** (недетерминированное полиномиальное время): за полиномиальное время можно проверить, что $f(x) = y$, но не вычислить $f(x)$.

Соотношения между классами:

- $\mathbf{P} \subseteq \mathbf{NP}$,
- $\mathbf{P} \subsetneq \mathbf{EXP}$,
- Предполагается, что $\mathbf{P} \subsetneq \mathbf{NP}$ (одна из "проблем тысячелетия").

Пример полиномиальной задачи

- Поиск подстроки: Даны две строки u и v, требуется проверить, является ли u подстрокой v.

Пример полиномиальной задачи

- Поиск подстроки: Даны две строки u и v, требуется проверить, является ли u подстрокой v.
- Формально: дана строка $w \in \Sigma^*$, требуется вернуть 1, если верно, что

$$w = u\#v,\ |u|_\# = |v|_\# = 0,\ \exists x, y\ (u = xvy),$$

и 0 иначе.

Пример полиномиальной задачи

- Поиск подстроки: Даны две строки u и v, требуется проверить, является ли u подстрокой v.
- Формально: дана строка $w \in \Sigma^*$, требуется вернуть 1, если верно, что

$$w = u\#v,\ |u|_\# = |v|_\# = 0,\ \exists x, y\ (u = xvy),$$

и 0 иначе.
- Идея алгоритма: будем “прикладывать” v в каждой позиции u и проверять, совпадают ли символы в соответствующих позициях.

Алгоритм поиска подстроки

Вход: Слова u, v.
Выход: **true**, если v — подстрока u, **false** — иначе.

```
m = |u|, n = |v|
for i = 0, ..., m − n do
    has_unequal = false
    for j = 0, ..., n do
        if u[i + j] ≠ v[j] then
            has_unequal = true
            break
        end if
        if not has_unequal then
            return true
        end if
    end for
end for
return false
```

- Суммарное число операций не превосходит Cmn, то есть CN^2, где N – суммарная длина входа.
- Можно сделать за $C(m+n)$ (префикс-функция, алгоритм Кнута-Морриса-Пратта).

Алгоритм проверки выводимости в КС-грамматике

- Вход: фиксированная контекстно-свободная грамматика $G = \langle N, \Sigma, P, S \rangle$ и слово $w \in \Sigma^*$.
- Выход: **True** если $w \in L(G)$ и **False** иначе.

Алгоритм проверки выводимости в КС-грамматике

- Вход: фиксированная контекстно-свободная грамматика $G = \langle N, \Sigma, P, S \rangle$ и слово $w \in \Sigma^*$.
- Выход: **True** если $w \in L(G)$ и **False** иначе.
- Напоминание: G в нормальной форме Хомского, если все правила в P имеют вид:
  - $A \to BC$, $B, C \in N - \{S\}$,
  - $A \to a$, $a \in \Sigma$,
  - $S \to \varepsilon$.

Алгоритм проверки выводимости в КС-грамматике

- Вход: фиксированная контекстно-свободная грамматика $G = \langle N, \Sigma, P, S \rangle$ и слово $w \in \Sigma^*$.
- Выход: **True** если $w \in L(G)$ и **False** иначе.
- Напоминание: G в нормальной форме Хомского, если все правила в P имеют вид:
  - $A \to BC$, $B, C \in N - \{S\}$,
  - $A \to a$, $a \in \Sigma$,
  - $S \to \varepsilon$.
- Мы будем считать, что грамматика уже дана в нормальной форме Хомского (иначе можно привести).

Алгоритм проверки выводимости в КС-грамматике

- Вход: фиксированная контекстно-свободная грамматика $G = \langle N, \Sigma, P, S \rangle$ и слово $w \in \Sigma^*$.
- Выход: **True** если $w \in L(G)$ и **False** иначе.
- Напоминание: G в нормальной форме Хомского, если все правила в P имеют вид:
  - $A \to BC$, $B, C \in N - \{S\}$,
  - $A \to a$, $a \in \Sigma$,
  - $S \to \varepsilon$.
- Мы будем считать, что грамматика уже дана в нормальной форме Хомского (иначе можно привести).
- Идея алгоритма: если $B \vdash w[i:j]$, $C \vdash w[j:k]$ и $A \to BC \in P$, то $A \vdash w[i:k]$. Здесь $w[i:j] = w[i] \dots w[j-1]$.

Алгоритм Кока-Янгера-Касами

Вход: $G = \langle N, \Sigma, P, S \rangle$ — КС-грамматика в НФ Хомского, $w \in \Sigma^*$,
Выход: **True** если $w \in L(G)$; **False** иначе.

```
▷ D — таблица размера |N| × (n + 1) × (n + 1), n = |w|
▷ вначале D заполнена нулями, в конце D[A][i][j] = 1 ⇔ A ⊢ w[i : j].
for A → a ∈ P do
    for i = 0, . . . , n − 1 do
        if w[i] == a then
            D[A][i][i + 1] = 1
        end if
    end for
end for
for k = 2, . . . , n do
    for i = 0, . . . , n − k do
        for j = 1, . . . , k − 1 do
            for (A → BC) ∈ P do
                if D[B][i][i + j] == 1 & D[C][i + j][i + k] == 1 then
                    D[A][i][i + k] = 1
                end if
            end for
        end for
    end for
end for
     return D[S][0][n]
```

Суммарное число операций равно $C|w|^3$, а константа C зависит от грамматики G, но не от слова w.

Пример работы алгоритма Кока-Янгера-Касами

Исходная грамматика: грамматика для арифметических выражений.

$$S \to S + S \qquad S \to S * S \qquad S \to (S)$$
$$S \to x \qquad S \to y \qquad S \to z$$

Пример работы алгоритма Кока-Янгера-Касами

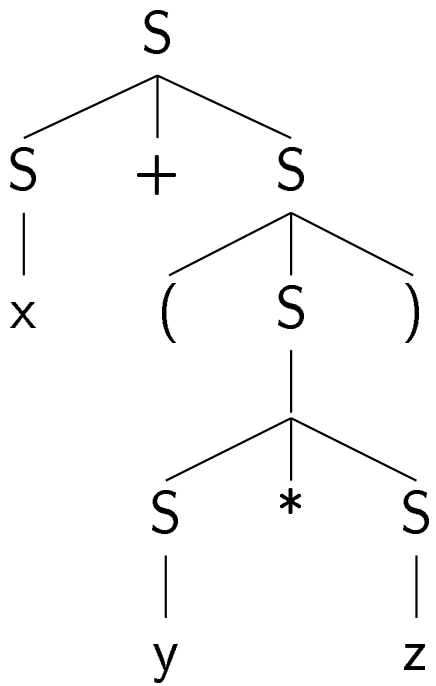
Исходная грамматика: грамматика для арифметических выражений.

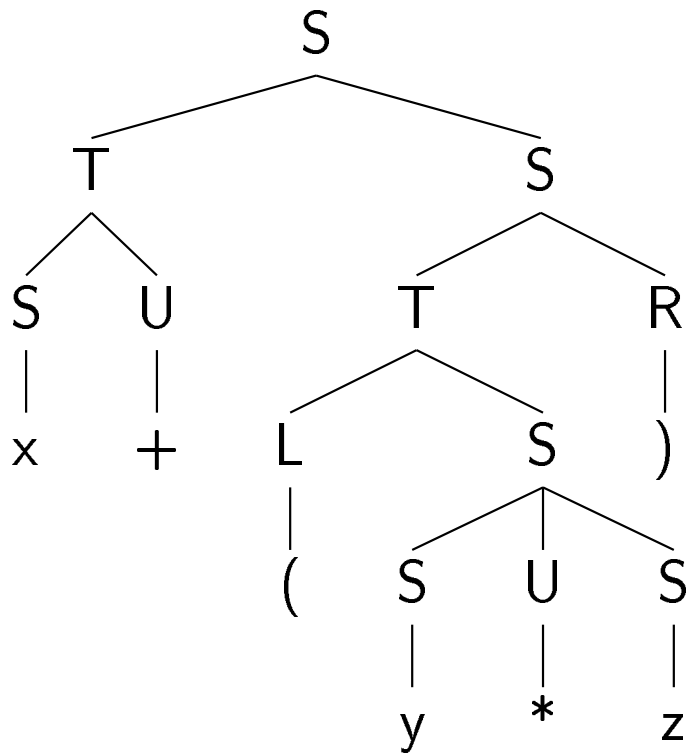
$$S \to S + S \qquad S \to S * S \qquad S \to (S)$$
$$S \to x \qquad S \to y \qquad S \to z$$

Грамматика в нормальной форме Хомского:

$$S \to TS \qquad T \to SU \qquad U \to +$$
$$U \to * \qquad S \to VR \qquad V \to LS$$
$$L \to (\qquad R \to)$$
$$S \to x \qquad S \to y \qquad S \to z$$

Пример вывода в грамматике





Проверка выводимости

Проверим, что $G \vdash (x + y) * z$, построив матрицу D.
База индукции:

$$D(S, 0, 1) = 1, \text{ т.к. } S \to x \in P \text{ и } w[0] = x,$$
$$D(U, 1, 2) = 1, \text{ т.к. } U \to + \in P \text{ и } w[1] = \text{`}+\text{'} \ldots$$

Аналогично:

$$D(L, 2, 3) = 1 \qquad D(S, 3, 4) = 1$$
$$D(U, 4, 5) = 1 \qquad D(S, 5, 6) = 1$$
$$D(R, 6, 7) = 1$$

Шаг индукции:

$$V \to LS, D(L, 2, 3) = 1, D(S, 3, 4) = 1 \Rightarrow D(V, 2, 4) = 1$$

Проверка выводимости

$$
\begin{array}{llll}
T \to SU & D(S,3,4)=1 & D(U,4,5)=1 & \Rightarrow \quad D(T,3,5)=1 \\
S \to TS & D(T,3,5)=1 & D(S,5,6)=1 & \Rightarrow \quad D(S,3,6)=1 \\
V \to LS & D(L,2,3)=1 & D(S,3,6)=1 & \Rightarrow \quad D(V,2,6)=1 \\
S \to VR & D(V,2,6)=1 & D(R,6,7)=1 & \Rightarrow \quad D(S,2,7)=1 \\
T \to SU & D(S,0,1)=1 & D(U,1,2)=1 & \Rightarrow \quad D(T,0,2)=1 \\
S \to TS & D(T,0,2)=1 & D(S,2,7)=1 & \Rightarrow \quad D(S,0,7)=1
\end{array}
$$

Таким образом, $S \vdash w[0:7] = w$, т.е. $w \in L(G)$.

Сложность проверки выводимости

- Для КС-грамматик: $O(|G||w|^3)$.
- Для широкого подкласса КС-грамматик: $O(2^{|G|}|n|)$ (LR-грамматик

Сложность проверки выводимости

- Для КС-грамматик: $O(|G||w|^3)$.
- Для широкого подкласса КС-грамматик: $O(2^{|G|}|n|)$ (LR-грамматик
- Для исчисления Ламбека: $O(2^d|n|^3)$, где n — максимальная глубина типа.
- Для грамматик Ламбека: $O(2^{|G|}|n|^3)$.