Теоретическая информатика.

Лектор — А.С. Охотин, Э.А. Гирш и Д.О. Соколов
Создатель конспекта — Глеб Минаев *

TODOs

Картиночки. 11
Написать. 16
Тут должна быть пара страшных алгоритмов, которые мне лень писать. (см.) 26
Смущает фраза “’эффективно построены” + ассимптотика? 27
Дописать по см. 29
Дописать по см. 29
Надо написать? 31
На лекции было сопосатвление алгоритма, а не единый на всех алгоритм. 31
Тут есть нюанс, ... 35

Содержание

1 Вычислимость **1**

2 Формальные зыки и автоматы **6**
2.1 Регулярные выражение и равносильность конечным автоматам 9
2.2 Разные действия над автоматами 13
2.3 Минимальные ДКА 13
2.4 Двухсторонние автоматы 15
2.5 Вероятностные конечные автоматы 17
2.6 Формальные грамматики 20
2.7 Операции над грамматиками. 21
2.8 Лемма о накачке 23
2.9 Неразрешимость задач для грамматик 27

3 Сложность вычислений **29**
3.1 NP-полные задачи 34

Литература:

- ...

Страницы курса:

- Часть курса, прочитанная А.С. Охотиным.

*Оригинал конспекта расположен на GitHub. Также на GitHub доступен репозиторий с другими конспектами.

1 Вычислимость

Определение 1. *Машина Тьюринга* — это реализация понятия вычисления. Она заключается в том, что имеется бесконечная в обе стороны клетчатая лента с записанным на ней конечным словом (по букве на клетку) — входные данные вычисления — и головка — вычисляющий аппарат, которая в каждый момент времени смотрит на какую-то клетку ленты и имеет в себе некоторое внутреннее состояние вычислений. Каждым своим действием она читает символ в клетке, на которую смотрит, и в зависимости от прочитанного символа и внутреннего состояния в данный момент она пишет в клетку, на которую смотрит новый символ, стирая старый, переходит на новую клетку и меняет внутренне состояние.

Формально машина Тьюринга M — это совокупность

- входного алфавита Σ — (конечного) алфавита входных данных,
- рабочего алфавита Γ — (конечного) алфавита, над которым мы работаем, что $\Gamma \supseteq \Sigma \sqcup \{\text{␣}\}$,
- конечного множества (внутренних) состояний Q,
- начального состояния $q_0 \in Q$,
- функции переходов $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1; +1\}$
- и дополнительных объектов и условий в зависимости от предназначения машины.

Входными данными программы является конечная строка $s_1 \dots s_l$, которая превращается в запись $(a_i)_{i \in \mathbb{Z}}$ на ленте по правилу

$$a_i = \begin{cases} s_{i+1} & \text{если } 0 \leqslant i \leqslant l-1, \\ \text{␣} & \text{иначе.} \end{cases}$$

Состояние каждого момент вычисления — это совокупность

- состояния ленты $(a_i)_{i \in \mathbb{Z}}$, где $a_i \in \Sigma$ и все a_i кроме конечного числа элементов являются пробелом ␣,
- положения головки $n \in \mathbb{Z}$ на ленте и
- состояния головки $q \in Q$.

Состояние начального момента вычислений образуется из

- состояния ленты, полученного из входных данных,
- положения головки по умолчанию $n = 0$
- начального состояния головки q_0.

Каждым шагом состояние $((a_i)_{i \in \mathbb{Z}}, n, q)$ заменяется на состояние $((a'_i)_{i \in \mathbb{Z}}, n', q')$, где $a'_i = a_i$ для всех $i \in \mathbb{Z} \setminus \{n\}$ и

$$(q', a'_n, n' - n) := \delta(q, a_n).$$

Далее есть два вида предназначений машины:

1. **Распознование свойства.** В этом случае мы хотим уметь распознавать разные конечные строки, т.е. чтобы машина отвечала “да” или “нет” на вопрос “Лежит ли эта строка в заданном семействе?”. В таком случае у машины выделяются *принимающее состояние* $q_{acc} \in Q$ и *отвергающее состояние* $q_{rej} \in Q$, и когда машина переходит в одно из этих двух состояний, вычисления прекращаются. В таком случае $\delta(q, a)$ должно быть не определено в случае $q \in \{q_{acc}; q_{rej}\}$.

2. **Вычисление функции.** В этом случае мы хотим вычислять значение некоторой функции $M : \Sigma^* \to \Sigma^*$, т.е. чтобы машаина после некоторого количества действий говорила “Готово.” и оставляла на ленте конечное слово в том же формате, в котором производится ввод. В таком случае у машины выделяется *состояние останова* $q_{halt} \in Q$, и когда машина переходит в это состояние, вычисления прекращаются, а на ленте должно остаться слово в правильном формате. В таком случае $\delta(q, a)$ должно быть не определено в случае $q = q_{halt}$.

Замечание. Фактически множество результатов работы машины состоит из переходов в одно из терминальных состояний (q_{acc}, q_{rej} или q_{halt}) и попадания в “вечный цикл”.

Теорема 1. *В определении машина Тьюринга множество возможных смещений головки* $\{+1; -1\}$ *можно сменить на некоторое* $S \subseteq \mathbb{Z}$.

1. *Тогда если* S *конечно и всякое целое число представляется в виде суммы хотя бы одного значения (возможно, с повторами) из* S*, то определение получается равносильным.*

2. *То же самое верно для всякого* S *(не обязательно конечного).*

Доказательство.

1. Действительно, пусть у нас имеется машина с множеством возможных сдвигов S_1, а у нас есть множество возможных сдвигов S_2, что всякое число из S_1 представляется в виде суммы некоторых (хотя бы одного и, возможно, с повторениями) членов из S_2. Тогда для всякой перехода между состояниями $q_1 \to q_2$ можно создать такие фальшсостояния, что смещение на значение из S_1 будет заменено на несколько смещений на значения из S_2 с тем же итогом. Таким образом мы можем всякую машину на S_1 поменять на машину на S_2 и большим числом состояний.

   Таким образом множества смещений S_1 и S_2 реализуют равносильные модели, если всякое значение из S_1 расскладывается в сумму значений S_2 и наоборот. При для $S_1 = \{+1; -1\}$ переход $S_2 \to S_1$ очевиден, а переход $S_1 \to S_2$ означает, что 1 и -1 раскладываются в суммы некоторых чисел из S_2, что равносильно разложимости всякого целого.

2. Несложно понять, что в каждой конкретной машине Тьюринга множество $Q \times \Sigma$ конечно, а значит S можно заменить на конечное подмножества, что задача для данной машины не изменится.

□

Следствие 1.1. *Вместо* $\{+1; -1\}$ *для удобства можно также подразумевать* $\{+1; 0; -1\}$.

Замечание. В данный момент мы будем рассматривать только задачу распознавания свойства.

Определение 2. *Язык машины* M или *язык, распознаваемый машиной* M, — это множество $L(M) \subseteq \Sigma^*$ входных строк, которые машина принимает. Т.е. это множество всех конечных строк $w \in \Sigma^*$ таких, что машина M на входе w завершает работу и принимает данный вход.

Язык, распознаваемый какой-нибудь машиной, называется *рекурсивно-перечислимым*.

Язык, распознаваемый какой-нибудь машиной, которая останавливается на любом входе, называется *рекурсивным*.

Пример 1. Пусть $\Sigma = \{a; b\}$, $L = \{a^n b^n\}_{n \geqslant 0}$. Тогда L распознаётся машиной Тьюринга (которая причём останавливается на всяком входе!).

Действительно, давайте напишем машину, которая будить ездить из одного конца нашей строки в другую и стирать a справа и b слева, и, если получит пустую строку, примет вход, а если поймет, что в какой-то момент получается что-то неправильное, то отвергнет вход. Формально, возьмём $Q = \{q_0; q_1; q_2; q_3; q_{acc}; q_{rej}\}$ и $\Gamma = \{a; b; \text{␣}\}$, а функцию перехода опишем следующей таблицей:

$Q \backslash \Gamma$	a	b	␣
q_0	q_1, ␣, $+1$	q_{rej}	q_{acc}
q_1	q_1, a, $+1$	q_1, b, $+1$	q_2, ␣, -1
q_2	q_{rej}	q_3, ␣, -1	q_{rej}
q_3	q_3, a, -1	q_3, b, -1	q_0, ␣, $+1$

Т.е. во с помощью q_0 и q_2 мы стираем a слева и b справа соответственно, а с помощью q_1 и q_2 перемещаемя до конца вправо и влево соответственно.

Теорема 2. *Есть язык, нераспознаваемый никакой машиной Тьюринга.*

Доказательство. Несложно видеть, что для всякого конечного непустого алфавита Σ количество $|\Sigma^*|$ конечных строк над Σ счётно, а значит языков над Σ континуум. При этом машин Тьюринга с входным алфавитом Σ счётное число. Значит почти все языки не распознаются машинами Тьюринга.

При этом есть довольно простые конкретные примеры нераспознаваемых языков. □

Определение 3. *Запись $\sigma(M)$ машины Тьюринга M* — это конечной битовая строка, созданная по следующему правилу. Пусть у машины $M = (\Sigma, \Gamma, Q, q_0, \delta, q_{acc}, q_{rej})$

- $\Sigma = \{a_1; \ldots; a_l\}$,
- $\Gamma = \Sigma \cup \{a_{l+1}; \ldots; a_m\}$, где причём $a_m = \text{␣}$,
- $Q = \{q_0; \ldots; q_{n-1}\}$, где причём $q_{acc} = q_{n-2}$, $q_{rej} = q_{n-1}$.

Тогда запишем всё в унарной системе счисления, т.е. (всякий абстрактный) массив числовых данных будет храниться в виде последовательностей единиц, длины которых будут равняться соответствующим значениям массиве, разделённых нулями. Тогда машина M будет записана строкой

$$\sigma(M) := 1^l\,0\,1^m\,0\,1^n\,0 \prod_{\delta(q_i, a_j) = q_{i'}, a_{j'}, d} 1^i\,0\,1^j\,0\,1^{i'}\,0\,1^{j'}\,0\,1^{d+1}$$

(где умножение, безусловно, — конкатенация, а $\prod$ — конкатенация нескольких строк).

Определение 4. Языки L_0 и L_1 — это

$$L_0 := \{\sigma(M) \mid \sigma(M) \notin L(M)\} \qquad \text{и} \qquad L_1 := \{\sigma(M) \mid \sigma(M) \in L(M)\}.$$

Теорема 3.

1. *L_0 не рекурсивно-перечислимый.*
2. *L_1 рекурсивно-перечислимый.*
3. *L_1 не рекурсивный.*

Доказательство.

1. Предположим противное, т.е. есть машина M, распознающая L_0. Тогда если M принимает $\sigma(M)$, то $\sigma(M) \in L(M)$, но тогда $\sigma(M) \notin L$. Если же M не принимает $\sigma(M)$, то $\sigma(M) \notin L(M)$, но тогда $\sigma(M) \in L$. Противоречие наподобие парадокса Рассела. Значит нет никакой машины, распознающей L_0.

2. Неформально опишем машину, которая будет распознавать L_1. Идея машины заключается в том, что получая на вход $\sigma(M)$, после этого запсанного кода машины M она запишет код начального состояния q_0 машины M, а затем код $\sigma(M)$ как входную строчку для M. После этого она начнёт моделировать работу машины M на входе $\sigma(M)$. Состояние головки будет писаться перед клеткой, на которую головка смотрит. Если места слева будет не хватать, то вся запись будет просто сдвигаться вправо.

3. Покажим, что всякая машина, распознающая L_1 зацикливается при некотором входе. Предположим противное, т.е. есть машина $\widetilde{M}$, которая распознаёт L_1 и всегда остансавливается. Тогда построим $\widehat{M}$ по алгоритму:

   - Проверить правда ли, что на было подано некоторое $\sigma(M)$. Если нет, то отвергнуть.
   - Работать как $\widetilde{M}$ на $\sigma(M)$. Если $\widetilde{M}$ принимает, то отвергнуть. Если отвергает — принять.

   Тогда $\widehat{M}$ распознаёт L_0 — противоречие. Значит машины $\widetilde{M}$ не существует.

□

Теорема 4. *Язык*

$$L_\varnothing := \{\sigma(M) \mid L(M) = \varnothing\}$$

не рекурсивно-перечислимый.

Доказательство. Предположим противное, т.е. есть машина $M_\varnothing$, которая распознаёт $L_\varnothing$. Тогда построим машину M_0 по следующему алгоритму.

1. Проверить правда ли, что на было подано некоторое $\sigma(M)$. Если нет, то отвергнуть.

2. Построить $\sigma(M')$, где машина M' работает по следующему алгоритму.

   - Стереть входную строку.
   - Написать $\sigma(M)$.
   - Запустить M.

3. Запустить $M_\varnothing$ на $\sigma(M')$.

M' принимает любую строку, если M принимает $\sigma(M)$, отвергает любую строку, если M отвергает $\sigma(M)$, и зацикливается, если M зацикливается на $\sigma(M)$. Следовательно $M_\varnothing$ примет $\sigma(M')$ тогда и только тогда, когда $\sigma(M) \notin L(M)$. Таким образом M_0 распознаёт L_0. □

Определение 5. Для всякого свойства P языков можно определить язык

$$L_P := \{\sigma(M) \mid L(M) \text{ обладает свойством } P\}.$$

Можно считать, что свойство есть некоторое множество языков (т.е. подмножество 2^{Σ^*}), а обладание свойством означает содержание в нём как в множестве.

Свойство называется *тривиальным*, если либо ни один рекурсивно-перечислимый язык, либо все рекурсивно-перечислимые языки обладают этим свойством.

Пример 2.

1. Если P — это “непустота”, то

$$L_P := \{\sigma(M) \mid L(M) \neq \varnothing\}.$$

   Такой язык рекурсивно-перечислим. Но нерекурсивно.

2. $$L_P := \{\sigma(M) \mid M \text{ отвергает конечное число программ на C++}\}.$$

3. $$L_P := \{\sigma(M) \mid L(M) = L_0\}.$$

   Как мы уже знаем $L_P = \varnothing$, т.е. данное свойство тривиально.

Теорема 5 (Райса)**.** *Всякое нетривиальное свойство нерекурсивно. Точнее для всякого нетривиального свойства P язык L_P нерекурсивен.*

Доказательство. Пусть дано нетривиальное свойство $P \subseteq 2^{\Sigma^*}$ рекурсивно-перечислимых языков. Предположим противное: есть машина M_P, которая распознаёт L_P (не зацикливается и принимает только язык L_P).

Предположим, что $\varnothing \notin P$. Тогда есть машина $\widehat{M}$, что $L(\widehat{M}) \in P$. Тогда построим машину Тьюринга M_1, принимающую в себя $\sigma(M)$ некоторой машины Тьюринга M, со следующим алгоритмом:

1. Построить машину Тьюринга $\widetilde{M}$ по следующему описанию.
   (a) Сохранить вход w.
   (b) Запустить M на $\sigma(M)$.
   (c) Если отвергнет, зациклиться. Если примет, запустить $\widehat{M}$ на w.
2. Запустить M_P на $\sigma(\widetilde{M})$.

Заметим, что если $\sigma(M) \notin L(M)$, то машина $\widetilde{M}$ зацикливается на любом входе, т.е. $L(\widetilde{M}) = \varnothing$. Иначе $\widetilde{M}$ совпадает с $\widehat{M}$, т.е. $L(\widetilde{M}) = L(\widehat{M})$. Таким образом

$$L(\widetilde{M}) \in P \quad \Longleftrightarrow \quad \sigma(M) \in L(M).$$

Таким образом ответ M_p на входе $\sigma(\widetilde{M})$ есть ответ на вопрос принадлежности $\sigma(M)$ множеству $L(M)$. Таким образом M_1 не зацикливается и распознаёт язык L_1, что противоречит ранее доказанным утверждениям.

Теперь предположим $\varnothing \in P$. Тогда есть машина $\widehat{M}$, что $L(\widehat{M}) \notin P$. По аналогии можно построить машину, которая распознаёт L_0. □

2 Формальные зыки и автоматы

Определение 6. *Алфавит* Σ — фиксированный (контекстом) набор элементов (*символов*). Мы будем рассматривать только конечные алфавиты.

Строка — (если не оговорено обратное, конечная) последовательность символов из алфавита. Строки обозначаются либо просто как переменные, т.е. w, либо в виде

$$a_1 \dots a_n,$$

где a_i — символы из алфавита Σ. Строка, где $n = 0$, называется *пустой строкой* и обозначается ε. Значение n называется длиной строки w и обозначается $|w|$. Количество вхождение некоторого символа a (т.е. количество индексов i, что $a_i = a$) обозначается $|w|_a$. Множество всех строк обозначается Σ^*.

Конкатенация — бинарная операция на строках, определяемая по правилу

$$(a_1 \dots a_n, b_1 \dots b_m) \mapsto a_1 \dots a_n b_1 \dots b_m.$$

Конкатенация строк u и v обозначается uv или $u{\cdot}v$. Множество строк с операцией конкатенации и выделенной ε образуют моноид.

Обращение строки — унарная операция на строках, определяемая по правилу

$$a_1 \dots a_n \mapsto a_n \dots a_1.$$

Обращение строки w обозначается w^R.

Язык — множество строк, т.е. подмножество Σ^*.

Определение 7. *Детерминированный конечный автомат* (также "ДКА" или "DFA") A — это совокупность

- входного алфавита Σ,
- конечного множества состояний Q,
- начального состояния $q_0 \in Q$,
- множество принимающих состояний $F \subseteq Q$,
- функция перехода $\delta : Q \times \Sigma \to Q$.

Входными данными является конечная строка $w = s_1 \dots s_n$. Состояние вычисления после k шагов — это некоторое состояние автомата $q_k \in Q$. Соответственно, начальное состояние вычисления (оно же состояние вычисления после 0 шагов) — начальное состояние автомата q_0. Каждым шагом состояние q_k заменяется на $q_{k+1} := \delta(q_k, s_{k+1})$. Т.е. формально вычисление — это последовательность состояний $(q_k)_{k=0}^n$, где

- q_0 — начальное состояние автомата,
- $q_{k+1} := \delta(q_k, s_{k+1})$.

Состояние q_n называется *результатом вычисления*. Также говорят, что автомат принимает строку w, если $q_n \in F$, и отвергает в ином случае.

Язык автомата A или *язык, распознаваемый автоматом* A, — это множество $L(A)$ всех строк, распознаваемых (принимаемых) автоматом A. Язык, распознаваемый хоть каким-нибудь автоматом называется *регулярным*.

Также в качестве удобства обозначений определим функцию δ^* на $Q \times \Sigma^*$ как

$$\delta^*(q, a_1 \dots a_n) := \delta^*(\dots \delta(q, a_1), \dots, a_n),$$

т.е. δ^* задаётся рекурсивно как

$$\delta^*(q, \varepsilon) = q, \qquad \delta^*(q, aw) = \delta^*(\delta(q, a), w).$$

Иногда мы будем писать δ, подразумевая δ^*.

Замечание. Детерминированные конечные автоматы удобно изображать в виде ориентированного графа. Пусть Q — вершины графа, и из каждой вершины q ведёт по $|\Sigma|$ рёбер: ребро из q с меткой s (для всякого $s \in \Sigma$) ведёт в $\delta(q, s)$. Также небольшой стрелочкой ведущей из ниоткуда в q_0 удобно помечать q_0 как начальную вершину, и удобно обвести каждую вершину из F, чтобы пометить её как принимающую.

Замечание. Несложно видеть, что

$$L(A) = \{w \in \Sigma^* \mid \delta_A^*(q_{A,0}, w) \in F_A\}.$$

Это также значит, что $w \in L(A)$ тогда и только тогда, когда при проходе в графе A по пути, соответствующему слову w, мы попадаем в принимающее состояние. Говоря иначе, есть некоторая последовательность состояний $q_0, \ldots, q_n$, что

- $n = |w|$,
- $q_0 = q_{A,0}$ — начальное состояние автомата,
- $q_{k+1} = \delta_A(q_k, a_{k+1})$,
- $q_n \in F_A$.

Теорема 6. *Язык $L := \{a^n b^n\}_{n \in \mathbb{N}}$ нерегулярен.*

Доказательство. Предположим имеется ДКА A, что $L(A) = L$. Заметим, что тогда строка $a^{|Q|}b^{|Q|} \in L$. Пусть вычисление этой строки имеет вид $q_0 \ldots q_{|Q|} \ldots$ (понятно, что $q_{|Q|}$ — состояние вычисления после последней буквы a). По принципу Дирихле есть из состояний q_0, …, $q_{|Q|}$ есть два совпадающих; пусть это будут q_i и q_j $(i < j)$. Тогда покажем, что $a^{|Q|-(j-i)}b^{|Q|}$ тоже распознаётся (хотя не должна). Заметим, что процесс вычисления букв a будет иметь вид

$$q_0 \ldots q_i = q_j \ldots q_{|Q|}.$$

Таким образом состояние после прочтения последней буквы a будет тем же, а значит состояние после прочтения всей строки тоже будет тем же, а значит принимающим. □

Определение 8. Недетерминированный конечный автомат (также “НКА” или “NFA”) — это совокупность

- входного алфавита Σ,
- конечного множества состояний Q,
- множества начальных состояния $S \subseteq Q$,
- множество принимающих состояний $F \subseteq Q$,
- функция перехода $\delta : Q \times \Sigma \to 2^Q$.

Входными данными является конечная строка $w = s_1 \ldots s_n$. Состояние вычисления после k шагов — это некоторое множество состояний автомата $S_k \in Q$. Соответственно, начальное состояние вычисления (оно же состояние вычисления после 0 шагов) — множество начальных состояние автомата $S_0 = S$. Каждым шагом состояние S_k заменяется на $S_{k+1} := \bigcup_{q \in S_k} \delta(q, s_{k+1})$. Т.е. формально вычисление — это последовательность множеств состояний $(S_k)_{k=0}^n$, где

- $S_0 = S$ — начальное состояние автомата,

- $S_{k+1} := \bigcup_{q \in S_k} \delta(q, s_{k+1})$.

Множество состояний S_n называется *результатом вычисления*. Также говорят, что автомат принимает строку w, если $S_n \cap F \neq \varnothing$, и отвергает в ином случае.

Язык автомата A или *язык, распознаваемый автоматом* A, — это множество $L(A)$ всех строк, распознаваемых (принимаемых) автоматом A. Язык, распознаваемый хоть каким-нибудь автоматом называется *регулярным*.

Также в качестве удобства обозначений определим функцию δ^* на $2^Q \times \Sigma^*$ рекурсивно как

$$\delta^*(T, w) = \bigcup_{q \in T} \delta^*(q, w), \qquad \delta^*(\{q\}, \varepsilon) = \{q\}, \qquad \delta^*(\{q\}, aw) = \delta^*(\delta(q, a), w).$$

Иногда мы будем писать δ, подразумевая δ^*.

Замечание. Недетерминированные конечные автоматы удобно изображать в виде ориентированного графа. Пусть Q — вершины графа, и из каждой вершины q ведёт некоторое количество рёбер: по ребру из q с меткой s (для всякого $s \in \Sigma$) ведёт в каждую вершину из $\delta(q, s)$. Также небольшими стрелочками ведущими из ниоткуда в каждую вершину из S удобно помечать S как множество начальных вершин, и удобно обвести каждую вершину из F, чтобы пометить её как принимающую.

Замечание. Несложно видеть, что $w \in L(A)$ тогда и только тогда, когда есть путь в графе A, соответствующий строке w, что конечное состояние является принимающим. Говоря иначе, есть некоторая последовательность состояний $q_0, \ldots, q_n$, что

- $n = |w|$,
- $q_0 \in S_A$,
- $q_{k+1} \in \delta_A(q_k, a_{k+1})$,
- $q_n \in F_A$.

С другой стороны это также равносильно тому, что $\delta_A^*(S_A, w) \in F_A$.

Теорема 7. *У всякого недетерминированного автомата A есть детерминированный автомат A', что $L(A') = L(A)$.*

Доказательство. Пусть дано, что $A = (\Sigma, Q, S, \delta, F)$. Тогда определим

$$A' = (\Sigma, 2^Q, S, \delta', \{T \subseteq Q \mid T \cap F \neq \varnothing\}),$$

где δ' определяется по правилу

$$\delta'(T, s) := \delta(T, s) = \bigcap_{q \in T} \delta(q, s).$$

Проще говоря, A' имитирует A, запоминая в каком множестве состояний мы находимся в каждый момент и эмулируя правильный переход к новому множеству состояний. Формально говоря, несложно показать по индукции по $|w|$, что множество состояний автомата A после прочтения слова w является состоянием автомата A' после прочтения того же слова w. □

Замечание 1. Можно на языках ввести следующие операции.

- $K \cup L$, $K \cap L$, $K \triangle L$, $\overline{L}$ — стандартные операции на множествах.
- Конкатенация. $KL = K \cdot L := \{uv \mid u \in K \wedge v \in L\}$.
- Конкатенация нескольких копий. $L^k := L \cdot \cdots \cdot L$. $L^0 = \{\varepsilon\}$.
- Конкатенация любого конечного числа копий или звёздочка Клини. $L^* = \bigcup_{k=0}^{\infty} L^k$.

В таком случае $(2^{\Sigma^*}, \cup, \cdot, \varnothing, \{\varepsilon\})$ — полукольцо.

2.1 Регулярные выражение и равносильность конечным автоматам

Определение 9. Рекурсивно определим понятие *регулярного выражения.*

- $\varnothing$ — регулярное выражение.
- a, где $a \in \Sigma$, — регулярное выражение.
- $(\varphi\psi)$, $(\varphi|\psi)$ и φ^*, где φ и ψ есть регулярные выражения, — регулярные выражения.

Множество регулярных выражений над алфавитом Σ обозначается $\mathrm{RE}(\Sigma)$.

Теперь рекурсивно определим *язык регулярного выражения.*

- $L(\varnothing) = \varnothing$.
- $L(a) = \{a\}$.
- $L(\varphi\psi) = L(\varphi)L(\psi)$.
- $L(\varphi|\psi) = L(\varphi) \cup L(\psi)$.
- $L(\varphi^*) = L(\varphi)^*$.

Теорема 8. *Язык распознаётся конечным автоматом тогда и только тогда, когда задаётся регулярным выражением.*

Определение 10. Недетерминированный конечный автомат с ε-переходами (также "ε-НКА" или "ε-NFA") — это совокупность

- входного алфавита Σ,
- конечного множества состояний Q,
- множества начальных состояния $S \subseteq Q$,
- множество принимающих состояний $F \subseteq Q$,
- функция перехода $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$.

ε-замыканием состояния q называется множество состояний

$$E(q) := \{p \mid \exists k \in \mathbb{N}\colon \exists q_0, \ldots, q_k \in Q\colon q_0 = q \wedge q_k = p \wedge \forall i = 1, \ldots, k\ q_i = \delta(q_{i-1}, \varepsilon)\}.$$

Также по аналогии ε-замыканием множества состояния T называется множество состояний

$$E(T) := \bigcup_{q \in T} E(q),$$

откуда, в частности, получается "рекурсивное определение"

$$E(T) := T \cup \bigcup_{q \in T} E(\delta(q, \varepsilon)).$$

Входными данными является конечная строка $w = s_1 \ldots s_n$. Состояние вычисления после k шагов — это некоторое множество состояний автомата $S_k \in Q$. Соответственно, начальное состояние вычисления (оно же состояние вычисления после 0 шагов) — множество начальных состояние автомата $S_0 = E(S)$. Каждым шагом состояние S_k заменяется на $S_{k+1} := \bigcup_{q \in S_k} E(\delta(q, s_{k+1}))$. Т.е. формально вычисление — это последовательность множеств состояний $(S_k)_{k=0}^n$, где

- $S_0 = E(S)$ — начальное состояние автомата,
- $S_{k+1} := \bigcup_{q \in S_k} E(\delta(q, s_{k+1}))$.

Множество состояний S_n называется *результатом вычисления*. Также говорят, что автомат принимает строку w, если $S_n \cap F \neq \varnothing$, и отвергает в ином случае.

Язык автомата A или *язык, распознаваемый автоматом* A, — это множество $L(A)$ всех строк, распознаваемых (принимаемых) автоматом A. Язык, распознаваемый хоть каким-нибудь автоматом называется *регулярным*.

Также в качестве удобства обозначений определим функцию δ^* на $2^Q \times \Sigma^*$ рекурсивно как

$$\delta^*(T, w) = \bigcup_{q \in T} \delta^*(q, w), \qquad \delta^*(\{q\}, \varepsilon) = E(\{q\}), \qquad \delta^*(\{q\}, aw) = \delta^*\left(\bigcup_{p \in E(q)} \delta(p, a), w\right).$$

Иногда мы будем писать δ, подразумевая δ^*.

Замечание. Недетерминированные конечные автоматы с ε-переходами удобно изображать точно также как обычные НКА, но на стрелках (ориентированных рёбрах) переходов теперь можно писать и новодобавленный символ ε.

Замечание. Несложно видеть, что $w \in L(A)$ тогда и только тогда, когда есть некоторое представление $w = s_1 \dots s_n$, где $s_i \in \Sigma \cup \{\varepsilon\}$, и некоторая последовательность состояний $q_0, \dots, q_n$, что

- $q_0 \in S_A$,
- $q_{k+1} \in \delta_A(q_k, s_{k+1})$,
- $q_n \in F_A$.

С другой стороны это также равносильно тому, что $\delta_A^*(S_A, w) \in F_A$.

Лемма 9. *Всякое регулярное выражение можно заменить на ε-НКА, порождающий тот же язык.*

Доказательство. Давайте построим для каждого регулярного выражения ε-НКА с ровно одним начальным состоянием и ровно одним принимающим состоянием. И будем мы это делать по рекурсивному построению регулярного выражения:

- Автомат для регулярного выражения $\varnothing$ будет состоять только из одного начального и одного принимающего состояний, не иметь других состояний и не иметь никаких переходов.
- Автомат для регулярного выражения a ($a \in \Sigma$) будет состоять только из одного начального и одного принимающего состояний, не иметь других состояний и иметь единственный переход от начального состояния к принимающему по символу a.
- Автомат для регулярного выражения $\varphi\psi$ будет получаться проведением перехода по ε от принимающего состояния φ к начальному состоянию ψ. Соединённые состояния теряют свои роли (т.е. больше не являются принимающей и начальной соответственно).
- Автомат для регулярного выражения $\varphi|\psi$ будет получаться проведением переходов по ε от начального состояния строимого автомата к начальным состояниям автоматов φ и ψ и переходов по ε от принимающих состояний автоматов φ и ψ к принимающему состоянию строимого автомата. Начальные и принимающие состояния старых автоматов теряют свои роли.

- Автомат для регулярного выражения $\varphi*$ будет получаться проведением перехода по ε от начального состояния строимого автомата к начальному состоянию автомата φ, перехода по ε от принимающего состояния автомата φ к начальному состоянию строимого автомата и перехода по ε от начального состояния строимого автомата к принимающему состоянию строимого автомата. Начальные и принимающие состояния старого автомата теряют свои роли.

Картиночки.

Несложно видеть по индукции по построению выражения, что строимые автоматы, действительно, строят язык соответствующих регулярных выражений. □

Лемма 10. *Всякий ε-НКА можно заменить на НКА, порождающий тот же язык.*

Доказательство. Неформально говоря, мы хотим во всяком пути $s_1 \dots s_n$ произвести разбиение на несколько блоков вида $\varepsilon^k a$ (для некоторых $k \in \mathbb{N}$ и $a \in \Sigma$) оканчивающееся на дополнительный блок вида ε^k и заменить каждый (не дополнительный) блок на ровно один переход; дополнительный блок можно будет убрать при правильной замене множества принимающих состояний.

Теперь формальное построение. Пусть дан ε-НКА $A = (\Sigma, Q, S, F, \delta)$. Будем строить НКА A'. Σ, Q и S оставим без изменений. Теперь определим новые δ' и F'. δ' определяется по правилу

$$\delta'(q, a) := \bigcup_{p \in E(q)} \delta(p, a),$$

а F' определяется как

$$F' := \{q \in Q \mid E(q) \cap F \neq \varnothing\}.$$

□

Определение 11. Недетерминированный конечный автомат с regex-переходами (также “RE-НКА” или “RE-NFA”) — это совокупность

- входного алфавита Σ,
- конечного множества состояний Q,
- множества начальных состояния $S \subseteq Q$,
- множество принимающих состояний $F \subseteq Q$,
- функция перехода $\delta : Q \times \mathrm{RE}(\Sigma) \to 2^Q$, что для всякого $q \in Q$ и всех $r \in RE(\Sigma)$ кроме, быть может, конечного множества $\delta(q, r) = \varnothing$.

Входными данными является конечная строка w. Входная строка принимается автоматом, если есть её разбиение на подстроки $w = u_1 \dots u_n$, последовательность регулярных выражений r_1, …, r_n и последовательность состояний q_0, …, q_n, что

- $u_k \in L(r_k)$ для всех k,
- $q_0 \in S$,
- $q_{k+1} \in \delta(q_k, r_{k+1})$,
- $q_n \in F$.

Язык автомата A или *язык, распознаваемый автоматом* A, — это множество $L(A)$ всех строк, распознаваемых (принимаемых) автоматом A. Язык, распознаваемый хоть каким-нибудь автоматом называется *регулярным*.

Замечание. Недетерминированные конечные автоматы с ε-переходами удобно изображать точно также как обычные НКА, но на стрелках (ориентированных рёбрах) переходов теперь можно писать не символы из Σ, а регулярные выражения из $\mathrm{RE}(\Sigma)$. Также по аналогии можно описать про существование пути в графе, но проще будет это вывести напрямик из уже данного определения.

Замечание. В этом случае также можно написать рекурсивную формулу δ^* (и через неё определить принимаемые слова). Но в этот раз она будет совсем муторной и бесполезной.

Лемма 11. *Всякий RE-НКА можно заменить на регулярное выражение, порождающее тот же язык.*

Доказательство. Заметим, что регулярное выражение для пустой строки есть $\varnothing^*$. Таким образом WLOG рассматриваемый RE-НКА автомат содержит одно начальное и одно принимающее состояние; так как иначе можно создать новые начальное и принимающее состояния, провести ε-переходы из нового начального в старые начальные состояния и из старых принимающих в новое принимающее и забыть роли старых начальных и принимающих состояний.

Будем уменьшать когда можно количество рёбер, а затем когда можно количество вершин. Рёбра будем склеивать кратные, т.е. если есть две вершины p и q два перехода из p в q по регулярным выражениям r_1 и r_2, то заменим на один переход из p в q по регулярному выражению $r_1|r_2$. Очевидно, что язык автомата не меняется. Таким образом кратных рёбер у нас исчезают.

Теперь пусть имеется состояние q отличное от начального и принимающего. Попытаемся удалить q. Для всяких вершин p_1 и p_2 рассмотрим регулярные выражения r_1, r_2 и r, соответствующие переходам из p_1 в q, из q в r_2 и из q в q соответственно, и заменим эти переходы (они будут удалены с удалением q) на перход из p_1 в p_2 по регулярному выражению $r_1 r * r_2$. Несложно проверить, что это тоже не изменит язык автомата.

Таким образом у нас останутся только два состояния и не более двух переходов между ними. Пусть регулярное выражение перехода от начального состояния к принимающему — r, а от принимающего к начальному — s. Следовательно, язык автомата порождается регулярным выражением $r(sr)^*$. □

2.2 Разные действия над автоматами

Определение 12. Пусть даны ДКА $A = (\Sigma, P, p_0, \eta, E)$ и $B = (\Sigma, Q, q_0, \delta, F)$. *Прямым произведением автоматов* A и B есть автомат $A \times B := C = (\Sigma, P \times Q, (p_0, q_0), \theta, E \times F)$, где

$$\theta((p, q), a) := (\eta(p, a), \delta(q, a)).$$

В таком случае $L(A \times B) = L(A) \cap L(B)$.

Также можно построить автомат $C = (\Sigma, P \times Q, (p_0, q_0), \theta, E \times Q \cup P \times F)$, где

$$\theta((p, q), a) := (\eta(p, a), \delta(q, a)).$$

В таком случае $L(C) = L(A) \cup L(B)$.

Также можно построить автомат $C = (\Sigma, P, p_0, \eta, P \setminus E)$. В таком случае $L(C) = \Sigma^* \setminus L(A)$.

Также можно построить автомат $C = (\Sigma, P^P, f_\varepsilon = \mathrm{Id}, \widehat{\varepsilon}, \{f \in P^P \mid f^2(q_0) \in F\})$, где P^P — множество функций $P \to P$, $f_w : P \to P$ — функция, переводящее всякое состояние p в состояние, которое получается из p прохождением по слову w, а

$$\widehat{\delta}(f, a) := f_a \circ f.$$

В таком случае $L(C) = \sqrt{L(A)} := \{w \in \Sigma^* \mid ww \in L(A)\}$.

Лемма 12 (о накачке, Рабин, Скотт)**.** *Пусть $L \subseteq \Sigma^*$ регулярен. Тогда есть константа $p \geqslant 1$, что для всякого слова w длины хотя бы p существует разбиение $w = xyz$, где $y \neq \varepsilon$ и $|xy| \leqslant p$, что для всех $l \geqslant 0$*

$$xy^l z \in L.$$

Пример 3. Язык $L = \{(ab)^n a^n\}_{n \in \mathbb{N}}$ не регулярен, но удовлетворяет лемме о накачке.

2.3 Минимальные ДКА

Определение 13. Пусть дан ДКА A. *Язык, принимаемый из состояния q* — язык

$$L_A(q) := \{w \in \Sigma^* \mid \delta_A^*(q, w) \in F_A\}.$$

Состояния, из которых принимаются эквивалентные состояния, называются *равносильными.*

Лемма 13. *Пусть в ДКА A для двух различных вершин q и $q' \neq q_0$ верно, что $L_A(q) = L_A(q')$. Рассмотрим автомат A', полученный из автомата A удалением вершины q' и перенаправлением всех переходов, идущих в q', на q.*

1. *$L_{A'}(p) = L_A(p)$ для всякой вершины $p \in Q \setminus \{q'\}$.*
2. *$L(A') = L(A)$.*

Лемма 14. *Пусть дан регулярный язык L и строки $u, v \in \Sigma^*$, что существует строка $w \in \Sigma^*$, что одна из uw и vw лежит в L, а другая — нет. Тогда для всякого ДКА A, реализующего L, верно, что*

$$\delta^*(q_0, u) \neq \delta^*(q_0, v).$$

Лемма 15. *Если ДКА A минимален, то*

- *все состояния достижимы,*
- *из разных состояний принимаются разные языки.*

Теорема 16. *ДКА, реализующий язык L, без недостижимых состояний, где из разных состояний принимаются разные языки, единственен с точностью до изоморфизма (переименовывания состояний).*

Доказательство. Пусть A, B — два таких ДКА, порождающие язык L. Для каждого состояния $q \in Q_A$ выберем какую-нибудь строку w_q, что

$$\delta_A^*(q_{A,0}, w_q) = q.$$

Тогда обозначим

$$r_q := \delta_B^*(q_{B,0}, w_q).$$

Если для каких-то различных p и q верно, что $r_p = r_q$, то для всякой строки $w \in \Sigma^*$ верно, что

$$w_p w \in L \quad \Longleftrightarrow \quad w_q w \in L.$$

Отсюда следует, что $L_A(p) = L_A(q)$ — противоречие с определением A. Значит все r_q попарно различны. Следовательно $|Q_A| \leqslant |Q_B|$; аналогично наоборот. Значит отображение $\varphi : q \mapsto r_q$ задаёт биекцию.

Если какая-то строка $w \in L_A(q)$, то $w_q w \in L(A) = L(B)$, значит $w \in L_B(\varphi(q))$. Таким образом $L_A(q) \subseteq L_B(\varphi(q))$; аналогично наоборот. Таким образом $L_A(q) = L_B(\varphi(q))$.

Таким образом начальные и принимающие состояние автоматов соответствуют по φ, так как состояние q начально тогда и только тогда, когда $L_A(q) = L(A)$, и принимающее тогда и только тогда, когда $\varepsilon \in L_A(q)$.

Возьмём любое состояние $q \in Q_A$ и $a \in \Sigma$. Обозначим $p_A := \delta_A(q, a)$, $p_B := \delta_B(\varphi(q), a)$. Тогда если есть $w \in L_A(p_A) \triangle L_B(p_B)$, то $aw \in L_A(q) \triangle L_B(\varphi(q))$. Но $L_A(q) = L_B(\varphi(q))$, а значит

$$\varphi(\delta_A(q, a)) = \delta_B(\varphi(q), a),$$

т.е. δ_A и δ_B соответствуют друг другу по φ.

Таким образом φ — изоморфизм автоматов A и B. □

Следствие 16.1. *Любой автомат, реализующий L, является минимальным тогда и только тогда, когда не имеет недостижимых состояний, а из разных состояний принимаются разные языки.*

Следствие 16.2. *Из любого автомата можно получить минимальный, убрав недостижимые состояния и склеив эквивалентные.*

Следствие 16.3. *У всех автоматов без недостижимых состояний множество (следовательно, и количество) языков, принимаемых из состояний одинаковое.*

Теорема 17. *Есть алгоритм, минимизирующий ДКА.*

Доказательство. Уберём из автомата все недостижимые состояния. Язык не поменяется.

В начале рассмотрим разбиение всех состояний $\Omega_0 := \{F; Q \setminus F\}$. Определим для всякого разбиения Ω множества Q функцию $s_\Omega : Q \to \Omega$, которая сопоставляет всякому состоянию q, множество разбиения Ω в котором оно находится, и отношение $\sim_\Omega$ на Q по правилу

$$p \sim_\Omega q \quad \Longleftrightarrow \quad s_\Omega(p) = s_\Omega(q) \wedge \forall a \in \Sigma\ s_\Omega(\delta(p, a)) = s_\Omega(\delta(q, a)).$$

Тогда построим последовательность $(\Omega_i)_{i=0}^{\infty}$, где Ω_0 уже определено, а $\Omega_{i+1} = Q/{\sim_{\Omega_i}}$.

Заметим следующее.

- Ω_{i+1} есть подразбиение Ω_i.
- Всякие состояния, из которых принимаются одинаковые языки, лежат в одних и тех же множествах разбиения Σ_0. То же верно и для всякого Ω_i, что можно доказать по индукции по i: действительно, если они не были разделены разбиением Ω_i, то они эквивалентны по отношению $\sim_{\Omega_i}$, а значит не будут разделены разбиением Ω_{i+1}.
- Разбиений Ω конечное множество. Значит с некоторого момента последовательность стабилизируется.

Таким образом в пределе имеется разбиение Ω, для которого верно

$$\Omega = Q/{\sim_\Omega}.$$

Если есть какие-то два состояния p и q, из которых принимаются разные языки, то есть слово $w \in L_A(p) \triangle L_A(q)$. Значит $\delta^*(p, w)$ и $\delta^*(q, w)$ разделены разбиением Ω_0, а значит и Ω. Но если бы p и q не были бы разделены разбиением Ω, то и $\delta^*(p, w)$ и $\delta^*(q, w)$ не были бы. Значит Ω — разбиение по принимаемым из состояний языкам.

Построение последовательности Ω_i реализуется алгоритмически, а вместе с ней и проверка $\Omega_{i+1} = \Omega_i$. Значит можно построить разбиение Ω и склеить вершины по этому разбиению (рассмотреть автомат по модулю разбиения как отношения эквивалентности). Получим автомат, реализующий тот же язык, но без недостижимых состояний, где языки, принимаемые из состояний, попарно различны. Значит данный автомат минимален. □

2.4 Двухсторонние автоматы

Определение 14. *Двухсторонний детерминированный конечный автомат* (также “2ДКА” или “2DFA”) A — это совокупность

- входного алфавита Σ,
- конечного множества состояний Q,
- начального состояния $q_0 \in Q$,
- множество принимающих состояний $F \subseteq Q$,
- функция перехода $\delta : Q \times \Sigma \cup \{\vdash; \dashv\} \to Q \times \{+1; -1\}$.

Выполнены следующие условия:

- $\vdash$ и $\dashv$ не лежат в Σ.
- Для любого $q \in Q$
$$\delta(q, \vdash) = (p, +1) \qquad \text{и} \qquad \delta(q, \dashv) = (r, -1)$$
для некоторых p и r.

Входными данными является конечная строка $w = s_1 \dots s_l$, преобразуемая в ленту длины $l+2$, на которой написано
$$s_0 := \vdash;\ s_1;\ \dots;\ s_l;\ s_{l+1} := \dashv .$$
Состояние вычисления после k шагов — это совокупность некоторого состояния автомата $q_k \in Q$ и числа $i_k \in \{0; \dots; l+1\}$. Начальное состояние вычисления (оно же состояние вычисления после 0 шагов) — совокупность начального состояния автомата q_0 и $i_0 = 0$. Каждым шагом состояние (q_k, i_k) заменяется на (q_{k+1}, i_{k+1}), где
$$(q_{k+1}, i_{k+1} - i_k) := \delta(q_k, s_{i_k}).$$
Т.е. формально вычисление — это последовательность состояний $((q_k, i_k))_{k=0}^{\infty}$, где

- q_0 — начальное состояние автомата,
- $i_0 = 0$,
- $(q_{k+1}, i_{k+1} - i_k) := \delta(q_k, s_{i_k})$.

Входная строка принимается автоматом, если есть такое n, что $q_n \in F$ и $i_n = l + 1$.

Язык автомата A или *язык, распознаваемый автоматом* A, — это множество $L(A)$ всех строк, распознаваемых (принимаемых) автоматом A. Язык, распознаваемый хоть каким-нибудь автоматом называется *регулярным*.

Теорема 18. *Всякий 2DFA можно свести к DFA.*

Доказательство.

Написать.

□

Теорема 19 (Майхилла-Нероуда). *Пусть дан язык L. Введём отношение эквивалентности $\equiv_L$ на Σ^* по правилу*

$$u \equiv_L v \qquad \Longleftrightarrow \qquad \forall w \in \Sigma^* \quad uw \in L \Leftrightarrow vw \in L.$$

Тогда L регулярен тогда и только тогда, когда количество классов эквивалентности $|\Sigma^*/\equiv_L|$ конечно.

Доказательство. Пусть L регулярен. Тогда есть минимальный автомат A, порождающий язык L. Для всякого слова u можно определить состояние $q_u := \delta^*(q_0, u)$. Тогда множество слов $w \in \Sigma^*$, что $uw \in L$ образует алфавит $L_A(q_u)$. Следовательно,

$$u \equiv_L v \qquad \Longleftrightarrow \qquad L_A(q_u) = L_A(q_v) \qquad \Longleftrightarrow \qquad q_u = q_v$$

(последний переход верен в силу минимальности A). Таким образом

$$|\Sigma^*/\equiv_L| = |Q|, \qquad \Sigma^*/\equiv_L = \{\{u \in \Sigma^* \mid \delta^*(q_0, u) = q\}\}_{q \in Q}.$$

Теперь пусть $\Sigma^*/\equiv_L$ конечно. Рассмотрим ДКА

$$A = (\Sigma, \Sigma^*/\equiv_L, [\varepsilon]_{\equiv_L}, F, \delta), \text{ где} \qquad F := \{N \in \Sigma^*/\equiv_L \mid N \cap L \neq \varnothing\}, \qquad \delta([u]_{\equiv_L}, a) := [ua]_{\equiv_L}.$$

Несложно проверить, что δ определено корректно. При этом понятно, что

$$\delta^*([\varepsilon]_{\equiv_L}, u) = [u]_{\equiv_L}.$$

Заметим для всякого $N \in \Sigma^*/\equiv_L$, что если $N \cap L \neq \varnothing$, то есть $v \in N \cap L$, поэтому $v\varepsilon \in L$, а значит для всякого $u \in N$ верно, что $u\varepsilon \in L$, т.е. $N \subseteq L$. Поэтому понятно, что

$$v \in L \qquad \Longleftrightarrow \qquad [v]_{\equiv_L} \cap L \neq \varnothing \qquad \Longleftrightarrow \qquad \delta^*([\varepsilon]_{\equiv_L}, v) \cap L \neq \varnothing \qquad \Longleftrightarrow \qquad \delta^*([\varepsilon]_{\equiv_L}, v) \in F.$$

Это значит, что $L(A) = L$. □

Замечание. Заодно мы также показали, что

$$L = \bigcup_{u \in L} [u]_{\equiv_L}.$$

2.5 Вероятностные конечные автоматы

Определение 15. *Вероятностный конечный автомат* (также “ВКА” или “PFA”) A — это совокупность

- входного алфавита Σ,
- конечного множества состояний Q,
- начального состояния $q_0 \in Q$,
- множество принимающих состояний $F \subseteq Q$,
- функция перехода $\delta : Q \times \Sigma \to [0; 1]^Q$,

что для любых $q \in Q$ и $a \in \Sigma$

$$\sum_{p \in Q} \delta(q, a)(p) = 1.$$

Входными данными является конечная строка $w = s_1 \dots s_n$. Состояние вычисления после k шагов — это некоторое распределение вероятности между состояниями автомата $f_k : Q \to [0; 1]$. Соответственно, начальное состояние вычисления (оно же состояние вычисления после 0 шагов) — распределение $f_0(q) := [q = q_0]$. Каждым шагом состояние f_k заменяется на $f_{k+1} := \sum_{p \in Q} f_k(p)\delta(p, s_{k+1})$. Т.е. формально вычисление — это последовательность распределений вероятности $(f_k)_{k=0}^n$, где

- f_0 — начальное состояние автомата,
- $f_{k+1} := \sum_{p \in Q} f_k(p)\delta(p, s_{k+1})$.

Распределение f_n называется *результатом вычисления*. Также говорят, что автомат принимает строку w, если $\sum_{q \in F} f_n(q) \geqslant \frac{1}{2}$, и отвергает в ином случае.

Язык автомата A или *язык, распознаваемый автоматом* A, — это множество $L(A)$ всех строк, распознаваемых (принимаемых) автоматом A. Язык, распознаваемый хоть каким-нибудь автоматом называется *регулярным*.

Также определяют *условие ограниченной ошибки* — условие, заключающееся в том, что вероятность принятия для любого слова всегда либо $\geqslant \frac{2}{3}$, либо $\leqslant \frac{1}{3}$. Будем рассматривать ВКА, удовлетворяющие этому свойству.

Теорема 20. *Для всякого ВКА есть ДКА, распознающий тот же язык.*

Доказательство. Пусть дан ВКА A. Пронумеруем его состояния: $M := |Q_A|$, $Q = \{q_i\}_{i=1}^M$. Тогда всякое состояние вычисления есть вектор распределения вероятности $(a_i)_{i=1}^M$ (a_i — вероятность попадания в a_i; $\sum_{i=1}^M a_i = 1$). Также для всякого слова $u \in \Sigma^*$ определим вектора $p(u)$ и $q(u)$, где $p_i(u)$ — вероятность попадания в вершину q_i по прохождению по слову u, а $q_i(u)$ — вероятность попадания в принимающее состояния, выходя из q_i и идя по слову u. В частности, $p(u) \cdot q(v)$ — вероятность принятия слова $u \cdot v$.

Лемма 20.1. *Пусть для некоторых слов u и u' верно, что $\|p(u) - p(u')\| := \max_i |p_i(u) - p_i(u')| \leqslant \varepsilon$. Тогда для любого слова v*

$$|p(u)q(v) - p(u')q(v)| \leqslant n\varepsilon.$$

Доказательство.

$$\begin{aligned}
|p(u)q(v) - p(u')q(v)| &= \left|\sum p_i(u)q_i(v) - \sum p_i(u')q_i(v)\right| \\
&= \left|\sum (p_i(u) - p_i(u'))q_i(v)\right| \\
&\leqslant \sum |p_i(u) - p_i(u')|q_i(v) \\
&\leqslant \sum |p_i(u) - p_i(u')| \\
&\leqslant n\|p(u) - p(u')\| \\
&= n\varepsilon
\end{aligned}$$

□

Следствие 20.1. *Если $\|p(u) - p(u')\| < \frac{1}{3n}$, то из распределений $p(u)$ и $p(u')$ принимаются одинаковые языки, т.е. для всякого слова v верно $uv \in L \Leftrightarrow u'v \in L$.*

Доказательство. Как мы показали разница в вероятностях принятия uv и $u'v$ равна

$$|p(u)q(v) - p(u')q(v)| \leqslant n\|p(u) - p(u')\| < \frac{1}{3}.$$

Поскольку A удовлетворяет условию ограниченной ошибки, то вероятности принятия uv и $u'v$ либо обе $\geqslant \frac{2}{3}$, либо обе $\leqslant \frac{1}{3}$, т.е. uv и $u'v$ либо оба принимаются, либо оба не принимаются. Отсюда и выходит требуемое. □

Вспомним, что все возможные вектора $p(u)$ лежат в пространстве $[0;1]^M$ (в реальности можно ещё поставить условие, что сумма координат равна 1, но это не обязательно). Тогда разобьём его по каждой координате на отрезки длины $1/(3n+1)$; получится $(3n+1)^M$ кубиков, на которые распалось всё пространство $[0;1]^M$. Как мы показали, для любых двух векторов $p(u)$ и $p(u')$ из одного кубика $\|p(u) - p(u')\|$, а значит языки принимаемые внутри каждого кубика одинаковы (если в кубике есть хоть один вектор $p(u)$).

Пусть L — язык, распознаваемый автоматом A. Тогда мы имеем, что $|\Sigma^*/\equiv_L|$ конечно, а значит есть ДКА, реализующий тот же язык. □

Определение 16. *Двухсторонний вероятностный конечный автомат* (также “2ВКА” или “2PFA”) A — это совокупность

- входного алфавита Σ,
- конечного множества состояний Q,
- начального состояния $q_0 \in Q$,
- множество принимающих состояний $F \subseteq Q$,
- функция перехода $\delta : Q \times (\Sigma \cup \{\vdash; \dashv\}) \to [0;1]^{Q\times\{+1;-1\}}$.

Выполнены следующие условия:

- $\vdash$ и $\dashv$ не лежат в Σ.
- Для любых $q, p \in Q$

$$\delta(q,\vdash)(p,-1) = 0 \qquad \text{и} \qquad \delta(q,\dashv)(p,+1) = 0.$$

- Для любых $q \in Q$ и $a \in \Sigma \cup \{\vdash; \dashv\}$

$$\sum_{t\in Q\times\{+1;-1\}} \delta(q,a)(t) = 1.$$

Входными данными является конечная строка $w = s_1 \dots s_n$, преобразуемая в ленту длины $l+2$, на которой написано

$$s_0 := \vdash; s_1; \dots; s_l; s_{l+1} := \dashv .$$

Автомат также блуждает по полосе как $2DFA$, но каждый раз делает то или иное действие случайно с заданным распределением вероятностей. И как для PFA мы принимаем строку, если вероятность её принять хотя бы $\frac{1}{2}$, а также будем требовать условие ограниченной ошибки.

Пример 4. 2ВКА распознают больше языков, чем только регулярные. Для примера построим 2ВКА, распознающий $\{a^n b^n\}_{n\in\mathbb{N}}$.

Автомат будет работать следующим образом. Он проверит, что строка имеет вид $a^i b^j$, где $i \equiv j \pmod 3$ (это можно сделать с помощью 2ДКА, а значит и с помощью 2ВКА). Затем автомат будет до бесконечности проходить по всему слову, на каждом символе (отличном от граничных) подбрасывая монетку. Назовём успехом символа a проход, после которого на каком-то символе a выпал орёл, но ни на одном символе b он не выпал; аналогично назовём успех символа b. В течение процесса проходов будем считать успехи символов (на оба символ будет хватать счётчиков до 3). Если в какой-то момент выпали 3 успеха одного из символов и ни один успех другого, то будем отвергать строку; если же в какой-то момент у каждого символа будет успеха, но у каждого символа их меньше трёх, то будем принимать строку.

Теперь покажем, что автомат действительно распознаёт описанный язык. В случае, если строка не подошла шаблону $a^i b^j$, где $i \equiv j \pmod 3$, строка будет отвергнута с вероятностью 1 — правильно. Следовательно, будем рассматривать только строки такого шаблона.

Если $i = j$, то вероятность выпадения успеха a равна вероятности выпадения успеха b. Т.е. вероятность выпадения успеха a при условии выпадения хоть какого-нибудь успеха равна $1/2$. Следовательно, вероятность вероятность выпадения хотя бы трёх каких-нибудь успехов равна 1, а вероятность выпадения 1 успеха a и двух успехов b или наоборот равна $6/8 = 3/4$. Таким образом вероятность принятия равна $3/4 > 2/3$.

Если $i \neq j$, то WLOG $i > j$, т.е. $i - 3 \geqslant j$. Значит вероятность успеха a при условии выпадения какого-нибудь успеха равна

$$\frac{1/2^j(1-1/2^i)}{1/2^j(1-1/2^i)+1/2^i(1-1/2^j)} = \frac{2^i-1}{2^i+2^j-2} \geqslant \frac{2^i-1}{2^i+2^{i-3}-2} = \frac{8\cdot 2^{i-3}-1}{9\cdot 2^{i-3}-2} = \frac{8}{9}+\frac{7/9}{9\cdot 2^{i-3}-2} > \frac{8}{9}.$$

Таким образом вероятность трёх успехов a подряд $> (8/9)^3 > 2/3$. Т.е. вероятность отвержения $> 2/3$.

Итого автомат принимает только описанный язык и удовлетворяет условию ограниченной ошибки.

2.6 Формальные грамматики

Определение 17. *Граматика* G — это совокупность

- Σ — конечный алфавит языка, который мы определяем,
- N — конечный алфавит "нетерминальных символов", которые являются переменными задания грамматики,
- R — конечное множество правил грамматики, т.е. конечное подмножество $N \times (\Sigma \cup N)^*$
- $S \in N$ — "начальный символ", выделенный нетерминальный, с которого начинается построение.

Определим следующие объекты.

- Отношение $\underset{G}{\Rightarrow}$ (выводимость в один шаг) на $(\Sigma \cup N)^*$ задаётся по правилу:

$$x \underset{G}{\Rightarrow} y \qquad \Longleftrightarrow \qquad \exists u, v, p, q \in (\Sigma \cup N)^* : \quad x = upv \wedge (p; q) \in P \wedge y = uqv.$$

(Из x выводится y в один шаг тогда и только тогда, когда x имеет вид upv, y — uqv, а $p \to q$ — правило в грамматике G.)

- Отношение $\overset{*}{\underset{G}{\Rightarrow}}$ (выводимость) — рефлексивно-транзитивное замыкание отношения $\underset{G}{\Rightarrow}$.
- Сентенциальная форма (также “sentential form”) — всякое $w \in (\Sigma \cup N)^*$, что $S \overset{*}{\underset{G}{\Rightarrow}} w$.
- *Предложение, построенное грамматикой G* — всякое $w \in \Sigma^*$, являющееся сентенциальной формой.
- *Язык грамматики G* — множество всех предложений, построенных грамматикой G.

Замечание. Часто правило записывают не в виде $(p; q)$, а $p \to q$. А когда имеется много правил $p \to q_1, \ldots, p \to q_n$, то обычно пишут $p \to q_1 \mid \cdots \mid q_n$.

Замечание. Всякое слово, полученное из некоторого нетерминала удобно изображать в виде дерева (с введённым на нём порядком "слева направо"), где каждая вершина хранит символ, а переход от родителя к детям соответствует правилам грамматики.

Пример 5 (синтаксис арифметических выражение). Неформально, арифметическое выражение можно построить так.

- Переменная x — арифметическое выражение.
- Число 1 — арифметическое выражение.
- Если e и e' — выражения, то $e + e'$ и $e \cdot e'$ — тоже выражения.
- Если e — выражение, то (e) — тоже.

Формально она строится грамматикой $G = (\Sigma, N, R, S)$, где

- $\Sigma := \{x; 1; +; \cdot; “(”; “)”\}$,
- $N := \{S\}$,
- $R := \{(S; x); (S; 1); (S; S + S); (S; S \cdot S); (S; “(S)”)\}$.

То есть, изображая правила проще, имеем

$$S \to x \mid 1 \mid S + S \mid S \cdot S \mid (S).$$

Замечание 2. Изначально множество правил могло являться подмножеством $((\Sigma \cup N)^* \setminus \{\varepsilon\}) \times (\Sigma \cup N)^*$. В в таком случае была введена *иерархия Хомского*:

- **Тип 0. “Рекурсивно перечислимые” грамматики.** Это грамматики без каких-либо ограничений на правила (кроме, как уде говорилось, того, что заменяемое слово в каждом правиле должно быть непусто).
- **Тип 1. “Контекстно-зависимые” грамматики.** Это грамматики, где правила имеют вид $\alpha A \beta \to \alpha \gamma \beta$, где $\alpha, \beta \in (\Sigma \cup N)^*$, $\gamma \in (\Sigma \cup N)(\Sigma \cup N)^*$, $A \in N$.
- **Тип 2. “Контекстно-свободные” грамматики.** Это грамматики, где правила имеют вид $A \to \alpha$, где $\alpha \in (\Sigma \cup N)^*$, $A \in N$.
- **Тип 3. “Регулярные” грамматики.** Это грамматики, где правила имеют вид $A \to a$ или $A \to aB$, где $a \in \Sigma$, $A, B \in N$.

Например, несложно понять, что регулярные грамматики равносильны ДКА. **Мы же будем обсуждать только тип 2!!!**

Определение 18. Пусть дана грамматика G, а A — нетерминал в ней. *Язык, задаваемый нетерминальным символом A* — это язык слов w, что $A \overset{*}{\underset{G}{\Rightarrow}} w$.

2.7 Операции над грамматиками.

Лемма 21.

1. *Если языки L и M задаются грамматиками, то и $L \cup M$ задаётся грамматикой.*
2. *Если языки L и M задаются грамматиками, то и LM задаётся грамматикой.*
3. *Если языки L задаётся грамматикой, то и L^* задаётся грамматикой.*
4. *Всякий регулярный язык задаётся грамматикой.*

Доказательство.

1. Пусть (Σ, N_L, R_L, S_L) и (Σ, N_M, R_M, S_M) задают языки L и M. Рассмотрим грамматику (Σ, N, R, S), где
   - $N := N_L \cup N_M \cup \{S\}$,
   - $R := R_L \cup R_M \cup \{S \to S_L; S \to S_M\}$.

   Понятно, что тогда такая грамматика задаёт $L \cup M$.
2. Пусть (Σ, N_L, R_L, S_L) и (Σ, N_M, R_M, S_M) задают языки L и M. Рассмотрим грамматику (Σ, N, R, S), где
   - $N := N_L \cup N_M \cup \{S\}$,
   - $R := R_L \cup R_M \cup \{S \to S_L S_M\}$.

   Понятно, что тогда такая грамматика задаёт LM.
3. Пусть (Σ, N_L, R_L, S_L) задаёт язык L. Рассмотрим грамматику (Σ, N, R, S), где
   - $N := N_L \cup \{S\}$,
   - $R := R_L \cup \{S \to S_L S; S \to \varepsilon\}$.

   Понятно, что тогда такая грамматика задаёт L^*.
4. Пусть ДКА $(\Sigma, Q, q_0, \delta, F)$ задаёт язык L. Рассмотрим грамматику (Σ, N, R, S), где
   - $N := \{A_q\}_{q \in Q}$,
   - $R := \{A_q \to aA_{\delta(q,a)}\}_{\substack{q \in Q \\ a \in \Sigma}} \cup \{A_q \to \varepsilon\}_{q \in F}$,
   - $S := A_{q_0}$.

   Понятно, что тогда такая грамматика задаёт L.

□

Теорема 22. *Пусть язык L задаётся грамматикой, а M регулярен. Тогда $L \cap M$ задаётся грамматикой.*

Доказательство. Пусть L порождается грамматикой (Σ, N, R, S), а M — автоматом $(\Sigma, Q, q_0, \delta, F)$. Рассмотрим грамматику (Σ, N', R', S'), где

- $N' := \{A_{p,r}\}_{\substack{A \in N \\ p,r \in Q}} \cup \{S'\}$,

•

$$R' := \left\{ A_{p,r} \to u_0 (B_1)_{r_0,p_1} u_1 \dots u_{n-1} (B_n)_{r_{n-1},p_n} u_n \;\middle|\; \begin{array}{c} (A \to u_0 B_1 u_1 \dots u_{n-1} B_n u_n) \in R \\ \wedge\, p_0, r_0, \dots, p_n, r_n \in Q \\ \wedge\, \forall i\ \delta(p_i, u_i) = r_i \\ \wedge\, p_0 = p \wedge r_0 = r \end{array} \right\}$$
$$\cup \{S' \to S_{q_0,q}\}_{q \in F}.$$

$A_{p,r}$ работает как состояние A из старой грамматики, но теперь с условием, что порождаемое слово должно переводить состояние p в состояние r в данном автомате. Таким образом

$$L(A_{p,r}) = \{w \in L(A) \mid \delta(p, w) = r\}.$$

Следовательно,

$$L(S') = \bigcup_{q \in F} L(S_{q_0,q}) = \{w \in L(S) \mid \delta(q_0, w) \in F\} = L(S) \cap M = L \cap M.$$

□

Теорема 23. *Пусть есть язык L, задающийся грамматикой. Тогда язык префиксов L задаётся грамматикой.*

2.8 Лемма о накачке

Лемма 24 (о накачке, aka Бар-Хиллель, Перлес, Шамир, 1961)**.** *Пусть язык L задаётся грамматикой. Тогда есть константа $p \geqslant 1$, что для всякого слова $w \in L$ длины хотя бы p есть разбиение $w = xuyvz$, что $|uyv| \leqslant p$, $uv \neq \varepsilon$, и для всякого $n \geqslant 0$ слово $xu^n yv^n z \in L$.*

Доказательство. Пусть грамматика $G = (\Sigma, N, R, S)$ задаёт L. Пусть m — наибольшая длина подстановочного слова в G, т.е.

$$m := \max_{(A \to \alpha) \in R} |\alpha|.$$

Пусть $p := m^{|N|+1}$. Возьмём любое слово w на $\geqslant p$ символов и рассмотрим минимальное (в смысле числа вершин) дерево разбора этого слова. Заметим, что в нём хотя бы $m^{|N|}$ нелистовых вершин, так как терминальные символы могут порождены только такими вершинами, а каждая вершина порождает не более m символов (любых, а значит и терминальных). Если без листов дерево имеет глубину $\leqslant |N|$, то в нём не более $\frac{m^{|N|}-1}{m-1}$ вершин, т.е. у него глубина хотя бы $|N| + 1$, т.е. есть путь глубины $\geqslant |N| + 1$. Рассмотрим его постфикс длины $|N| + 1$. В нём будет два одинаковых нетерминальных символа, а значит блок с верхнего по нижний можно дублировать у получать правильные деревья разбора. Это и значит, что некоторые $w = xuyvz$, а новые слова — $xu^n yv^n z$. При этом $uv \not\in$, так как иначе дерево разбора без этого куска будет деревом разбора того же слова w, что противоречит с минимальностью изначального дерева. С другой стороны $|uyv| \leqslant p$, так как верхняя из найденных одинаковых вершин находится на уровне $|N| + 1$ снизу или ниже, а значит её поддерево содержит $\leqslant m^{|N|}$ вершин, а значит порождаемое подслово содержит $\leqslant m^{|N|+1} = p$ символов; и этим словом является uyv. □

Пример 6. Язык $\{a^n b^n c^n\}_{n \in \mathbb{Z}}$ не задаётся грамматикой.

Предположим противное. Тогда по лемме о накачке есть некоторая константа p. Возьмём слово $w = a^p b^p c^p$. По лемме $w = xuyvz$, что $|uyv| \leqslant p$, $uv \neq \varepsilon$, а $xu^n yv^n z$ лежит в языке. Но так как $|uyv| \leqslant p$, то uyv точно не захватывает хотя бы один из блоков a^p, b^p и c^p. Но тогда в слове $xu^n yv^n z$ будет не одинаковое количество a-шек, b-шек и c-шек.

Замечание 3. Как следствие, пересечение грамматизируемых языков не обязательно грамматизируемо. Например, если $L := \{a^n b^n c^m \mid n, m \geqslant 0\}$, а $M := \{a^n b^m c^m \mid n, m \geqslant 0\}$, то L и M задаются грамматиками, а $L \cap M = \{a^n b^n c^n\}_{n \geqslant 0}$ не задаётся.

Лемма 25 (Огдан, 1968)**.** *Пусть язык L задаётся грамматикой. Тогда есть константа $p \geqslant 1$, что для всякого слова $w \in L$ длины хотя бы p и всякого $P \subseteq \{1; \ldots; |w|\}$, что $|P| \geqslant p$, есть разбиение $w = xuyvz$, что uyv содержит не более p позиций из P, а uv — хотя бы одну позицию из P, и для всякого $n \geqslant 0$ слово $xu^n yv^n z \in L$.*

Доказательство. Пусть грамматика $G = (\Sigma, N, R, S)$ задаёт L. Пусть m — наибольшая длина подстановочного слова в G, т.е.

$$m := \max_{(A \to \alpha) \in R} |\alpha|.$$

Пусть $p := m^{|N|+1}$. Возьмём любое слово w на $\geqslant p$ символов, выберем любое множество позиций $P \subseteq \{1; \ldots; |w|\}$ мощности $\geqslant p$ и рассмотрим дерево разбора этого слова. Назовём вершину дерева вершиной ветвления, если у хотя бы двух её детей есть символы-потомки с позициями из P. Тогда в дереве можно выделить позиции из P и вершины ветвления и они образуют "поддерево"; назовём его поддеревом для P.

Заметим, что в нём хотя бы $m^{|N|}$ нелистовых вершин, так как терминальные символы могут порождены только такими вершинами, а каждая вершина порождает не более m символов (любых, а значит и терминальных). Если без листов поддерево (для P) имеет глубину $\leqslant |N|$, то в нём не более $\frac{m^{|N|}-1}{m-1}$ вершин, т.е. у него глубина хотя бы $|N|+1$, т.е. есть путь глубины $\geqslant |N|+1$. Рассмотрим его постфикс длины $|N|+1$. В нём будет два одинаковых нетерминальных символа, а значит блок с верхнего по нижний можно дублировать у получать правильные деревья разбора. Это и значит, что некоторые $w = xuyvz$, а новые слова — $xu^n yv^n z$. При этом uv содержит хотя бы одну позицию из P, так как у каждой вершины поддерева есть хотя бы два ребёнка, а значит хоть какая-то позиция из P попала в повторяемый блок. С другой стороны $|uyv|$ содержит $\leqslant p$ позиций из P, так как верхняя из найденных одинаковых вершин находится в поддереве для P на уровне $|N|+1$ снизу или ниже, а значит её поддерево содержит $\leqslant m^{|N|}$ вершин, а значит порождаемое подслово содержит $\leqslant m^{|N|+1} = p$ позиций из P; и этим словом является uyv. □

Пример 7. Язык $L := \{a^l b^m c^n \mid l, m, n \geqslant 1 \wedge l \neq m \neq n \neq l\}$ удовлетворяет лемме о накачке, но не лемме Огдена.

Действительно, пусть $p = 7$. Возьмём любое слово w длины хотя бы p. Т.е. $w = a^{l_a} b^{l_b} c^{l_c}$, l_a, l_b и l_c попарно различны, а $l_a + l_b + l_c \geqslant 7$. Пусть $s \in \{a; b; c\}$ — такой символ, что $l_s = \max(l_a, l_b, l_c)$. Заметим, что $l_s \geqslant 4$, так как иначе $l_a + l_b + l_c \leqslant l_c + (l_c - 1) + (l_c - 2) = 3l_c - 3 \leqslant 6$. Тогда пусть k — наименьшее натруальное число, что $k \notin \{l_s - l_a; l_s - l_b; l_s - l_c\}$. Одно из $l_s - l_a$, $l_s - l_b$ и $l_s - l_c$ является нулём, а остальные положительны и различны. Значит $k \leqslant 3$, и следовательно, $k < l_s$. Тогда возьмём такое разбиение $w = xuyvz$, что $u = s^k$, а $yv = \varepsilon$. Тогда $xu^n yv^n z$ тоже будет лежать в L для всех $n \geqslant 0$.

При этом пусть L задаётся грамматикой. Тогда лемма Огдана даёт нам константу p. Тогда рассмотрим строку $w = a^p b^{p+p!} c^{p+2p!} \in L$ и выделим первые p позиций. Тогда есть разбиение $w = xuyvz$ как в лемме Огдана. При этом если u или v захватывает захватывает разные символы, то тогда при повторениях u и v получаемая строка не будет сортированной. Значит u и v содержат символы только одного вида. При этом, uv содержит хотя бы один символ a по лемме Огдана. Значит u — подслово в блоке a^p, а v — подслове в одном из блоков a^p, $b^{p+p!}$ и $c^{p+2p!}$. Если v находится в блоке $b^{p+p!}$, то в слове $xu^n yv^n z$, где $n = \frac{2p!}{|u|}$, блок a-шек будет иметь ту же длину, что и блок c-шек, что противоречит лемме Огдана; аналогично, если v будет в блоке $c^{p+2p!}$. Таким образом u и v находятся в блоке a^p. Тогда в слове $xu^n yv^n z$, где $n = \frac{p!}{|uv|}$, блоки a-шек и b-шек будут одной длины, что тоже противоречит лемме Огдана.

Теорема 26. *Языки над односимвольным алфавитом, задающиеся грамматикой, регулярны.*

Доказательство. WLOG $\Sigma = \{a\}$. Пусть L задаётся грамматикой. По лемме о накачке есть $p > 0$, что для всякого $k \geqslant p$ если $a^k \in L$, то есть $q \in [1; p]$, что $a^{k-q+qn} \in L$ для всякого $n \geqslant 0$. Следовательно $a^{k+p!n} \in L$ для всех n. Т.е. если какой-то остаток по модулю $p!$ реализуется, то все реализации, начиная с некоторой задаются. Т.е. множество всех возможных k, что $a^k \in L$ периодично с некоторого места, значит L задаётся автоматом. □

Определение 19. *Нормальный вид Хомского* — это грамматика, где каждое правило имеет один из следующих видов.

- $A \to BC$, где $A \in N$, $B, C \in N \setminus \{S\}$.
- $A \to a$, где $A \in N$, $a \in \Sigma$.
- (Иногда добавляют. Влияет только лишь на то, разрешаем ли мы пустой строке присутствовать в языке или нет.) $S \to \varepsilon$.

Теорема 27. *Всякая грамматика приводится к нормальному виду Хомского (не изменяя порождаемого языка). Точнее, пусть имеется грамматика $G = (\Sigma, N, R, S)$. Тогда можно проделать следующие операции (в заданном порядке).*

1. *Можно подправить грамматику, чтобы каждое правило имело вид $A \to \alpha$, где $|\alpha| \leqslant 2$. ("Нарезание правил.")*
2. *Можно подправить грамматику, чтобы не было правил $A \to \varepsilon$ для $A \in N \setminus \{S\}$.*
3. *Можно подправить грамматику, чтобы не было правил $A \to B$ для $A, B \in N$. ("Удаление единичных (цепных) правил.")*
4. *Можно подправить грамматику, чтобы не было правил $A \to bX$ и $A \to Xb$ для $A, X \in N$, $b \in \Sigma$.*
5. *Можно подправить грамматику, чтобы всякое правило имело вид $N \to (\Sigma \cup N \setminus \{S\})^*$.*
6. *Построенная таким образом грамматика будет иметь размер $O(n^2)$, где n — размер изначальной грамматики. Размер грамматики есть $\sum_{(A\to\alpha)\in R} |\alpha| + 2$.*

Доказательство.

1. Просто каждый раз всякое правило $A \to \alpha X$ ($X \in \Sigma \cup N$, $\alpha \in (\Sigma \cup N)^*$), где $|\alpha| \geqslant 2$, заменим на правила $A \to TX$ и $T \to \alpha$, где T — новый нетерминальный символ.
2. Пока будем добиваться данной цели с возможностью потери ε в $L(G)$. Пусть Nullable := $\{A \in N \mid L_G(A) \ni \varepsilon\}$. Заметим, что Nullable строится "за $|N| + 1$ шаг".

   Действительно, пусть Nullable_k — множество нетерминалов $A \in N$, что есть дерево разбора из A глубины не более $k + 2$ (верхний уровень имеет глубину 1) для ε. Тогда $\text{Nullable}_0 = \{A \in N \mid (A \to \varepsilon) \in R\}$, а

   $$\text{Nullable}_{k+1} = \{A \in N \mid \exists (A \to \alpha) \in R\colon\ \alpha \in \text{Nullable}_k^*\},$$

   и

   $$\text{Nullable} = \bigcup_{k=0}^{\infty} \text{Nullable}_k.$$

Тогда Nullable_0 легко строится само по себе, а Nullable_{k+1} легко получается из Nullable_k. При этом $\text{Nullable}_{k+1} \supseteq \text{Nullable}_k$, и если $\text{Nullable}_{k+1} = \text{Nullable}_k$, то для всякого $m \geqslant k$ верно $\text{Nullable}_m = \text{Nullable}_k$. Следовательно

$$\text{Nullable} = \bigcup_{k=0}^{|N|} \text{Nullable}_k.$$

Тогда рассмотрим пересоставим множество правил R, чтобы получить R'. Для всякого правила $A \to \Theta_0 X_1 \Theta_1 \dots X_k \Theta_k$ из R, где $\Theta_i \in \text{Nullable}^*$, $X_i \in \Sigma \cup N$, а $k > 0$, добавим в R' правило $A \to X_1 \dots X_k$. (Фактически, одно правило может перевращено в несколько. Например, если есть правило $A \to \Theta_1\Theta_2$, где $\Theta_1, \Theta_2 \in \text{Nullable}$, то оно будет превращено в правила $A \to \Theta_1\Theta_2$, $A \to \Theta_1$ и $A \to \Theta_2$.) Тогда всякое старое непустое слово порождается новой грамматикой, так как достаточно в дереве разбора отрубить ветви, которые порождают ε (это и есть замена у каждой вершины подстановки $A \to \Theta_0 X_1 \dots x_k \Theta_k$ на $A \to X_1 \dots X_k$, так как ветви каждого Θ_i порождали пустые строки). А всякое новое слово, порождается старой грамматикой, так как в каждой подстановке $A \to X_1 \dots X_k$ у всякой вершины A можно добавить дополнительных детей Θ_i, получая подстановку $A \to \Theta_0 X_1 \dots X_k \Theta_k$ из старых правил, а на каждого ребёнка повесить какое-нибудь дерево, дающее пустую строку. Следовательно, язык новой грамматики — $L(G) \setminus \{\varepsilon\}$.

Чтобы вернуть на место пустую строку, добавим $S \to \varepsilon$.

3. Просто вместо того делать цепочку $A \to B_1 \to \dots \to B_k \to C \to \alpha$ будем теперь сразу писать $A \to \alpha$. Для этого просто добавим в R правила $A \to \alpha$ для всяких A и α, для которых есть $B_1, \dots, B_k \in N$ ($k > 0$), что $(A \to B_1) \in R$, $(B_i \to B_{i+1}) \in R$, $(B_k \to \alpha) \in A$.

4. Для всякого $b \in \Sigma$ заведём символ S_b и правило $S_b \to b$, и если есть правило $A \to bX$, то заменим его на правило $A \to S_b X$; аналогично для $A \to Xb$.

5. Заменим S везде на новый нетерминал S', а также добавим правило $S \to S'$. Тогда S не будет использоваться нигде кроме правила $S \to S'$, а значит не будет в подставляемых частях правил. Правило $S \to \varepsilon$ не трогается (не заменяется на $S' \to \varepsilon$).

6. Нарезание правил линейно изменит размер грамматики. Удаление правил $A \to \varepsilon$ линейно изменит размер грамматики. Удаление цепных правил квадратично изменит размер грамматики. Подправление правил $A \to bX$ и $A \to Xb$ линейно изменит размер грамматики. Удаление S из подставляемых частей увеличит грамматику на константу. Итого грамматика изменится не более чем квадратично.

□

Теорема 28. *Пусть фиксирована грамматика G в нормальном виде Хомского. Тогда для всякой строки w можно за $O(|w|^3)$ времени и $O(|w|^2)$ памяти либо сказать, что $w \notin L(G)$, либо построить дерево разбора w.*

Доказательство. Пусть $w = a_1 \dots a_n$. Определим для всяких i и j, где $0 \leqslant i < j \leqslant n$,

$$T_{i,j} := \{A \in N \mid a_{i+1} \dots a_j \in L_G(A)\}.$$

Тогда заметим, что

$$T_{i-1,i} = \{A \in N \mid (A \to a_i) \in R\}$$

и для $i < j - 1$

$$T_{i,j} = \{A \in N \mid \exists (A \to BC) \in R, k \in \{i+1; \ldots; j-1\}\colon\ B \in T_{i,k} \wedge C \in T_{k,j}\}.$$

При этом такое определение $T_{i,j}$ использует другие $T_{\ldots}$ с меньшим значением $j - i$. Таким образом можно запустить динамическое вычисление, по одному вычисляя уровни $j - i = \text{const}$ от $j - i = 1$ до $j - i = n$.

Вычисление каждого $T_{i,j}$ проводится за $O(j - i - 1)$. При этом всего пар (i, j) примерно n^2. Значит это всё можно проделать за n^3 времени. Память же выделяемая на каждый $T_{i,j}$ равна $O(1)$, поэтому суммарная память имеет размер $O(n^2)$.

Чтобы понять, лежит ли w в L, надо проверить пусто ли $T_{0,n}$. Чтобы построить дерево разбора, нужно для каждого $A \in T_{i,j}$ вместе с ним хранить правило для подстановки, и тогда дерево восстанавливается спуском вниз. □

Лемма 29. *В двоичном дереве с n листьями есть поддерево, где листьев $> \frac{1}{3}n$ и $\leqslant \frac{2}{3}n$.*

Доказательство. Будем спускаться от вершины к ребёнку с большим количеством вершин, пока не попадём в вершину, чьё поддерево содержит $\leqslant \frac{2}{3}n$ листьев. Поскольку каждый спуск уменьшает количество листьев не более чем вдвое, то у полученного поддерева $> \frac{1}{3}n$ листьев. □

Тут должна быть пара страшных алгоритмов, которые мне лень писать. (см.)

Теорема 30 (Льюис, Стинрс, Хартманис)**.** *Пусть фиксирована грамматика G в нормальном виде Хомского. Тогда для всякой строки w можно за $n^{O(\log n)}$ времени и $O((\log n)^2)$ памяти либо сказать, что $w \notin L(G)$, либо построить дерево разбора w.*

Теорема 31 (Руззо, Брент и Гольдшлягер; Риттер)**.** ...

2.9 Неразрешимость задач для грамматик

Определение 20. Пусть дана машина Тьюринга $M = (\Sigma, \Gamma, Q, q_0, \delta, q_{acc})$. Пусть также машина M после i шагов вычисления на w находится в состоянии q_0, на символе a, что слева от неё написана (в хотя бы раз посещённых клетках) строка u, а справа — w. Тогда будем обозначать эту конфигурацию машины M за

$$C_i(M, w) := uqav \in \Gamma^* Q \Gamma \Gamma^*.$$

Также иногда, если машина и входная строка подразумеваются контекстом будем писать будем просто C_i.

Также если M останавливается на w, то будем записывать всю историю вычислений M на w за

$$C_M(w) := C_0 \# C_1 \# \ldots \# C_n \$ C_n^R \# \ldots \# C_0^R \in (\Gamma \cup Q \cup \{\#; \$\})^*.$$

Множество всех историй принимающих вычислений M обозначается за $\text{VALC}(M)$.

Лемма 32. *Для всякой машины Тьюринга M существуют и могут быть эффективно построены такие грамматики G_1 и G_2, что $L(G_1) \cap L(G_2) = \text{VALC}(M)$. Кроме того, существуют грамматики G_1' и G_2', задающие дополнения этих языков: $L(G_i') = \overline{L(G_i)}$ для $i \in \{1; 2\}$ — и потому есть и грамматика, задающая язык $\overline{VALC(M)}$.*

Смущает фраза “эффективно построены” +

Доказательство. Пусть $M = (\Sigma, \Gamma, Q, q_0, \delta, q_{acc}, q_{rej})$.

Хотим построить грамматику G_1, что она будет строить строку вида

$$A_{0,c}\# \dots \# A_{n-1,c}\# B_n \$ A^R_{n-1,n}\# \dots \# A^R_{0,n}\# B^R_0,$$

где $A_{i,c}$, $A_{i,n}$ и B_i являются корректными описаниями конфигураций M (т.е. элементами $\Gamma Q \Gamma\Gamma^*$), и при этом $A_{i,n}$ было бы следующей конфигурацией после $A_{i,c}$, а B_0 — корректной начальной конфигурацией. От G_2 хотим, чтобы он распознавал язык $\{w\$w \mid w \in (\Gamma \cup Q \cup \{\#\})^*\}$. Если эти требования будут выполнены, то $L(G_1) \cap L(G_2)$ будет образован словами вида

$$A_0\# \dots \# A_n \$ A^R_n\# \dots \# A^R_0,$$

где A_{i+1} — конфигурация, следующая за A_i.

Грамматика G_2 задаётся просто:

$$S \to aSa \quad (\forall a \in \Gamma \cup \{\#\}) \qquad\qquad S \to \$.$$

Грамматика G_1 задаётся следующим образом:

$$\begin{aligned}
&S \to Aaq_0 \quad (a \in \Sigma)\\
&A \to Aa \quad (a \in \Sigma)\\
&A \to S_1\#\\
&A \to S_2\$\\
&S \to S_1\#\textvisiblespace q_0\\
&S \to S_2\$\textvisiblespace q_0\\
&S_1 \to B\\
&B \to bBb \quad (b \in \Gamma)\\
&B \to aqbCb'aq' \quad (a, b \in \Gamma, q \in Q, \delta(q, b) = (q', b', -1))\\
&B \to qbcCcq'b' \quad (b, c \in \Gamma, q \in Q, \delta(q, b) = (q', b', +1))\\
&C \to bCb \quad (b \in \Gamma)\\
&C \to \#S_1\#\\
&C \to \#S_2\$\\
&S_1 \to qbCb'\textvisiblespace q' \quad (b \in \Gamma, q \in Q, \delta(q, b) = (q', b', -1))\\
&B \to qb\#S_1\#\textvisiblespace q'b' \quad (b \in \Gamma, q \in Q, \delta(q, b) = (q', b', +1))\\
&B \to qb\#S_2\$\textvisiblespace q'b' \quad (b \in \Gamma, q \in Q, \delta(q, b) = (q', b', +1))\\
&S_2 \to D\\
&D \to bD \quad (b \in \Gamma)\\
&D \to q_{acc}bE \quad (b \in \Gamma)\\
&E \to bE \quad (b \in \Gamma)\\
&E \to \varepsilon
\end{aligned}$$

Давайте разбираться в ней.

Мысленно разделим задачу на 3 части: построение блока B^R_0, построение последовательности пар блоков $A_{i,c}$ и $A^R_{i,n}$ и построение блока B_n. За них будут "отвечать" символы S, S_1 и S_2 соответственно.

Сначала нужно построить блок B^R_0. Заметим, что сам блок B_0 выглядит либо как q_0w, где w — входная строка и $w \neq \varepsilon$ ($w \in \Sigma^*$), либо, если $w = \varepsilon$, $q_0\textvisiblespace$. Первый случай реализуют правила:

$$\begin{aligned}
&S \to Aaq_0 \quad (a \in \Sigma)\\
&A \to Aa \quad (a \in \Sigma).
\end{aligned}$$

После этого мы либо должны написать # и перейти к построению блоков $A_{i,c}$ и $A_{i,n}$, т.е. написать $S_1\#$, либо (при отсутствии блоков $A_{i,c}$ и $A_{i,n}$) сразу перейти к построению блока B_n, т.е. написать $S_2\$$. Это реализуется правилами

$$A \to S_1\#$$
$$A \to S_2\$.$$

При этом во втором случае ($w = \varepsilon$) и построение блока B_0^R и переход к следующему этапу реализуется правилами

$$S \to S_1\#\_q_0$$
$$S \to S_2\$\_q_0.$$

Теперь нужно построить блок ...

Дописать по см.

□

Теорема 33. *Следующие задачи алгоритмически неразрешимы:*

1. *пустота пересечения для двух данных грамматик;*
2. *однозначность данной грамматики (т.е. отсутствие строк с двумя и более различными деревьями разбора);*
3. *определяет ли данная грамматика множество всех строк;*
4. *равносильность двух данных грамматик (определяют ли они один и тот же язык);*
5. *регулярность языка, порождаемого данной грамматикой.*

Доказательство. Вспомним, что проверка принимаемого языка машины Тьюринга на пустоту — алгоритмически неразрешимая задача.

1. Если бы пустота пересечения грамматик была бы алгоритмически разрешимой, то можно было бы построить машину, которая для всякой машины Тьюринга генерировала бы грамматики G_1 и G_2 как в лемме 32 и проверяла бы пустоту $L(G_1) \cap L(G_2)$. Такая машина проверяла бы принимающий язык любой машины Тьюринга на пустоту — противоречие.

2. Заметим, что для однозначных грамматик G_1 и G_2 непустота $L(G_1) \cap L(G_2)$ равносильна однозначности грамматики

   $$(\Sigma, N_1 \cup N_2 \cup \{S\}, R_1 \cup R_2 \cup \{S \to S_1; S \to S_2\}, S),$$

   где $G_1 = (\Sigma, N_1, R_1, S_1)$, $G_2 = (\Sigma, N_2, R_2, S_2)$. Несложно также заметить, что грамматики G_1 и G_2 из леммы 32 однозначны. Значит можно опять построить машину, проверяющую пустоту машины Тьюринга.

3. Аналогичное противоречие можно получить, если проверять $L(\overline{G}_1) \cup L(\overline{G}_2)$ (что задаётся грамматикой) на равенство Σ^*.

4. Аналогичное противоречие можно получить, если проверять равносильность грамматик для $L(\overline{G}_1) \cup L(\overline{G}_2)$ (что задаётся грамматикой) и для Σ^*.

5. ...

   Дописать по см.

□

3 Сложность вычислений

Замечание. Если не оговорено обратное, будем подразумевать, что алфавит $\Sigma = \{0; 1\}$.

Определение 21. *Индивидуальная задача* — пара (условие; решение) $\in \Sigma \times \Sigma$. *Массовая задача* — некоторое множество индивидуальных задач, т.е. отношение на Σ.

Замечание. Говоря по-человечески, массовая задача — формализация задачи, которую мы будем пытаться решать машинами Тьюринга и т.п.

Пример 8. Задача нахождения нетривиального делителя числа:

$$\{(n, d) \mid n \vdots d \wedge 1 \leqslant d \leqslant n\}.$$

Формально, n и d должны записываться строками из Σ^*.

Определение 22. Говорим, что "алгоритм" *решает задачу поиска для массовой задачи* R, если для всякого условия x (т.е. просто строки $x \in \Sigma^*$, называемой "условием") он находит решение w (т.е. строку $w \in \Sigma^*$), удовлетворяющее $(x, w) \in R$.

Определение 23. Всякой массовой задаче R можно сопоставить язык

$$L(R) := \{x \mid \exists w\colon\ (x, w) \in R\}.$$

Т.е. можно строить задачу распознавания (есть ли для данного x решение).

Определение 24. Детерминированная машина Тьюринга (ДМТ) — это совокупность

- входного алфавита Σ — конечного алфавита входных данных,
- символ начала строки $\triangleright \notin \Sigma$,
- символ пробела (нетронутой клетки) $\_ \notin \Sigma \cup \{\triangleright\}$,
- конечного множества внутренних состояний Q,
- начального состояния q_S, принимающего состояния q_Y и конечного состояния q_N,
- количества рабочих лент k
- и функции переходов $\delta : Q \times (\Sigma \cup \{\triangleright; \_\})^{k+1} \to Q \times \{-1; 0; +1\} \times ((\Sigma \cup \{\triangleright; \_\}) \times \{-1; 0; +1\})^{k+1}$.

Это та же МТ, но теперь есть входная лента (read-only), выходная лента (write-only) и k независимых рабочих лент. И все ленты бесконечны, но только вправо. При этом ставтся условия о том, что ДМТ намеренно не выезжает за пределы ленты и не рисует и не стирает символы $\triangleright$.

Входными данными программы является конечная строка $s_1 \dots s_l$, которая превращается в запись $(a_{i,j})_{\substack{i \in \mathbb{N} \\ 0 \leqslant j \leqslant k+1}}$ на лентах по правилу

$$a_{i,j} = \begin{cases} \triangleright & \text{если } i = 0 \\ s_i & \text{если } j = 0 \wedge 1 \leqslant i \leqslant l \\ \_ & \text{иначе.} \end{cases}$$

Состояние каждого момента вычисления — это совокупность

- состояния ленты $(a_{i,j})_{\substack{i\in\mathbb{N}\\0\leqslant j\leqslant k+1}}$, где $a_{i,j} \in \Sigma$, $a_{0,j} = \triangleright$ и все $a_{i,j}$ кроме конечного числа являются пробелами $\_$,
- положений головок $(n_j)_{0\leqslant j\leqslant k+1}$ на лентах и
- состояния машины $q \in Q$.

Состояния начального момента вычислений образуется Из

- состояния ленты, полученного из входных данных,
- положений головок по умолчанию $(n_j)_{0\leqslant j\leqslant k+1} = (0)_{0\leqslant j\leqslant k+1}$,
- начального состояния машины q_S.

Каждым шагом состояние $((a_{i,j}), (n_j), q)$ заменяется на состояние $((a'_{i,j}), (n'_j), q')$, где $a'_{i,j} = a_{i,j}$ для всех (i, j) кроме (n_j, j) для $1 \leqslant j \leqslant k + 1$ и

$$(q', n'_0 - n_0, a'_{n_1,1}, n'_1 - n_1, \ldots, n'_k - n_k, a_{n_{k+1},k+1}, n'_{k+1} - n_{k+1}) = \delta(q, a_0, \ldots, a_k).$$

Определение 25. ДМТ *принимает* входное слово, если она заканчивает работу свою работу в q_Y и *отвергает* – если в q_N. ДМТ M *распознаёт язык* A, если принимает все $x \in A$ и отвергает все $x \notin A$. Пишем $A = L(M)$. ДМТ может также *вычислять функцию* (решать задачу поиска). Значением этой функции над данном входе будем считать содержимое входной ленты после достижения q_Y.

Время работы M на входе x — количество шагов (применений инструкций) до достижения q_Y или q_N. *Используемая память* — суммарное крайнее правое положение всех головок на рабочих лентах.

Определение 26. *Универсальная машина Тьюринга* — такая МТ $M(a, x)$, которая принимает описание a машины M_a и её вход x, и выдаёт тот же результат, что и $M_a(x)$.

Лемма 34. *Вместо любого алфавита можно рассматривать только двусимвольный.*

Надо написать?

Лемма 35. *Двусторонняя лента эмулируется на односторонней с константным замедлением. Т.е. для всякой ДМТ M на k двусторонних лентах есть ДМТ M' на k односторонних лентах, выдающая те же ответы с константным замедлением. При этом $|\sigma(M')| = O(|\sigma(M)|^2)$, а сама M' алгоритмически построима за полиномиальное от $\sigma(M)$ время.*

Теорема 36. *Существует универсальная ДМТ $M(a, x)$, использующая лишь две рабочих ленты и выдающая результат за время $O(f(|a|)t\log(t))$, где t — время работы M_a на x, а f — некоторая функция.*

На лекции не было показано алгоритмического построения автомата, который читает любую машину и выполняет её код. Вместо этого всякой машине сопоставлялась новая на двух лентах.

Доказательство. Давайте попытаемся эмулировать ДМТ нескольких лент на двух лентах. Одна из лент будет вспомогательной. Сначала заметим, что мы можем хранить все k лент на одной, если писать i-ую ленту на клетках с отсатком i по модулю k.

Первая оптимизация времени работы в том, что будем двигать не головки по лентам, а ленты вдоль фиксированной головки. Т.е. будем сдвигать каждую из k компонент нашей ленты в зависимости от результата функции перехода.

Но двигать всю ленту очень долго. Поэтому будем оптимизировать время работы задачи "сдвинуть компоненту на 1". Для этого будем хранить ленту (компоненту) в разреженном состоянии. Для этого будем хранить на ленте последовательность блоков $L_n \dots L_1 H R_1 \dots R_n$, где размер блоков L_i и R_i — 2^i, а блок H состоит из одного элемента, на который смотрит головка. Будем делать так, чтобы в каждый момент в блоках L_i и R_i было либо 0, либо 2^{i-1}, либо 2^i элементов хранимой ленты (всё остальное занимают специальные пробелы), а также для всякого i в блоках L_i и R_i было суммарно ровно 2^i элементов.

Тогда пусть нам нужно сдвинуться влево. Найдём минимальное m, что блок L_m заполнен хотя бы на половину. Если такого блока нет, то поднимем n на один, создав полный блок L_n и пустой блок R_n (в таком случае $m = n$). Тогда половину из блока L_m (это 2^{m-1}) символов перенесём в блоки $L_{m-1}, \dots, L_1$ и H, что L-блоки будут заполнены на половину (т.е. 2^{m-1} будет разделено на части как $2^{(m-1)-1} + 2^{(m-2)-1} + \dots + 2^{(m-(m-1))-1} + 1$). При этом символы из H, $R_1, \dots, R_{m-1}$ будут перенесены в R_m как новая половина (где R_m как раз пуст хотя бы на половину).

Каждая такая операция будет произведена за $O(2^m)$ при этом ближайшие $2^{m-1} - 1$ шагов блоки L_m и R_m тронуты не будут (т.е. блоки L_i и R_i будут тронуты не чаще однго раза в 2^{i-1} ходов). При этом максимальный номер блока не больше $\log_2(t)$. Следовательно, время работы алгоритма будет не более

$$\sum_{i=1}^{\log(t)} C \cdot 2^i \cdot \frac{t}{2^{i-1}} = 2C \cdot t \log(t) = O(t \log(t)).$$

□

Определение 27. *Недетерминированная машина тьюринга (НМТ)* получается из детерминированной таким же образом, как НКА из ДКА. Т.е. δ становится многозначной функцией, и появляется, так называемое, *дерево вычислений*, а мы говорим, что машина принимает вход, если есть вычисление, заканчивающееся принятием (состоянием q_Y).

При этом если есть вычислимая за разумное время (конкретно, за собственное, т.е. двоичная запись $f(n)$ должно вычисляться за $f(n)$) верхняя граница (от n) на время работы какой-нибудь принимающей ветви для всякой входной строки длины n, то мы можем встроить "будильник": измерить длину входа, вычислить и записать верхнюю границу и на каждом ходу уменьшать этот счётчик, чтобы когда он достигнет нуля, насильно отвергнуть вход. Тогда у всякого входы длины n время работы любой ветви вычислений будет длиться не дольше данного времени (формально, не больше $O(f)$, где f — данная функция верхней границы). Действительно, если хранить время в двоичной записи, и всё время пробегаться от разряда единиц до ближайшего ненулевого разряда, уменьшать его на единицу, потом идти назад, заменяя все нули на единицы. То каждый раз придётся делать не более $O(\log_2(f))$ ходов. Но реально, количество ходов равно $\Theta(i)$, где i — номер найденного разряда. Но к нему надо ходить раз в 2^{-i} раз. Следовательно, количество ходов в среднем будет

$$\Theta\left(\sum_{i=0}^{\log_2(f)} 2^{-i}\right) = \Theta\left(\frac{1 - 2^{-\log_2(f)-1}}{1 - 2^{-1}}\right) = \Theta\left(2 - \frac{1}{2f}\right) = \Theta(1).$$

При этом ранее отвергаемые входы будут отвергаться, так как принимающих ветвей не появилось, а ранее принимаемые — приниматься, так как есть принимающее вычисление, попадающее в рамки будильника, и следовательно, которое останется принимающим. Таким образом

можно считать, что время работы машины на любом входе заданной длины ограничено независимо от направления вычисления.

Определение 28 (альтернативное)**.** НМТ — это ДТМ, принимающая в себя два аргумента: x и w, и принимающая строку x тогда и только тогда, когда существует w, что вход (x, w) принимается.

Определение 29. Функция $t : \mathbb{N} \to \mathbb{N}$ называется *конструируемой по времени*, если

- $t(n)$ не убывает,
- $t(n) \geqslant n$,
- двоичная запись $t(|x|)$ вычислима по входу x на ДМТ за $t(|x|)$ шагов.

$\mathrm{DTime}(f)$ — класс языков L, для которых есть ДМТ M, принимающая L за время $O(f)$. $\mathrm{NTime}(f)$ — класс языков L, для которых есть НМТ M, принимающая L за время $O(f)$.

$$\mathrm{P} := \bigcup_{c \geqslant 1} \mathrm{DTime}(n^c), \qquad \mathrm{NP} := \bigcup_{c \geqslant 1} \mathrm{NTime}(n^c)$$

Определение 30. Массовая задача R *полиномиально ограничена*, если существует полином p, ограничивающий длину кратчайшего решения:

$$\forall x (\exists u \colon (x, u) \in R \Rightarrow \exists w \colon ((x, w) \in R \wedge |w| \leqslant p(|x|))).$$

Массовая задача R *полиномиально проверяема*, если существует полином q, ограничивающий время проверки решения: для любой пары (x, w) можно проверить принадлежность $(x, w) \stackrel{?}{\in} R$ за время $q(|(x, w)|)$.

$\widetilde{\mathrm{NP}}$ — класс задач поиска полиномиально ограниченных полиномиально проверяемых.

$\widetilde{\mathrm{P}}$ — класс задач поиска из $\widetilde{\mathrm{NP}}$, разрешимых за полиномиальное время, т.е. задаваемых отношениями R, такими, что $\forall x \in \{0; 1\}^*$ за полиномиальное время можно найти w, для которого $(x, w) \in R$.

Определение 31 (альтернативное)**.** NP — класс языков (задач распознавания), задаваемых полиномиально ограниченными полиномиально проверяемыми массовыми задачами, т.е. $\mathrm{NP} = \{L(R) \mid R \in \widetilde{\mathrm{NP}}\}$. Иначе говоря, $L \in \mathrm{NP}$, если имеется полиномиально ограниченное полиномиально проверяемое R, такая, что $\forall x \in \{0; 1\}^*$ $x \in L \iff \exists w \colon (x, w) \in R$.

Определение 32. *Сведение языков по Карпу ("many-one")*: $L_1 \to L_2$. Если имеется полиномиально вычислимая (на ДМТ) функция $f : \Sigma^* \to \Sigma^*$, что для всякого $x \in \Sigma^*$ верно $x \in L_1 \Leftrightarrow f(x) \in L_2$.

Сведение задач поиска по Левину: $R_1 \to R_2$. Если имеются полиномиально вычислимые функции $f : \Sigma^* \to \Sigma^*$ и $g, h : (\Sigma^*)^2 \to \Sigma^*$, что

- для всяких $x, y \in \Sigma^*$ верно $R_1(x, y) \Rightarrow R_2(f(x), g(x, y))$,
- для всяких $x, y \in \Sigma^*$ верно $R_2(f(x), y) \Rightarrow R_1(x, h(f(x), y))$.

Пусть есть какое-то отображение $f : \Sigma^* \to \Sigma^*$. *Оракульной машиной Тьюринга* $M^{\cdot}$ называется (обычно детерминированная) МТ, где выделены состояния q_{in} и q_{out}, но не определён переход в δ из состояния q_{in}. За M^f обозначается "машина", которая из всех состояний кроме q_{in} считает по правилам $M^{\cdot}$, а при попадании в q_{in} она определённым образом двигает головки,

но на выделенной (третьей) ленте заменяет всё содержимое s на $f(s)$ (и это считается одним ходом).

Сведение чего-угодно по Тьюрингу: $A \to B$. Если имеется оракульная полиномиальная по времени машина $M^{\cdot}$, что M^B решает A. То есть если A — язык, то имеется ввиду $A = L(M^B)$, а если A — массовая задача, то M^B её решает; если B — язык, то под M^B подразумевается M^{χ_B}, а если массоваая задача, то подразумевается M^f, где f решает задачу поиска в B.

Теорема 37.

1. *Классы* P *и* NP *замкнуты относительно сведений по Карпу.*
2. *Классы* $\widetilde{\mathrm{P}}$ *и* $\widetilde{\mathrm{NP}}$ *замкнуты относительно сведений по Левину.*
3. *Классы* P *и* $\widetilde{\mathrm{P}}$ *замкнуты относительно сведений по Тьюрингу.*

Доказательство.

1. Если L_1 сведён по Карпу к L_2, то задачу проверки $x \overset{?}{\in} L_1$ можно за полиномиальное время на ДМТ свести к задаче $f(x) \overset{?}{\in} L_2$. Т.е. из определения следует, что $x \in L_1$ тогда и только тогда, когда $f(x) \in L_2$, а $f(x)$ можно вычислить за полномиальное время. Следовательно, если вторая решается за полиномиальное время на ДМТ (НМТ), то и исходная задача решается за полиномильное время на ДМТ (НМТ).

2. Если R_1 сведена по Левину к R_2, то задачу поиска для x в R_1 можно за полиномиальное время на ДМТ свести к задаче поиска для $f(x)$ в R_2. Т.е. из определения следует, что у x есть решение в R_1 тогда и только тогда, когда у $f(x)$ есть решение в R_2. При этом если мы нашли решение y_2 входа $f(x)$ в R_2, то за полиномиальное время мы можем получить решение $h(f(x), y_2)$ входа x в R_1, а если хотим проверить решение y_1 входа x в R_1, то за полиномиальное время можем свести задачу к проверке решения $g(x, y_1)$ входа $f(x)$ в R_2.

   Следовательно, если R_2 полиномиально ограничено, то и R_1 полиномиально ограничено (так как f на строке полиномиального размера сможет сделать строку только полномиального размера), если R_2 полиномиально проверяема, то и R_1 полиномиально проверяема, а если R_2 полномиально разрешима, то и R_1 полиномиально разрешима.

3. Без нормального доказательства.

□

Определение 33. Пусть дан класс задач C. Тогда задача X называется

- C*-трудной*, если всякая задача $Y \in C$ сводима к X,
- C*-полной*, если она C-трудная и лежит в C.

Теорема 38. *Если задача* A NP*-трудна и лежит в* P*, то* P = NP.

Доказательство. Если есть NP-полная задача B, то для неё

$$B \in \mathrm{P} \Leftrightarrow \mathrm{P} = \mathrm{NP}.$$

Поскольку A лежит в $\mathrm{P} \subseteq \mathrm{NP}$, то она NP-полная, а тогда P = NP. □

3.1 NP-полные задачи

Определение 34 (задача об ограниченной остановке)**.** $\widetilde{\text{BH}}(\langle M, x, 1^t\rangle, w)$ принимает в себя запись НМТ M, входную строку x, t в унарном виде и подсказку w и отвечает, правда ли $M(x, w)$ принимает и делает это за t шагов.

Теорема 39. *Задача об ограниченной остановке* $\widetilde{\text{NP}}$*-полна, а соответствующий ей язык* NP*-полон.*

Доказательство. Покажем $\widetilde{\text{NP}}$-трудность. Пусть имеется $R \in \widetilde{\text{NP}}$ и M^* будет проверять R, а также дано x. Просится найти ω, что $M^*(x, \omega) = 1$. Поскольку M^* полиномиально проверяет пару (x, ω), то есть некоторый полином $p(|x|)$, за время которого этот ввод будет принят. Тогда построим функцию

$$f(x) := \langle M^*, x, 1^{p(|x|)}\rangle.$$

Очевидно, что результат функции f построим за полиномиальное время от $|x|$. Также определим функции

$$g(x, \omega) = h(y, \omega) := \omega,$$

которые тоже вычисляются за полиномиальное время. При этом заметим, что если $(x, \omega) \in R$, то это значит M^* остановится и примет (x, ω) за $p(|x|)$ шагов, а тогда $(\langle M^*, x, 1^{p(|x|)}\rangle, \omega) = (f(x), g(x, \omega))$ является решением задачи $\widetilde{\text{BH}}$. А если $(f(x), \omega) = (\langle M^*, x, 1^{p(|x|)}\rangle, \omega)$ решает задачу $\widetilde{\text{BH}}$, то M^* примет $(x, \omega) = (x, h(f(x), \omega))$ за полиномиальное время. Таким образом мы получили сведение R к *BHtilde* по Левину. Также принадлежность *BHtilde* к $\widetilde{\text{NP}}$ очевидна, так как существует универсальная НМТ.

NP-полнота BH следует из $\widetilde{\text{NP}}$-полноты $\widetilde{\text{BH}}$. □

Тут есть нюанс, который я не понял: бывают решения ω неполиномиальной (более чем полиномиальной) длины (точнее их длина вылезает за рамки $p(|x|)$), но которые принимаются M^* за полномиальное от $|(x, \omega)|$ (от длины обеих строк!) время (это и есть определение полиномиальной проверяемости). Но при этом M^* очевидно не примет этот вход, так как M^* не то что принять не успеет за время $p(|x|)$ эту пару, она не прочтёт её, а больше времени нет, так как задача $\widetilde{\text{BH}}$ обрежет её на этом моменте.

То есть формальная придирка: верность перехода $R(x, \omega) \Rightarrow \widetilde{\text{BH}}(f(x), g(x, \omega))$ пояснена неверно.

Пример: пусть $R = \{(1^x; 1^y) \mid x < y\}$. (Вспомним, что $2^n \geqslant n$.) Тогда понятное дело строку $(1^n; 1^{2^n}) \in R$, $(f(1^n), g(1^n, 1^{2^n})) = (\langle M^*, 1^n, 1^{p(n)}\rangle, 1^{2^n})$ (где M^* — тривиальная машина, принимающая R, а p — любой полином, за который можно хоть какую-нибудь пару принять) не принадлежит $\widetilde{\text{BH}}$, так как M^* не успеет прочитать, а значит, и принять $(1^n, 1^{2^n})$ за полиномиальное время (точнее не будет успевать с некоторого n), а значит это не сведение по Левину.

Определение 35. *Булева схема* C — это ориентированный граф без циклов, где вершины делятся на три типа: “входы”, “операции” и “операции-выходы”, во входные ничего не входит, из выходных – не выходит, а в вершинах-операциях стоят бинарные, унарные или 0-арные операции, и их входная степень совпадает с арностью оператора в них. Тогда *значение* C *на некотором входе* w — значение последовательности выходов, где во входы записали значения из w (по некоторому определённому контекстом соответствию), а в каждой вершине значение определяется рекуррентно как значение оператора в этой вершине, применённого к значениям

из входящих в него по рёбрам вершин (соответствие между вершинами и аргументами оператора тоже является частью структуры булевой схемы).

Также проще булеву схему определять просто как набор из нескольких булевых выражений нескольких (одних и тех же) переменных.

Тогда задача $\widetilde{\text{CURCUIT}}\_\text{SAT}$ — есть множество пар (C, w) булевых схем C с одним выходом и входом, которые выполняют схему (схема на этом выходе даёт 1: $C(w) = 1$).

Теорема 40. $\widetilde{\text{BH}}$ *сводится к* $\widetilde{\text{CURCUIT}}\_\text{SAT}$*, а значит* $\widetilde{\text{CURCUIT}}\_\text{SAT}$ $\widetilde{\text{NP}}$*-полна.*

Доказательство.

□