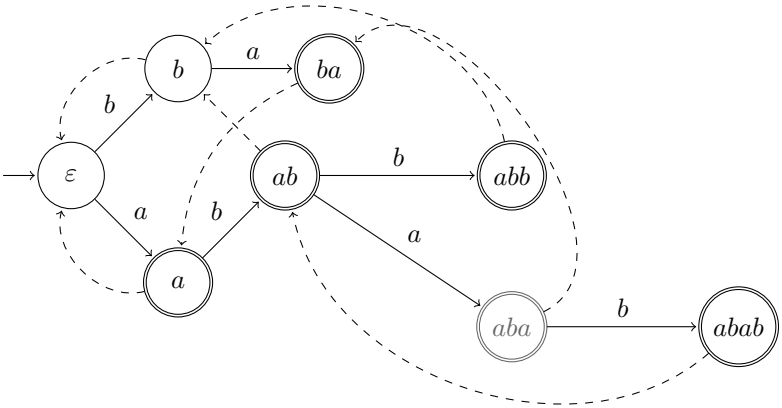
Заметки и задачи о регулярных языках и конечных автоматах

Александр Рубцов

alex@rubtsov.su



Содержание

Предисловие

Эти заметки написаны по материалам моих семинаров по курсу «Теория и реализация языков программирования». В течение последних пяти лет, я давал своим студентам еженедельные домашние задания, которые включали теоретический материал. В этом году я решил переработать еженедельные задания и скомпоновать их в единый текст, дополнив его материалом с семинаров. Сил хватит только на половину курса — на регулярные языки.

Я буду добавлять материал в файл каждую неделю, по мере его прохождения на семинарах.

При написании этого пособия я исходил из следующей идеологии. Полноценно изучать математику невозможно, не умея работать с определениями. Однако, недостаточное освоение этого навыка позволяет студентам успешно сдать математические предметы, требующие вычислений. К таким предметам относится, например, математический анализ, освоение которого без достижения должного уровня навыка работы с определениями кажется невозможным, но жизнь показывает обратное. В случае формальных языков, слово «формальный» далеко не случайно. Естественные и интуитивные гипотезы в этой области могут оказаться ложными, поэтому обоснование словом «очевидно» здесь, пожалуй, более опасно, чем во многих других предметах.

Для того, чтобы помочь студентам лучше работать с определениями, я старался обратить внимание на тонкие моменты при помощи контрольных вопросов (к которым приведены ответы и решения[1]!). Прежде чем читать авторские ответы, нужно потратить силы и попытаться ответить самостоятельно. После чего стоить свериться с автором, дабы убедиться, что вы не попались в ловушку.

Основная цель контрольных вопросов — показать подход к

[1] Но публиковать их я буду только после того как материал будет пройден.

изучению математики, адептом которого является в том числе и автор пособия. Увидев новое определение или формулировку теоремы, нужно не говорить «угу, вроде ясно», а сразу попытаться понять какие объекты попадают под определение, а какие нет. Подумать, какие бедствия произойдут, если в формулировке теоремы изменить некоторые условия или вовсе их вычеркнуть. Контрольные вопросы в этом пособии — часто естественные вопросы, которые возникают у меня при работе с определениями, а порой отражают встреченные за время преподавания мною трудности студентов, часто возникающие не на пустом месте.

Для достижения этой цели необходимо, чтобы всё было хорошо определено. Поэтому, пособие начинается с вводного раздела по теории множеств. Отдельные определения встречаются редко, поэтому определяемое понятие выделяется *курсивом*[2] по ходу текста.

Текст организован следующим образом. В каждом разделе идёт сначала теоретический материал, который снабжён упражнениями и контрольными вопросами по ходу изложения. Их лучше выполнять в процессе чтения. Далее идёт раздел с задачами и дополнительными (но обязательными) контрольными вопросами, после чего приведены ответы к контрольным вопросам и иногда решения избранных задач.

Этот ещё черновая версия текста, поэтому, если вы видите, что в ней что-то не так, пожалуйста, присылайте замечания мне на почту.

Виды задач и обозначения

- Упражнения — вопросы на понимание теории, которые стоит делать для лучшего понимания, но можно и пропускать.
- Контрольные вопросы — вопросы, которые нужны, чтобы

[2] Автор, разумеется, в курсе, что читатель прекрасно знает как выглядит курсив, но не смог отказать себе в странном удовольствии определить курсив, используя курсив, который используется для определений.

проверить, что читатель всё понял правильно и не упустил важных деталей. На эти вопросы нужно постараться ответить самостоятельно, и если не получается, посмотреть ответы в конце раздела.

0 Теоретико-множественное отступление

Теория множеств — входной билет для изучения теории формальных языков и автоматов. Мы предполагаем, что читатель знаком с её азами, однако опыт показывает, что некоторые важные для наших целей вещи ускользают при изучении теории множеств в базовых курсах. Мы приведём здесь короткое введение, за более детальным изучением рекомендуем обратиться к книгам [1] и [2].

Определить понятие множества формально дело непростое и мы не будем этого делать, а лишь опишем основные свойства, которых хватит для наших нужд. Множество — совокупность элементов, в которой игнорируются любые соотношения между ними, и, кроме того, каждый элемент входит в множество не более одного раза. Самый простой способ описать множество — перечислить все его элементы. В этом случае, элементы множества записывают в фигурных скобках:

$$\{1, 2, 3, 4, 5\}.$$

Из сказанного выше следует, что

$$\{1, 2, 3, 4, 5\} = \{5, 4, 3, 2, 1\} = \{1, 2, 3, 2, 4, 5, 5, 5\}.$$

Множество может и вовсе не содержать ни одного элемента. Такое множество называют *пустым* и обозначают $\varnothing$ или $\{\}$.

Явно выписать все элементы можно только у конечных множеств. Количество различных элементов конечного множества A обозначают $|A|$ и называют *мощностью* множества A.

Некоторые бесконечные множества записывают схожим образом: перечисляют начальный отрезок множества до тех пор, пока не станет ясно как продолжить последовательность. Так, множество нечётных чисел не меньше трёх можно записать как

$$B = \{3, 5, 7, \ldots\}.$$

Однако, строгих правил тут нет и при таком подходе нужно быть настороже: кто-то может принять B за множество простых чисел, поэтому, в данном случае, лучше указать в списке ещё и 9.

Другой важный способ описания множеств состоит в описании свойства, которым обладают элементы множества и только они: для этого используют равноправные записи

$$\{x \mid \text{ свойство } x\} \quad \text{и} \quad \{x : \text{ свойство } x\}. \tag{1}$$

Так $C = \{x \mid x = p^2$ для некоторого простого $p > 2\}$ — множество квадратов простых чисел, начиная с 9.

Символ $\in$ обозначает принадлежность элемента множеству, а символ $\notin$ обозначает противное. Так $9 \in B$, $4 \notin C$. Запись $X \subseteq Y$ означает, что каждый элемент множества X принадлежит множеству Y: $\forall x \in X : x \in Y$. Говорят, что X — *подмножество* Y. Если X — подмножество Y и не совпадает с ним, то X называют *собственным подмножеством* (множества Y) и, чтобы это подчеркнуть, используют обозначение $X \subsetneq Y$. Обозначение $\not\subseteq$ аналогично обозначению $\notin$.

Упражнение 1. Убедитесь, что $C \subseteq B$, $B \not\subseteq C$, $C \subsetneq B$, $B \subseteq B$.

Множества A и B равны, если они состоят из одинаковых элементов: $x \in A \iff x \in B$. Из определения подмножества вытекает, что множества A и B равны тогда и только тогда, когда $A \subseteq B$ и $B \subseteq A$.

При решении задач по формальным языкам, часто приходится доказывать равенство двух множеств. Помните, что доказать это можно только доказав оба включения: даже если вы действуете по определению, то нужно доказать, что $x \in A \Rightarrow x \in B$ и $x \in B \Rightarrow x \in A$. Бывают случаи, когда доказательства включений в обе стороны плавно вытекают из оригинального решения, но эти случаи редки.

Определим теперь операции с множествами:

- *объединение* $A \cup B$ множеств A и B состоит из элементов, которые принадлежат хотя бы одному из множеств;

- *пересечение* $A \cap B$ состоит из элементов, которые принадлежат обоим множествам;
- *разность* $A \setminus B$ состоит из элементов, которые принадлежат множеству A, но не множеству принадлежат B;
- *симметрическая разность* $A \triangle B$ состоит из элементов, принадлежащих ровно одному из множеств.

Изобразим эти операции при помощи кругов.[3]

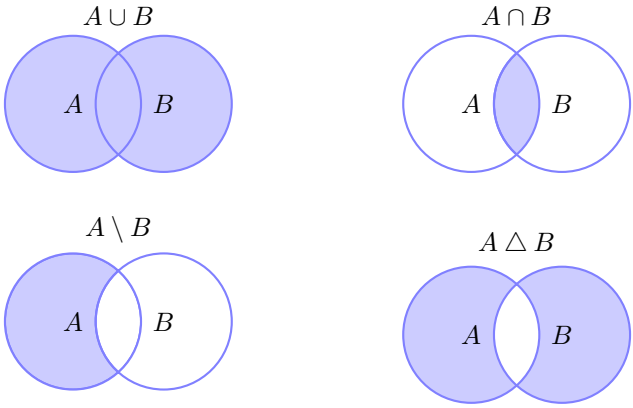


Рис. 1. Диаграммы Эйлера-Венна.

Говорят, что множества A и B *пересекаются*, если $A \cap B \neq \varnothing$.

Замечание 1. *Пустое множество не содержит элементов, поэтому оно не пересекается с собой, хотя и совпадает с собой. Вообще, с пустыми множествами много странностей. Например, считают, что* $\inf \varnothing = +\infty$.

[3] Рисунок с сайта TEXample.net (пример T. Tantau, U. Ziegenhagen).

Упражнение 2. Докажите справедливость формул:

а) $A \triangle B = (A \setminus B) \cup (B \setminus A)$; **б)** $A \triangle B = (A \cup B) \setminus (B \cap A)$.

Также при работе с множествами удобно использовать операцию дополнения. Однако, чтобы ею воспользоваться, нужно сначала договориться о том, какие элементы мы рассматриваем. Что есть дополнение к множеству чётных чисел? Если мы рассматриваем только целые числа ($\mathbb{Z}$), то это множество нечётных чисел, если же мы рассматриваем натуральные числа ($\mathbb{N}$), то это только положительные нечётные числа.

Итак, чтобы ввести операцию дополнения, требуется определить *юнивёрсум* — множество U, элементы которого мы изучаем. Тогда всякое множество A — подмножество множества U, а *дополнение* к A — это множество $U \setminus A$, которое обозначают $\bar{A}$.

Упражнение 3. Проверьте, что $A \cup \bar{A} = U$, $\overline{(\bar{A})} = A$, $A \cap \bar{A} = \varnothing$.

Первое соотношение из упражнения и объясняет смысл названия операции: множество $\bar{A}$ дополняет множество A до полного множества — юнивёрсума U.

Мы используем «наивную теории множеств». Её нужно использовать осторожно, иначе можно прийти к парадоксу Рассела, с которым мы предлагаем познакомиться любознательному читателю, например, в Википедии или в упомянутых выше книжках. Мы с подобными проблемами не столкнёмся, поскольку используемые нами юнивёрсумы достаточно простые.

Через 2^A обозначают множество всех подмножеств множества A.

Упражнение 4. Докажите, что у n-элементного множества ровно 2^n подмножеств.

Напомним читателю, что для каждого A множества A и 2^A неравномощны. В общем случае, множества *равномощны*, если между ними есть биекция (взаимно однозначное соответствие). Так, множество $2^{\mathbb{N}}$ имеет мощность *континуум* (равномощно

множеству действительных чисел $\mathbb{R}$), а множество натуральных чисел $\mathbb{N}$ *счётно* (как и все множества, равномощные $\mathbb{N}$, — например, $\mathbb{Z}$).

Среди математиков нет единого соглашения, считать ли ноль натуральным числом. Автор относится к тому лагерю, у которого натуральность нуля не вызывает сомнений. Но поскольку большинство студентов со школьной скамьи было приучено к обратному, то мы будем обозначать через $\mathbb{N}_0$ множество $\mathbb{N} \cup \{0\}$, где $\mathbb{N} = \{1, 2, 3, \ldots\}$.

Определим теперь последнюю, но не по значимости, операцию над множествами в этом разделе. *Декартовым произведением* двух множеств X и Y называют множество

$$X \times Y = \{(x, y) \mid x \in X,\ y \in Y\}.$$

Так, $\{1, 2, 3\} \times \{a, b\} = \{(1, a), (2, a), (3, a), (1, b), (2, b), (3, b)\}$.

Контрольный вопрос 1. 1. $(b, 1) \stackrel{?}{\in} \{1, 2, 3\} \times \{a, b\}$.

2. Пусть A и B — конечные множества. Найдите $|A \times B|$.

3. Опишите множество $\mathbb{N} \times \varnothing$.

Мы будем считать для удобства операцию декартова произведения ассоциативной:

$$X \times (Y \times Z) = (X \times Y) \times Z = \{(x, y, z) \mid x \in X, y \in Y, x \in Z\} =$$
$$= X \times Y \times Z$$

Определим теперь *декартову степень* множества X:

$$X^k = \underbrace{X \times X \times \ldots \times X}_{k}.$$

Читатель уже знаком с декартовой степенью по множествам $\mathbb{R}^n$ из курса анализа; декартова плоскость $\mathbb{R}^2$ является частным случаем декартова произведения, что объясняет схожесть названий.

В этом разделе не будет задач. Задачи и упражнения по теории множеств будут возникать дальше естественным образом при изучении формальных языков.

1 Слова, языки и операции над ними

Формальные языки возникли при попытке формализации естественных языков. Хотя нужды лингвистов до сих пор не удовлетворены, формальные языки нашли приложения в алгоритмах на строках (таких как поиск вхождения образца в текст), в разработке компиляторов и даже в биоинформатике.

Также формальные языки изучают и в чистой математике. С точки зрения алгебры, слова и операции над ними — достаточно простая алгебраическая структура. Если же читатель не любит алгебру, хотя бы в той же степени, как и автор, он может считать, что слова здесь — просто слова.

1.1 Слова

Под алфавитом понимают конечное множество. Мы будем обозначать алфавиты греческими буквами Σ, Δ, Γ. Чаще всего мы будем использовать алфавиты $\{a\}, \{a, b\}, \{a, b, c\}, \{0, 1\}$. Элементы алфавита называют символами или буквами. На протяжении этого раздела мы будем работать с алфавитом $\Sigma = \{a, b\}$, если не оговорено противного.

Чтобы определить слово, нужно сначала зафиксировать алфавит. *Слово* над алфавитом Σ — конечная последовательность элементов Σ. При описании слова, мы записываем элементы последовательности без разделителя: $w = abaab$; слова мы обозначаем буквами $u, v, w, \ldots$

Через $w[i]$ обозначим i-ю букву слова w, а $w[i, j], i \leqslant j$ обозначает последовательность $w[i]w[i+1]\ldots w[j]$. Для w из предыдущего абзаца $w[1] = a, w[2, 4] = baa$. *Длина слова* w— это длина последовательности символов w; её обозначают $|w|$.

Конкатенацией слов $u = u[1]u[2]\ldots u[n]$ и $v = v[1]v[2]\ldots v[m]$ назовём слово w, получаемое сцепкой последовательностей: $w = u[1]u[2]\ldots u[n]v[1]v[2]\ldots v[m]$. Операцию обозначают точкой или вовсе опускают как и в случае умножения: пишут $w = u \cdot v$ или

$w = uv$. Взяв $u = aba$, $v = ab$, получаем, что $uv = aba \cdot ab = abaab$. Также, подобно умножению, пишут u^3 вместо uuu.

Очевидно, что любое слово можно представить как конкатенацию букв, поэтому запись $w = abaab$ можно понимать как $w = a \cdot b \cdot a \cdot a \cdot b$. Вместо обозначения i-й буквы $w[i]$ часто удобно использовать обозначение w_i, но оно также удобно и для конкатенации последовательности слов, поэтому мы будем писать $w = w_1 \dots w_n, w_i \in \Sigma$ для однозначной трактовки.

Среди всех слов выделяют *пустое слово* ε — последовательности длины ноль: $|\varepsilon| = 0$. Из определения конкатенации следует, что для любого слова w справедливо

$$\varepsilon \cdot w = w \cdot \varepsilon = w.$$

Слово u называют *подсловом* слова w, если существуют такие слова x и z, что $w = xuz$. Через $|w|_a$ обозначают количество букв a в слово w, а через $|w|_u$ — количество различных вхождений подслова u в слово w. Вхождения подслова u в слово w считаются различными, если $w = xuz = x'uz'$ и $x \neq x'$. Отсюда следует, что вхождения одинаковых подслов $w[i, j]$ и $w[k, l]$ различны при $i \neq k$.

Контрольный вопрос 2.

1. Сколько различных подслов у слова $v = ababa$?

2. Чему равно **а)** $|v|$; **б)** $|v|_a$; **в)** $|v|_{aba}$; **г*)** $|v|_\varepsilon$?

3. Верно ли, что любое подслово слова w можно представить в виде $w[i, j]$?

1.2 Языки

Языком L называют множество слов над выбранным алфавитом Σ. Обозначим *множество всех слов* над алфавитом Σ через Σ^*; смысл этого обозначения станет ясен позже. Таким образом,

$L \subseteq \Sigma^*$ и множество всех языков над алфавитом Σ можно записать как $\{L \mid L \subseteq \Sigma^*\}$. Множество, которое в свою очередь состоит из множеств, называют *классом*.

Определим для языков следующие операции.

- Конкатенация: $X \cdot Y = \{x \cdot y \mid x \in X,\ y \in Y\}$.
- Возведение в степень: $X^n = \underbrace{X \cdot X \cdot \ldots \cdot X}_{n}$
- Объединение: $X \mid Y = X + Y = X \cup Y$.
- Итерация:[4] $X^* = \varepsilon + X + X^2 + X^3 + \ldots X^n + \ldots$
- Плюс Клини: $X^+ = X + X^2 + X^3 + \ldots X^n + \ldots = XX^*$.

Контрольный вопрос 3. Опишите следующие языки:

а) $\{\varepsilon, a, ab\} \cdot \{\varepsilon, a, b\}$; **б)** Σ^n; **в)** $X \cdot \varepsilon$; **г)** $X \cdot \varnothing$.

1.3 Регулярные языки

Мы начинаем изучение формальных языков с самого простого и в то же время базового класса — класса регулярных языков, который обозначают REG. Регулярные языки определяются индуктивно:

- $\varnothing \in \mathsf{REG}$;
- $\forall \sigma \in \Sigma :\ \{\sigma\} \in \mathsf{REG}$;
- $\forall X, Y \in \mathsf{REG}\ :\ X \cdot Y,\ X \mid Y,\ X^* \in \mathsf{REG}$.

[4] Также носит название звезда Клини в честь выдающегося математика Стивена Клини.

То есть, любой регулярный язык можно получить по данной схеме (используя указанные операции) и других регулярных языков нет.

Контрольный вопрос 4. Докажите, что следующие языки являются регулярными: **а)** язык, состоящий из одного слова;
б) любой конечный язык; **в)** язык всех слов;
г) язык, состоящий только из пустого слова.

Регулярные выражения — это формула, которая описывает язык, используя указанные выше операции. Как и в случае обычных математических формул, скобки, входящие в регулярное выражение, являются разделителем и задают приоритет операций, но дополнительной смысловой нагрузки не несут: регулярное выражение (a) задаёт язык $\{a\}$, $(a \mid b) = \{a, b\}$, $(a \mid b)^* = \{a, b\}^* = \Sigma^*$.

Для упрощения обозначений, мы не будем различать слово и язык, состоящий из одного слова: например, запись aL следует интерпретировать как $\{a\} \cdot L$, а равенство $w = \{u\}$ как $w = u$.

1.4 Не все языки регулярные

При изучении математики стоит выработать следующую гигиеническую привычку. Видя определение проверять, что вы понимаете, почему есть объекты, удовлетворяющие определению, и есть объекты, определению не удовлетворяющие. Чаще всего удаётся привести как подходящих, так и не подходящих под определение объектов. В случае индуктивных определений, таких как определение регулярных языков, это сделать не всегда просто. Поэтому, мы пока не будем приводить пример нерегулярного языка, однако объясним, почему таковые существуют.

Это легко объяснить использую мощностной метод. Чтобы превратить объяснение в доказательство, читателю нужно выполнить следующее упражнение (или схалтурить и изучить матчасть [2]).

Упражнение 5. Докажите следующие утверждения.

1. Счётное объединение конечных множеств конечно или счётно.

2. Счётное объединение счётных множеств счётно.

3. Множество всех слов Σ^* счётно.

4. Множество R_k регулярных языков, описываемых регулярными выражениями с ровно k операциями, конечно для любого k.

5. Множество всех подмножеств 2^{Σ^*} имеет мощность континуум.

Заметим, что множество регулярных языков счётно: $\mathsf{REG} = \bigcup_{k=1}^{\infty} R_k$, а счётное объединение конечных множеств не более чем счётно, в то время как множество всех языков 2^{Σ^*} имеет мощность континуум. Значит, множества $\mathsf{REG} \subsetneq 2^{\Sigma^*}$, откуда получаем, что есть и нерегулярные языки.

1.5 Контрольные вопросы и задачи

Во всех задачах данного раздела языки определены над алфавитом $\Sigma = \{a, b\}$. РВ — сокращение для регулярного выражения.

1. Вычислить: **а°)** $\varnothing \cdot a(a \mid b)^*$; **б°)** $\{a, a^3, a^5 \ldots\} \cdot \{a, a^3, a^5 \ldots\}$;

в) $\{\varepsilon, a^2, a^4, \ldots\} \cdot \{a, a^3, a^5 \ldots\}$.

2°. Верно ли, что **а)** $\varepsilon \in \{a, aab, aba\}$? **б)** $\varnothing \in \{a, aab, aba\}$?

3. Построить регулярное выражение (РВ) для

а°) языка из слов, содержащих в качестве подслова слово aab;

б) языка, слова которого не содержат подслово ab;

в) языка из слов, содержащих в качестве подслова ровно одно слово ab;

г) языка всех слов чётной длины[5];

д°) языка всех слов с чётным числом букв a;

е) языка, который содержит все слова, в которых есть как буква a, так и буква b;

ж) $\Sigma^* \setminus ((a \mid b)^* bb (a \mid b)^*)$.

Замечание 2. *В этой задаче необходимо доказать, что построенное РВ порождает требуемый язык. Доказательство корректности является важной частью решения, это относится и ко всем последующим задачам!*

Контрольный вопрос 5. Решив контрольный вопрос 4, вы доказали, что каждое слово по отдельности — регулярный язык. Возьмём произвольный язык $L \subseteq \Sigma^*$ и представим его в виде объединения всех входящих в него слов:

$$L = \bigcup_{w \in L} \{w\}.$$

Глядя на эту формулу, некоторые студенты заключают, что раз регулярные языки замкнуты относительно объединения, то L — регулярный язык. Значит, нерегулярных языков не бывает в силу произвольности L! В чём ошибка в этом рассуждении?

4. Постройте регулярное выражение для

а) десятичных записей натуральных чисел (алфавит $\{0, \ldots, 9\}$);

б) языка, состоящего из допустимых имён переменных в языке C или C++;

в) языка, состоящего из всех допустимых email-адресов.

[5] При виде этой задачи, студенты систематически спрашивают, является ли ноль чётным числом. Причины сих статистически значимых сомнений остаются загадкой (да, является!).

В последних двух пунктах самостоятельно выберите подходящий алфавит. При записи используйте сокращения, а ещё лучше выберите удобный для вас современный язык программирования, в который входят регулярные выражения, запишите на них требуемые РВ и поэкспериментируйте с ними.

1.6 Ответы на контрольные вопросы

К.В. 1. 1. **Ответ:** *Нет.* В декартовом произведении важен порядок: не смотря на то, что пара $(1, b)$ принадлежит множеству, пара $(b, 1)$ — нет.

2. **Ответ:** $\varnothing$. По определению $\mathbb{N} \times \varnothing = \{(x, y) \mid x \in \mathbb{N}, y \in \varnothing\}$, но пустое множество не содержит элементов, поэтому в это множество не входит ни одна пара (x, y), а значит это множество пусто.

□

К.В. 2 1. **Ответ:** *10.* Выпишем все подслова v длины от 0 до $|v| = 5$: $\varepsilon, a, b, ab, ba, aba, bab, abab, baba, ababa$.

2. **а)** $|v| = 5$; **б)** $|v|_a = 3$; **в)** $|v|_{aba} = 2$; **г*)** $|v|_\varepsilon = 6$.

3. **Ответ:** *Нет.* Пустое слово так представить не получится. □

К.В. 3. **а)** $\{\varepsilon, a, ab\} \cdot \{\varepsilon, a, b\} = \{\varepsilon, a, ab, aa, aba, b, abb\}$

б) Язык Σ^n состоит из всех слов длины n;

в) $X \cdot \varepsilon = X$; **г)** $X \cdot \varnothing = \varnothing$.

□

К.В. 4. **а)** Любая буква — регулярный язык, регулярные языки замкнуты относительно операции конкатенации, поэтому они замкнуты относительно конечного числа применения конкатенации; любое слово — конечная конкатенация букв.

б) Регулярные языки замкнуты относительно операции объединения, а значит они замкнуты и относительно конечного числа объединений. Любой конечный язык — конечное объединение слов, а каждое слово — регулярный язык.

в) Как было показано в предыдущем контрольном вопросе, язык Σ^n состоит из всех слов длины n. Для каждого $n \geqslant 1$ справедливо $\Sigma^n \subseteq \Sigma^*$, отсюда следует, что Σ^* содержит все слова ненулевой

длины, но $\varepsilon \in \Sigma^*$ по определению. Значит, Σ^* с одной стороны — язык всех слов, а с другой стороны, регулярный язык по построению.

г) Предлагаем читателю убедиться, что $\varepsilon = \varnothing^*$. □

К.В. 5. Замкнутость класса относительно операции объединения влечёт замкнутость только относительно конечного числа объединений. При этом, класс может быть не замкнут относительно счётного объединения, что и справедливо для класса регулярных языком. Обратим внимание, что первое утверждение решения справедливо для любой операции, а не только для операции объединения. □

2 Конечные автоматы: введение

Формальные языки нельзя рассматривать в отрыве от автоматов — абстрактных вычислительных устройств. В первом разделе мы привели теоретико-множественное определение класса регулярных языков, но его также можно определять и как языки, распознаваемые конечными автоматами: устройство обрабатывает слово и по результату сообщает, принято ли оно или нет. Все принятые слова и образуют язык, распознаваемый автоматом.

Регулярные выражения позволяют задать язык естественным образом — решив задачу 4, вы убедились, что описать классы допустимых имён переменных или email-адресов нетрудно и достаточно удобно. Но как проверить, что слово, отправленное пользователем через форму, является email-адресом, используя построенное регулярное выражение? Один из способов такой проверки — построить по регулярному выражению эквивалентный конечный автомат и проверить, принимает ли он отправленное пользователем слово.

Начнём с неформального описания конечного автомата и сосредоточимся на частном случае — детерминированном конечном автомате. Это устройство с конечной памятью и доступной только для чтения лентой, на которой в начале работы записано слово, поданное на вход. Автомат последовательно считывает символы с ленты при помощи головки, которая может двигаться только вправо. Всевозможные конфигурации памяти образуют состояния автомата — конечное множество. В начале работы, автомат находится в начальном состоянии q_0.

Автомат проиллюстрирован на рисунке 2. В последней клетке ленты записан технической символ $\lhd$, который обозначает конец входного слова.

За такт работы автомат считывает символ со входной ленты, после чего меняет состояние согласно фиксированным правилам — правилам перехода. Они имеют вид $(q, \sigma) \to q'$: находясь

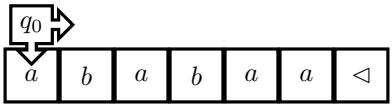


Рис. 2. Конечный автомат

в состоянии q и считав символ σ, автомат сдвигает головку вправо и меняет состояние на q'. Если же переход по паре (q, σ) не определён, то автомат прекращает работу и отвергает слово.

Все состояния автомата разделены на две группы: принимающие и непринимающие. Если после обработки слова автомат оказался в принимающем состоянии, то слово принято.

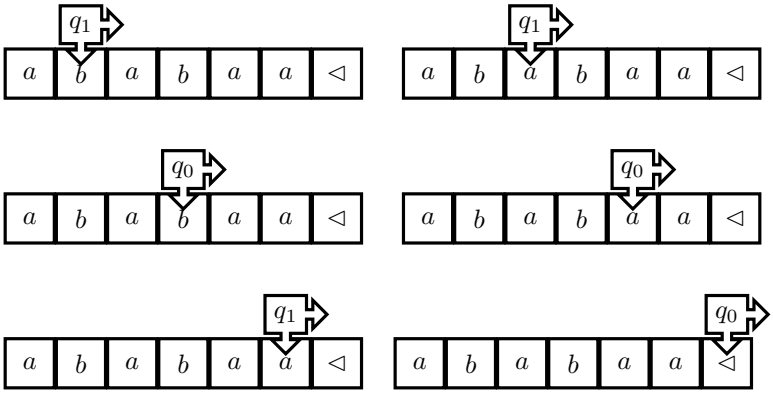
Контрольный вопрос 6. Опишите автомат, который распознаёт слова с чётным числом букв a.

Предлагаем читателю попробовать ответить на контрольный вопрос, прежде чем прочитать решение в следующем абзаце. Можете считать, что у вас есть язык Си и вам нужно написать программу, которая использует только конечную память, хотя входное слово может быть и неограниченно длинным: программа считывает слово посимвольно.

Решение. Достаточно использовать всего два состояния. Автомат находится в состоянии q_0, если в клетках левее от головки чётное число букв a, и в q_1, если нечётное. Переходы описать нетрудно: автомат меняет состояние при обработке буквы a и не меняет при обработке буквы b.

Мы не будем приводить здесь доказательство корректности, хотя и будем требовать его от читателя при решении даже таких простых задач.

Шаги работы автомата на слове $ababaa$ приведены на следующем рисунке.



□

Контрольный вопрос 7. Принимает ли описанный в решении автомат пустое слово? Принадлежит ли оно языку из слов с чётным числом букв a? Если пока сомневаетесь, изучите формальное определение конечного автомата, прежде чем отвечать на вопрос.

2.1 Формальное определение

Определение 1. Конечный автомат $\mathcal{A}$ – это устройство, описываемое набором $(Q, \Sigma, q_0, \delta, F)$, где

- Q – конечное множество состояний автомата;
- Σ – алфавит, слова над которым обрабатывает автомат;
- q_0 – начальное состояние автомата;

- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ – функция переходов;
- $F \subset Q$ – множество принимающих состояний.

Замечание 3. *Мы считаем, что если $\delta(q, \sigma) = \varnothing$, то переход из состояния q по символу $\sigma \in \Sigma \cup \{\varepsilon\}$ не определён.*

Автомат является *детерминированным* (ДКА), если функция переходов определена только на некотором подмножестве $Q \times \Sigma$ и определена однозначно, то есть для каждого состояния q и для каждого символа σ, существует не больше одного состояния $q' \in \delta(q, \sigma)$. В общем случае, автомат является *недетерминированным* (НКА).

На вход автомата подаётся слово $w = w_1 \dots w_n$. Автомат обрабатывает слово слева направо по тактам.

В детерминированном случае, за такт работы автомат находится в состоянии q, считывает символ σ и вычисляет функцию перехода $\delta(q, \sigma)$. Если $\delta(q, \sigma) = q'$, то автомат переходит в состояние q', если же $\delta(q, \sigma) = \varnothing$, то автомат прекращает работу.

В недетерминированном случае, за такт работы, автомат *недетерминированно выбирает состояние* q' из множества $\delta(q, \sigma)$ и переходит в состоянии q'. В случае, если $\delta(q, \sigma) = \varnothing$, автомат прекращает работу. Под недетерминированном выбором, мы понимаем, следующую ситуацию.

Слово *принимается автоматом*, если после обработки слова, автомат оказался в принимающем состоянии. Недетерминированный автомат *всегда* оказывается в принимающем, состоянии, если он может в него попасть. Таким образом, детерминированный автомат принимает слово, если после обработки слова он оказался в принимающем состоянии, а недетерминированный автомат принимает слово, если существует такая последовательность выборов состояний, что после обработки слова, он оказывается в принимающем состоянии.

В этом разделе мы будем работать только с детерминированными конечными автоматами, однако формальное определение нам удобно дать сразу для обоих устройств. Кроме того, мы

будем пользоваться фактом, что класс языков, распознаваемых ДКА, и класс языков, распознаваемых НКА, совпадают с классом регулярных языков (и потому совпадают между собой). Далее мы приведём алгоритмы, которые засвидетельствуют нужные для доказательства равенства классов включения.

Контрольный вопрос 8. Приведите формальное описание автомата из решения К.В. 6.

2.2 Способы описания автоматов

Описывать конечный автомат, явно выписывая все элементы набора, часто неудобно. Гораздо удобнее представить автомат с помощью графа или задать его таблицей. Мы приведём примеры таких описаний для автомата из решения К.В. 6, поэтому просим добросовестного читателя решить этот контрольный вопрос и К.В. 8, прежде чем двигаться дальше.

При задании автомата графом (рис. 3), вершины графа соответствуют состояниям, возле начального состояния q_0 стоит стрелка $\to$, а принимающие состояния (q_1) помечены двойным кружком. На рёбрах записаны символы, по которым и происходят переходы. Так ребро $q_0 \xrightarrow{a} q_1$ соответствует переходу $\delta(q_0, a) = q_1$.

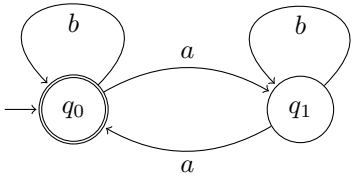


Рис. 3. Граф автомата $\mathcal{A}$.

Также автомат удобно задать таблицей переходов (рис. 4). В

первом столбце этой таблицы приведены все состояния автомата, а в первой строке — все символы алфавита, и быть может ε. В клетке на пересечении состояния q и символа σ записано значение $\delta(q, \sigma)$. Перед начальным состоянием указана стрелка, а принимающие помечены звёздочкой.

	a	b
$\to q_0^*$	q_1	q_0
q_1	q_0	q_1

Таблица переходов автомата $\mathcal{A}$.

	a	b	ε
$\to q_0$	q_1, q_0		q_1
q_1^*	q_0	q_1	

Таблица переходов автомата $\mathcal{B}$.

Рис. 4. Описание автоматов через таблицы.

Контрольный вопрос 9. Проверьте, что граф автомата $\mathcal{A}$ и таблица переходов автомата $\mathcal{A}$ действительно задают один и тот же автомат. Постройте по таблице переходов автомата $\mathcal{B}$ граф автомата.

2.3 Отступление об отношениях

Возможность задать конечный автомат графом — не случайность. Основная часть описания автоматов содержится в функции переходов $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$, однако эту функцию можно понимать и как отношение.

Бинарное отношения R — это подмножества декартова произведения двух множеств: $R \subseteq A \times B$. Отношения вида $R \subseteq A \times A$ удобно задать в виде ориентированного графа (быть может бесконечного, если такого множество A): ребро из вершины u ведёт в вершину v если и только если $(u, v) \in R$.

Зафиксировав символ алфавита σ, мы получим, что пары состояний (q, p), такие что $p \in \delta(q, \sigma)$ задают бинарное отношение

δ_σ. Граф этого отношения получится из графа автомата удалением всех рёбер кроме рёбер вида $q \xrightarrow{\sigma} p$.

Граф автомата можно понимать как объединение графов отношений δ_σ, но также можно считать, что граф автомата соответствует тернарному[6] отношению

$$D \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q = \{(q, \sigma, p) \mid p \in \delta(q, \sigma)\}.$$

В общем случае, под k-арным отношением понимается отношение $R \subseteq A_1 \times A_2 \times \ldots \times A_k$.

В силу ассоциативности декартова произведения, можно считать, что D — бинарное отношение: $D \subseteq (Q \times (\Sigma \cup \{\varepsilon\})) \times Q$.

Упражнение 6. Докажите, что каждой функции $f : A \to 2^B$ соответствует бинарное отношение $R \subseteq A \times B$.

Мы будем называть тернарное отношение D *отношением переходов* и будем дальше использовать для него то же обозначение, что и для функции переходов — δ.

Приведём важные свойства бинарных отношений специального вида. Отношение $R \subseteq A \times A$ называют *бинарным отношением на множестве* A.

- R *рефлексивно*, если $\forall a \in A : (a, a) \in R$;
- R *симметрично*, если $\forall (a, b) \in R : (b, a) \in R$;
- R *транзитивно*, если $\forall (a, b), (b, c) \in R : (a, c) \in R$.

Если выполняются все три этих свойств, то R — отношение эквивалентности. Отношения такого вида потребуются нам дальше, сейчас мы не будем изучать их подробно.

Помимо обозначения $(a, b) \in R$ используют обозначение aRb. Бинарные отношения знакомы читателю со школьной скамьи: математические обозначения $=, <, \geqslant$ и им подобные на самом

[6] Тернарное отношение — это отношение арности 3.

деле бинарные отношения. Так, запись $3 \leqslant 5$ означает, что 3 и 5 находятся в отношении $\leqslant$.

Упражнение 7. Какими из определённых выше свойствами обладают отношения $=, <, \leqslant, \neq$ на $\mathbb{R}$?

Итак, бинарному отношению R на множестве A соответсвует граф и пара (a, b) в отношении R, если в графе есть ребро из a в b. Или, иначе говоря, из a в b есть путь длины 1. Будем говорить, что отношение R^+ — *транзитивное замыкание* отношения R, если $(a, b) \in R^+$ тогда и только тогда, когда в графе R есть путь из вершины a в вершину b длины не меньше 1.

Рефлексивным замыканием отношения R назовём отношение $R \cup \{(a, a) \mid a \in A\}$: эта операция дополняет отношение R до рефлексивного.

Одновременное рефлексивное и транзитивное замыкание обозначим R^*.

Транзитивное замыкание можно было бы также определить и через операцию композиции отношений, схожей с композицией функций. Под *композицией* $P \circ Q$ отношений P, Q на A понимают отношение

$$P \circ Q = \{(a, c) \mid \exists b \in A : (b, c) \in P, (a, b) \in Q\}.$$

Обратный порядок выбран так, чтобы операция была согласована с композицией функций:

$$f \circ g(x) = f(g(x)).$$

Упражнение 8. Проверьте, что функция — частный случай отношения. Какие отношения являются функциями?

Упражнение 9. Загадка: как операции R^+ и R^* согласуются с операциями замыкания Клини?

Ответ на загадку приведён в разделе 2.8.

2.4 Язык конфигураций

Назовём *конфигурацией* пару $(q, w) \in Q \times \Sigma^*$. На множестве конфигураций введём соответствующее тактам работы автомата бинарное отношение $\vdash$: для всех $q' \in \delta(q, a)$ и для всех $w \in \Sigma^*$ $(q, aw) \vdash (q', w)$. Таким образом, автомат *принимает слово*, если существует последовательность конфигураций

$$(q_0, w_1 w[2, n]) \vdash (q'_1, w[2, n]), (q'_1, w_2 w[3, n]) \vdash (q'_2, w[3, n]), \ldots$$

$$\ldots, (q'_{n-1}, w_n) \vdash (q'_n, \varepsilon),$$

в которой $q'_n \in F$. Или в терминах рефлексивного и транзитивного замыкания $\exists q'_n : (q_0, w) \vdash^* (q'_n, \varepsilon)$.

Если термины «бинарное отношение» и «рефлексивное и транзитивное замыкание» кажутся сложными, можно считать, что «$\vdash$» означает такт работы автомата, а «$(q, u) \vdash^* (p, v)$» означает что автомат может попасть в конфигурацию (p, v), начав из конфигурации (q, u).

Также мы будем использовать обозначение $q \xrightarrow{u} p$, которое означает, что из состояния q есть путь в состояние p вдоль которого написано слово u. Это обозначение можно формализовать с помощью конфигураций:

$$q \xrightarrow{u} p \iff (q, u) \vdash^* (p, \varepsilon).$$

Множество всех слов, принимаемых автоматом $\mathcal{A}$ будем обозначать $L(\mathcal{A})$. Автомат $\mathcal{A}$ *принимает* язык L, если $\forall w \in L : w \in L(\mathcal{A})$, другими словами $L \subseteq L(\mathcal{A})$. Если при этом, $L(\mathcal{A}) \subseteq L$, то автомат $\mathcal{A}$ *распознаёт* язык L. Здесь есть тонкая лингвистическая разница, слова «принимает» и «распознаёт», конечно схожи, но за ними стоят разные определения. Также наравне со словом «распознаёт» используют слово «*допускает*».

2.5 Построение ДКА по РВ

В этом разделе мы ответим на поставленный ранее вопрос: как алгоритмически проверить, порождает ли регулярное выра-

жение R слово w. Для этого мы опишем алгоритм построения по РВ R эквивалентного ДКА $\mathcal{A}$. На практике этим алгоритмом стоит пользоваться с осторожностью, и позже мы обсудим причины этого. Однако из существования этого алгоритма следует важный теоретический результат: любой регулярный язык, распознаётся некоторым конечным автоматом. Верно и обратное: конечные автоматы распознают только регулярные языки; алгоритм построения ДКА по РВ мы изучим позже.

Мы ограничимся здесь лишь РВ, в которые не входят пустые множества и соответственно пустые слова (см. К.В. 4).

Начнём с объяснения общей идеи. В любое регулярное выражение входит конечное число символов; добавим к регулярному выражению в начало и конец символы $\triangleright$ и $\triangleleft$, маркеры начала и конца порождённого слова, и занумеруем все символы:

$$R = \underset{0}{\triangleright}(\underset{1}{a}\underset{2}{a} \mid \underset{3}{b})^* \underset{4}{b}(\underset{5}{a} \mid \underset{6}{b})^* \underset{7}{a} \underset{8}{\triangleleft}. \tag{2}$$

Пусть слово $w = w_1 \dots w_n, w_i \in \Sigma$ порождается РВ. Одно слово может вообще говоря быть порождено несколькими способами, но выбрав произвольный, мы получим, что каждый символ w_i порождён в конечном итоге некоторым символом РВ с номером j; будем называть такие номера *позициями*. Например, слову $bbba$ могут соответствовать последовательности позиций $3, 4, 6, 7$ или $3, 3, 4, 7$.

Важно, что каждой букве слова может соответствовать лишь конечное число позиций РВ, и кроме того за каждой позицией могут идти только определённые позиции. Так, первый символ слова, порождённого R, может быть только на позиции из множества $\{1, 3, 4\}$, а после символа на позиции 1 может идти только символ на позиции 2.

Все эти свойства можно вычислить устройством с конечной памятью. В начале работы автомат ожидает, что ему на вход будут поданы символы, которые могут иметь позиции[7] $\{1_a, 3_b, 4_b\}$. После обработки каждого символа, автомат меняет допустимое

[7]Запись i_a означает, что i-ым символом РВ является a.

множество позиций: если первым был обработан символ a, то дальше может идти символ с позицией из множестве $\{2_a\}$, то есть только a. После же первого символа b дальше может идти только символ с одной из позиций множества $\{3_b, 4_b, 5_a, 6_a, 7_a\}$, то есть как a, так и b.

Формализуем процесс построения этих множеств. Обозначим через $\mathsf{followpos}(i)$ множество позиций, которые могут идти после позиции i. Тогда, если на некотором шаге допустимы символы с позиций из множества A, то после обработки символа σ допустимы символы с позиций из множества

$$A_\sigma = \bigcup_{i_\sigma \in A} \mathsf{followpos}(i_\sigma). \tag{3}$$

Из сказанного выше явно следует критерий принадлежности слова языку, порождаемому РВ.

Утверждение 1. *Слово w порождается регулярным выражением R тогда и только тогда, когда существует последовательность позиций $i_{w_1}, i_{w_2}, \ldots, i_{w_n}$, такая что*

- $i_{w_1} \in \mathsf{followpos}(0_\triangleright)$;
- $i_{w_k} \in \mathsf{followpos}(i_{w_{k-1}})$ *для* $k \leqslant n$;
- $i_\triangleleft \in \mathsf{followpos}(0_{w_n})$.

Теперь ясно как построить ДКА по РВ. Состояниями автомата будут множества позиций. Начальным состоянием Q_0 будет множество $\mathsf{followpos}(0_\triangleright)$ допустимых позиций для первого символа, а дальше для каждого состояния A и символа σ определим переходы согласно формуле (3). Принимающими состояниями будут те состояния, которые содержат позицию $i_\triangleleft$.

Докажем, что такая процедура корректна.

Утверждение 2. *ДКА $\mathcal{A}$, построенный по РВ R, описанным образом распознаёт язык $L(R)$.*

Доказательство. Докажем, что все слова, принятые автоматом, порождаются РВ. Если слово w принято описанным автоматом, то у автомата есть ход

$$Q_0 \xrightarrow{w_1} Q_1 \xrightarrow{w_2} Q_2 \xrightarrow{w_3} \dots \xrightarrow{w_n} Q_n,\ Q_n \in F.$$

Значит, у символа w_n есть позиция $i_{w_n} \in Q_{n-1}$. Но позиция i_{w_n} оказалась в состоянии Q_n, потому что она принадлежит некоторому множеству $\mathsf{followpos}(i_{w_{n-1}})$, где $i_{w_{n-1}} \in Q_{n-2}$, значит подходящая позиция есть и для символа w_{n-1}. Продолжая это рассуждение по индукции, получаем, что символы w действительно можно занумеровать так, что $i_{w_j} \in \mathsf{followpos}(i_{w_{j-1}})$. Поскольку $i_{w_1} \in Q_0$, а $i_\lhd \in Q_n$, получаем, что i_{w_1} — допустимая позиция для начала слова, порождаемого РВ, а i_{w_n} — допустимая позиция для конца слова, а значит слово w действительно порождается РВ R. Итак, мы доказали, что $L(\mathcal{A}) \subseteq L(R)$.

Включение в обратную сторону очевидно. Если слово w порождается РВ, то позиция w_1 принадлежит множеству $Q_0 = \mathsf{followpos}(0_\rhd)$ по определению, а позиция w_i принадлежит Q_{i-1} по построению (формула 3). Поскольку Q_n — принимающее состояние, то $i_\lhd \in \mathsf{followpos}(i_{w_n})$. □

Тут есть небольшой обман трудящихся. Формально ещё нужно доказать, что множество $\mathsf{followpos}(i)$ вообще осмысленно: номера символов, которые могут идти после i-го, не зависят от предыдущих или последующих позиций порождаемого слова. Мы оставим это читателю в качестве упражнения (как принято делать в математических статьях, когда всё «очевидно»).

Этот момент должен проясниться после алгоритма вычисления функции $\mathsf{followpos}$ — заметьте, что пока никакого алгоритма построения ДКА по РВ приведено не было, поскольку не было приведено алгоритма вычисления этой функции! Был приведён рецепт, корректность которого была доказана, но вычисление основного ингредиента описано не было.

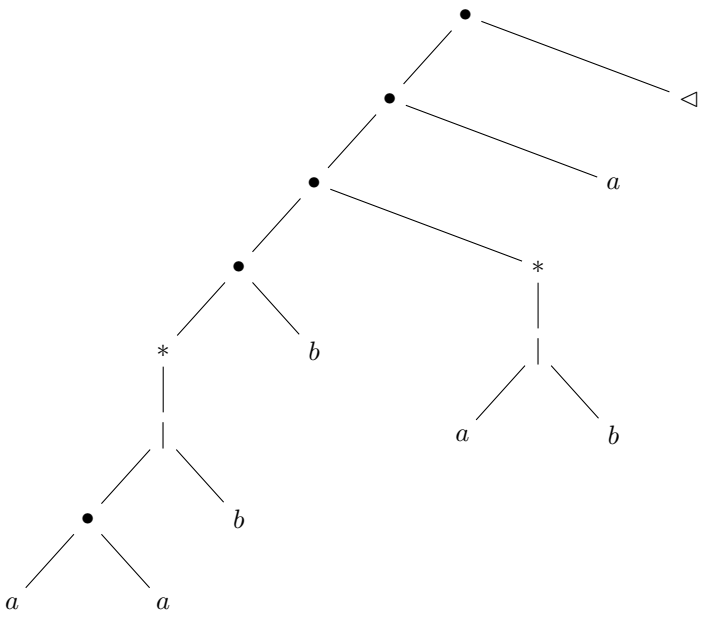


Рис. 5. Синтаксическое дерево

2.5.1 Вычисление followpos

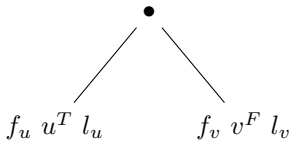
Алгоритм вычисления followpos тоже придётся начать издалека. Позволим себе ещё одно жульничество, допустимость которого мы объясним значительно позже. Представим, что есть алгоритм, который строит по РВ дерево как на рис[8] 5.

Каждая вершина u дерева является корнем поддерева, которое задаёт регулярное выражение R_u. Корню соответствует само регулярное выражение R, а листьям — буквы. Для вычисления функции followpos нам потребуется вычислить следующие атрибуты каждой вершины:

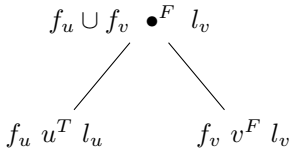
[8] Рисунки для этого примера подготовлены В. Алексеевым.

- $\mathsf{firstpos}(u)$ — множество номеров позиций, с которых может начинаться слово из R_u (записывается слева от u);
- $\mathsf{lastpos}(u)$ — множество номеров позиций, в которых может заканчиваться слово из R_u (записывается справа от u);
- $\mathsf{nullable}(u)$ — содержит ли R_u пустое слово (T или F).

Эти атрибуты вычисляются снизу-вверх: зная атрибуты дочерних узлов, вычисляются атрибуты родителя; атрибуты листьев вычисляются очевидным образом. Правила, по которым вычисляются атрибуты достаточно просты, но их получается достаточно много (они разные в зависимости от вершины и атрибутов детей). Приведём пример вычисления атрибутов в одном из случаев (атрибуты $\mathsf{firstpos}$ и $\mathsf{lastpos}$ обозначены как f и l):



Атрибуты узла конкатенации $\bullet$ приведены на картинке:



Объясним, почему атрибуты именно такие. Регулярное выражение $R_u \cdot R_v$ порождает слово $z = xy$, такое что $x \in R_u$, $y \in R_v$, и при этом слово x может быть пустым, а слово y – нет. Отсюда, если $x \neq \varepsilon$, то z начинается с тех же символов, что и x, а значит $f_u \subseteq f_\bullet$, а если $x = \varepsilon$, то z начинается с тех же символов, что и слово y, поэтому $f_v \subseteq f_\bullet$, отсюда $f_\bullet = f_u \cup f_v$. Поскольку слово z

имеет непустой суффикс y, порождённый R_v, то $l_\bullet = l_v$, и кроме того слово z не может быть пустым, а потому $\mathsf{nullable}(\bullet) = F$.

Упражнение 10. Восстановите правила вычисления атрибутов $\mathsf{firstpos}$, $\mathsf{lastpos}$ и $\mathsf{nullable}$ для остальных случаев (всевозможных операций и значений атрибута $\mathsf{nullable}$) и сверьте получившийся свод правил с правилами из [3].

Контрольный вопрос 10. Вычислите все атрибуты для дерева с рисунка 5 самостоятельно и сверьте результат вычислений с рисунком 6.

Опишем, алгоритм вычисления множества $\mathsf{followpos}(i)$, на вход которого подаётся дерево с вычисленными атрибутами.

- Положим сначала $\mathsf{followpos}(i) = \varnothing$ для каждого номера i.
- Для каждой вершины-конкатенации $u \cdot v$, для каждого $i \in \mathsf{lastpos}(u)$ добавим к $\mathsf{followpos}(i)$ множество $\mathsf{firstpos}(v)$.
- Для каждой вершины-итерации u^*, для каждого номера $i \in \mathsf{lastpos}(u)$ добавим к $\mathsf{followpos}(i)$ множество $\mathsf{firstpos}(u)$.

Упражнение 11. Докажите корректность данного алгоритма.

Контрольный вопрос 11. Вычислите множества $\mathsf{followpos}(i)$ для РВ 2 не заглядывая в следующую таблицу.

2.5.2 Построение автомата

Результаты вычислений атрибутов, оставленные выше читателю в качестве упражнений, приведены на рисунках 6 и 7.

Для построения автомата дело осталось за малым. Вычислить начальное состояние и выполнить построение ДКА по описанному рецепту.

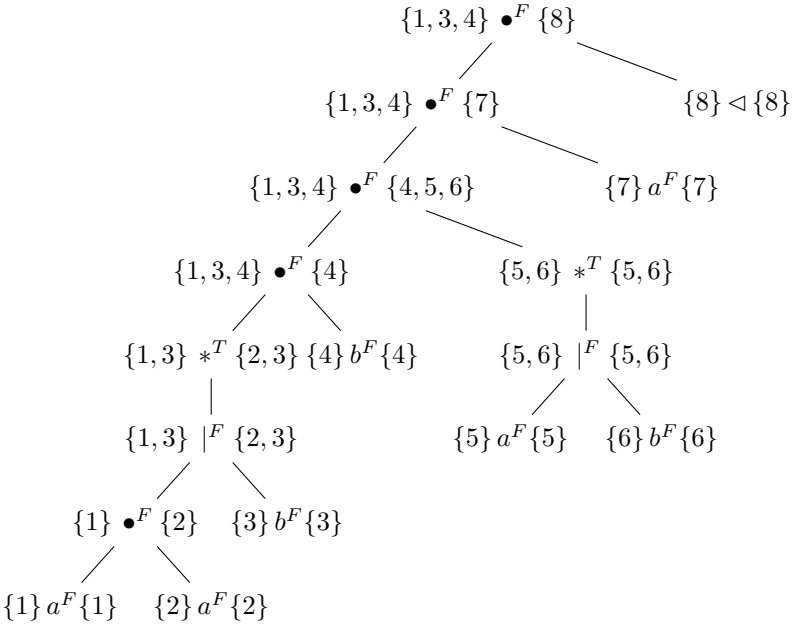


Рис. 6. Вычисление атрибутов firstpos, lastpos и nullable.

i	followpos(i)
1_a	2_a
2_a	$1_a, 3_b, 4_b$
3_b	$1_a, 3_b, 4_b$
4_b	$5_a, 6_b, 7_a$
5_a	$5_a, 6_b, 7_a$
6_b	$5_a, 6_b, 7_a$
7_a	$8_\triangleleft$

Рис. 7. Таблица followpos для РВ 2.

Контрольный вопрос 12. Как вычислить начальное состояние? Подумайте над этим и попытайтесь построить автомат самостоятельно перед тем как прочесть следующий абзац.

Решение. Начальное состояние — множество номеров позиций, с которых может начинаться слово. Это множество по определению является значением атрибута firstpos корня дерева. Под корнем всегда понимается вершина-конкатенация, правый потомок которой маркер $\triangleleft$. Заметьте, что для вычислений нам достаточно использовать только один маркер — два маркера удобней для теоретических обоснований. □

Приведём теперь граф автомата, построенного по данному алгоритму (рис. 8).

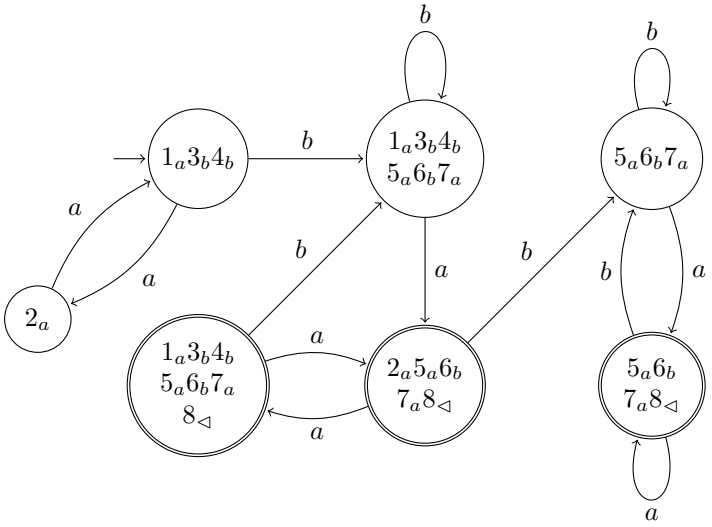


Рис. 8. Автомат, построенный по РВ 2.

Разобрались с алгоритмом? Тогда без труда справитесь со следующим контрольным вопросом.

Контрольный вопрос 13. Почему приведённый в этом разделе алгоритм не является алгоритмом построения ДКА по РВ?

2.6 Замкнутость относительно теоретико-множественных операций

Определение класса регулярных языков влечёт его замкнутость относительно операции объединения. Как это выразить на языке автоматов: построить по автоматам $\mathcal{A}$ и $\mathcal{B}$ автомат, распознающий $L(\mathcal{A}) \cup L(\mathcal{B})$, — обычно не сразу приходит в голову. Предлагаем читателю попробовать решить эту задачу, не заглядывая дальше.

Если это сходу не получается, то лучше начать с ответа на следующий контрольный вопрос

Контрольный вопрос 14. Как построить по (детерминированным) автоматам $\mathcal{A}$ и $\mathcal{B}$ автомат $\mathcal{C}$, распознающий язык

$$L(\mathcal{C}) = L(\mathcal{A}) \cap L(\mathcal{B})?$$

Перед спойлером в следующем абзаце отметим, что замкнутость регулярных языков относительно пересечения трудно заметить, глядя на теоретико-множественное определение.

Решение. Начнём с наивного ответа, который дальше преобразуем в алгоритм. Представим, что автоматы $\mathcal{A}$ и $\mathcal{B}$ работают параллельно. Сначала мысленно поставим две ленты рядом. Потом заметим, что автоматы смогут прекрасно ужиться и на одной ленте, на которой две головки. Немного подумав, сообразим, что вместо двух головок, достаточно использовать одну, а в оперативной памяти хранить состояния обоих автоматов, а точнее их пару, и пересчитывать их после обработки каждой буквы. □

Попробуйте воплотить эту идею в жизнь, решив следующую задачу:

Задача 5. Постройте ДКА $\mathcal{A}$, $\mathcal{B}$ и $\mathcal{C}$, такие что

а) ДКА $\mathcal{A}$ распознаёт язык, все слова которого содержат чётное число нулей;

б) ДКА $\mathcal{B}$ распознаёт язык, все слова которого содержат нечётное число единиц;

в) ДКА $\mathcal{C}$ распознаёт язык, все слова которого содержат чётное число нулей и нечётное число единиц.

Контрольный вопрос 15. Преобразуйте эту идею в алгоритм.

Решение. Зафиксируем описания автомата $\mathcal{A}$:

$$\mathcal{A} = (Q_{\mathcal{A}},\ \Sigma,\ q_0^{\mathcal{A}},\ \delta_{\mathcal{A}},\ F_{\mathcal{A}}).$$

Автомат $\mathcal{B}$ имеет аналогичное описание. Построим по ним автомат $\mathcal{C}$:

$$\mathcal{C} = (Q_{\mathcal{A}} \times Q_{\mathcal{B}},\ \Sigma,\ (q_0^{\mathcal{A}}, q_0^{\mathcal{B}}),\ \delta_{\mathcal{C}},\ F_{\mathcal{A}} \times F_{\mathcal{B}}),$$

где функция переходов определена следующим образом:

$$\delta_{\mathcal{C}}((q_{\mathcal{A}}, q_{\mathcal{B}}), \sigma) = (\delta_{\mathcal{A}}(q_{\mathcal{A}}, \sigma), \delta_{\mathcal{B}}(q_{\mathcal{B}}, \sigma)).$$

□

Упражнение 12. Докажите корректность этого алгоритма.

Приведённый алгоритм называют *конструкцией произведения*, как и схожие ему алгоритмы, изобрести которые мы оставили читателю в качестве задач.

Контрольный вопрос 16. Заменим в конструкции произведения множество принимающих состояний на множество

$$F_{\mathcal{C}} = F_{\mathcal{A}} \times Q_{\mathcal{B}} \cup Q_{\mathcal{A}} \times F_{\mathcal{B}}.$$

Верно ли, что тогда автомат $\mathcal{C}$ распознаёт язык $L(\mathcal{A}) \cup L(\mathcal{B})$?

Ответ на К.В. 16: *Нет!* Если вы ответили «да», то найдите в своих рассуждениях ошибку и постарайтесь придумать контрпример. Если самостоятельно придумать пример не получилось, то его можно найти решив задачу 11. □

2.6.1 Дополнение

С помощью детерминированных автоматов легко доказать замкнутость регулярных языков относительно операции дополнения: если язык L — регулярный, то регулярно и его дополнение $\bar{L} = \Sigma^* \setminus L$. Построим автомат $\bar{\mathcal{A}}$ по автомату $\mathcal{A}$, заменив множество принимающих состояний F на его дополнение — $Q \setminus F$.

Контрольный вопрос 17. Верно ли, что автомат $\bar{\mathcal{A}}$ распознаёт язык $\overline{L(\mathcal{A})}$?

Spoiler Alert!

Решение. Ответ на этот вопрос отрицательный. Если вы вдруг ответили «да», то выполните следующие действия, чтобы сообразить, что же пошло не так.

- Постройте (мысленно) автомат $\bar{\mathcal{A}}$ по автомату $\mathcal{A}$ с рис. 8;
- Подайте на вход каждому автомату слово ab и проверьте, принимает ли хотя бы один из автоматов это слово;
- Сделайте выводы по результатам эксперимента и предложите исправленную версию алгоритма, прежде чем читать дальше.

Чтобы идея с перекраской сработала, нужно, чтобы в результате обработки каждого слова, автомат $\mathcal{A}$ оказывался в некотором состоянии. ДКА, обладающие таким свойством называют *полными* — для каждого состояния в полном автомате определён переход по каждой букве алфавита. □

Контрольный вопрос 18. Предложите алгоритм, который по ДКА $\mathcal{A}$ строит полный эквивалентный ДКА $\mathcal{A}'$ ($L(\mathcal{A}) = L(\mathcal{A}')$).

Решение. Если ДКА $\mathcal{A}$ уже полный, то ничего с ним делать не нужно. Если же нет, добавим к нему состояние q_D с переходами в себя по каждому символу алфавита: $\forall \sigma \in \Sigma : q_D \xrightarrow{\sigma} q_D$.

Направим в q_D все переходы, которые раньше были не определены: если $\delta_{\mathcal{A}}(q, \sigma) = \varnothing$, то $\delta_{\mathcal{A}'}(q, \sigma) = q_D$.

Докажем корректность. Автомат $\mathcal{A}'$ является полным по построению — все неопределённые переходы в $\mathcal{A}$ были доопределены. При этом автоматы $\mathcal{A}$ и $\mathcal{A}'$ эквивалентны, поскольку если после обработки слова w автомат $\mathcal{A}$ оказывался в состоянии q, то автомат $\mathcal{A}'$ окажется в том же состоянии. Если же при обработке слова w некоторый переход был неопределён, то автомат $\mathcal{A}'$ попадёт в непринимающее состояние q_D. □

Итак, чтобы построить по ДКА $\mathcal{A}$ ДКА $\mathcal{B}$, распознающий язык $\overline{L(\mathcal{A})}$, нужно сначала пополнить автомат $\mathcal{A}$ и применить к получившемуся автомату $\mathcal{A}'$ перекраску: $\mathcal{B} = \overline{\mathcal{A}'}$.

2.7 Задачи и упражнения

6. Постройте РВ, порождающее язык $L(\mathcal{B})$; автомат $\mathcal{B}$ задан таблицей на рис. 4.

7. Построить РВ, эквивалентное языку, распознаваемому ДКА $\mathcal{B}_1$ с рис. 9.

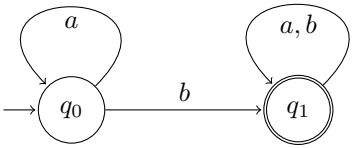


Рис. 9. Автомат $\mathcal{B}_1$.

8. Постройте[9] ДКА, распознающий язык $\Sigma^* aab \Sigma^*$

9. Построить ДКА по РВ по алгоритму:

а°) $a(a^*|ba)^*(b^*|a)$; **б)** $(b \mid ab^*)^* b(a \mid b)$; **в)** $(a \mid b)((a \mid bb)^* ab)^*$.

10. Постройте по ДКА $\mathcal{A}$, распознающий язык $\Sigma^* \setminus L(\mathcal{A})$, где $\mathcal{A}$ — автомат **а°)** из задачи **9(а)**; **б)** с рис. 9; **в)** с рис. 8.

11. ДКА $\mathcal{A}$ и ДКА $\mathcal{B}_1$ — автоматы с рис. 3 и рис. 9 соответственно. Используя конструкцию произведения, постройте ДКА $\mathcal{C}$, распознающий язык: **а)** $L(\mathcal{A}) \cap L(\mathcal{B}_1)$; **б)** $L(\mathcal{A}) \cup L(\mathcal{B}_1)$; **в)** $L(\mathcal{A}) \setminus L(\mathcal{B}_1)$; **г)** $L(\mathcal{A}) \triangle L(\mathcal{B}_1)$.

12. Предложите полиномиальный алгоритм, который получив на вход ДКА $\mathcal{A}$ и $\mathcal{B}$ проверяет, совпадают ли языки $L(\mathcal{A})$ и $L(\mathcal{B})$.

Упражнение 13. Докажите, что не смотря на то, что алгоритм построения ДКА по РВ из раздела 2.5 применим не для всех

[9] Лучше построить ДКА «методом внимательного вглядывания», после чего доказать корректность. М-построение и А-построение

РВ, из него следует, что для любого РВ R существует ДКА, распознающий $L(R)$.

Упражнение 14. Дополните алгоритм построения ДКА по РВ так, чтобы он стал алгоритмом построения ДКА по произвольному РВ.

2.8 Ответы на контрольные вопросы

К.В. 7. Ответ: *Да на оба вопроса.* □

К.В. 8.

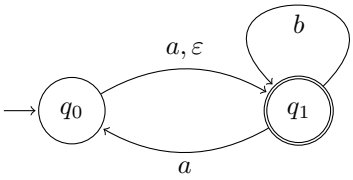
- $Q = \{q_0, q_1\}$
- $\Sigma = \{a, b\}$
- q_0
- $\delta : (q_0, a) \to q_1, (q_1, a) \to q_0, (q_0, b) \to q_0, (q_1, b) \to q_1$.
- $F = \{q_0\}$

Или можно записать весь набор как

$$(\{q_0, q_1\}, \{a, b\}, q_0, \{(q_0, a) \to q_1, (q_1, a) \to q_0,$$
$$(q_0, b) \to q_0, (q_1, b) \to q_1\}, \{q_0\}).$$

□

К.В. 9. Граф автомата $\mathcal{B}$:



□

К.В. 13. Как мы и обещали в начале раздела, он не учитывает случай, когда на некотором шаге построения регулярного языка используется пустое множество, что осмысленно, только если оно превращается в пустое слово (см. К.В. 4). □

Упр. 9. Представим, что бинарное отношение $R \subseteq A \times A$ — это язык над алфавитом $A \times A$ (не обязательно конечным). Тогда R^+ — это язык, получаемый из R замыканием Клини. Но если в формуле

$$R^+ = R \cup R \cdot R \cup R \cdot R \cdot R \cup \ldots$$

заменить операцию конкатенации на операцию композиции, то получится в точности формула для транзитивного замыкания:

$$R^+ = R \cup R \circ R \cup R \circ R \circ R \cup \ldots$$

Рефлексивное и транзитивное замыкание R^* также получается из итерации Клини, если заменить ещё пустое слово на тождественное отношение $\mathrm{Id} = \{(a, a) \mid a \in A\}$. □

3 Недетерминированные конечные автоматы

В предыдущем разделе мы работали с детерминированными автоматами, хотя формальное определение было приведено и для недетерминированного. Напомним особенности этого определения на примере автомата, заданного графом

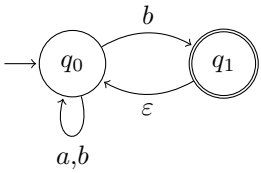


Рис. 10. НКА $\mathcal{A}$

В отличие от детерминированных автоматов, НКА может иметь два перехода из одного состояния по одному символу: $\delta(q_0, b) = \{q_0, q_1\}$. Помимо обычных переходов недетерминированные автоматы, имеют также ε-переходы, т.е. переходы вида $\delta(q_i, \varepsilon) = q_j$. Наличие таких переходов означает, что попав в состояние q_i, автомат может перейти в состояние q_j не обрабатывая следующий символ слова.

Контрольный вопрос 19. Опишите язык, распознаваемый автоматом $\mathcal{A}$ (например, в виде РВ).

Решение. Легко видеть, что автомат распознаёт язык $\Sigma^* b$, состоящий из слов, оканчивающихся на b. Чтобы принять слово. оканчивающееся на b автомату достаточно оставаться в состоянии q_0 до последнего символа и при обработке последнего b перейти в принимающее состояние q_1; отсюда $\Sigma^* b \subseteq L(\mathcal{A})$. Заметив, что автомат $\mathcal{A}$ оказывается в принимающем состоянии только после перехода по b получаем, что $L(\mathcal{A}) \subseteq \Sigma^* b$. □

В этом разделе мы докажем, что классы языков, распознаваемых НКА и ДКА совпадают. Этот результат может вызвать поначалу недоумение: детерминированные автоматы выглядят гораздо удобнее недетерминированных, зачем тогда изучать последние? Мы ответим и на этот вопрос.

3.1 Построение НКА по РВ

Алгоритм построения ДКА по РВ выглядит сложным. Строить НКА по РВ гораздо проще.

Для построения НКА по РВ будем использовать определение регулярного языка. Напомним, что по определению любой регулярный язык можно построить согласно согласно следующей схеме:

1. $\varnothing \in \mathsf{REG}$.
2. $\forall \sigma \in \Sigma\colon \{\sigma\} \in \mathsf{REG}$.
3. $\forall X, Y \in \mathsf{REG}\colon$
   а) $X \cdot Y \in \mathsf{REG}$;
   б) $X \mid Y \in \mathsf{REG}$;
   в) $X^* \in \mathsf{REG}$.

Мы будем строить НКА по РВ для каждого пункта. Более того, нам потребуется соблюсти технические условия:

(i) Каждый НКА имеет ровно одно принимающее состояние;

(ii) В начальное состояние НКА не ведёт ни один переход;

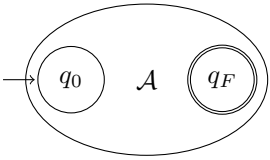
(iii) Из принимающего состояния НКА не выходит ни одного перехода.

Построить такие НКА для пунктов 1 и 2 не представляет труда и мы оставим их читателю в качестве контрольного вопроса.

Контрольный вопрос 20. Постройте НКА с единственным принимающим состоянием для языка **а)** $\varnothing$; **б)** $\{a\}$.

Автоматы для пунктов 3.а и 3.б тоже довольно простые, и читатель легко сможет построить их самостоятельно, если уже понял жанр этого алгоритма. В любом случае, мы приведём оставшиеся построения начиная со следующего абзаца.

Допустим уже построены автоматы $\mathcal{A}$ и $\mathcal{B}$ (удовлетворяющие (i-iii)) для регулярных языков X и Y соответственно. Будем схематично обозначать автоматы эллипсами, и помечать в них только начальное и принимающее состояние. Таким образом, автомат $\mathcal{A}$ имеет вид



В дальнейшем, мы будем предполагать, что начальное состояние на схеме находится слева, а принимающее справа. Построим явно автомат распознающий $X \cdot Y$.

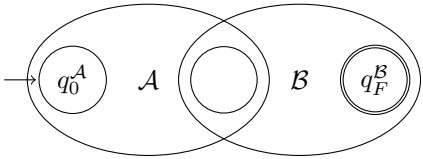


Рис. 11. Автомат для конкатенации.

Как видно из рисунка, результирующий автомат получен из автоматов $\mathcal{A}$ и $\mathcal{B}$ следующим образом. Начальное состояние ав-

томата $\mathcal{B}$ «склеено» с начальным состоянием автомата $\mathcal{A}$: мы удалили из автомата $\mathcal{B}$ начальное состояние $q_0^{\mathcal{B}}$, а все переходы, которые вели из него, добавили к принимающему состоянию $q_F^{\mathcal{A}}$ автомата $\mathcal{A}$, которое сделали непринимающим.

Упражнение 15. Доказать, что построенный автомат распознаёт язык $X \cdot Y$.

Для построения языка $X \mid Y$ используем следующую конструкцию:

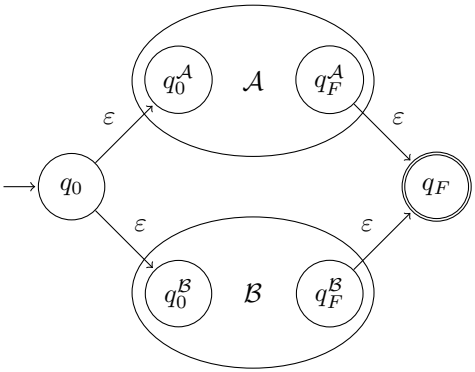


Рис. 12. Автомат для объединения.

Упражнение 16. Доказать, что построенный автомат распознаёт язык $X \mid Y$.

И наконец перейдём к построению автомата для языка X^* на рис. 13.

Упражнение 17. Доказать, что построенный автомат распознаёт язык X^*.

Помимо того, что приведённый алгоритм достаточно простой, легко видеть, что полученный автомат имеет примерно такой же размер, как и регулярное выражение: если у РВ длина n

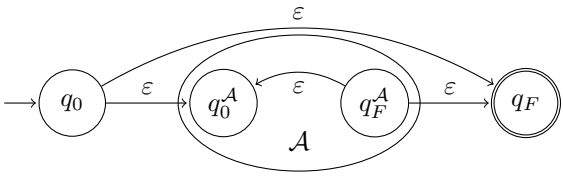


Рис. 13. Автомат для итерации.

(число символов + число операций), то у построенного автомата $\Theta(n)$ состояний[10].

Упражнение 18. Докажите это.

Задача 13. Пусть $\mathcal{A}$ и $\mathcal{B}$ — НКА, у которых ровно одно принимающее состояние. Верно ли, что автомат построенный по схеме на рис. 11 будет распознавать язык $L(\mathcal{A}) \cdot L(\mathcal{B})$?

3.2 Построение ДКА по НКА

В этом разделе мы приведём алгоритм построения ДКА по НКА из которого следует, что вычислительная сила у детерминированных и недетерминированных автоматов одинаковая. Но перед этим разберёмся как проверить, принимает ли НКА $\mathcal{A}$ слово w.

Если принимает, то в графе $\mathcal{A}$ существует путь из начального состояния в принимающее, символы на рёбрах которого образуют слово w; если не принимает, то такого пути нет.

Контрольный вопрос 21. Постройте алгоритм, который отвечает на вопрос $w \stackrel{?}{\in} L(\mathcal{A})$, получив на вход описание НКА $\mathcal{A}$ и слово w.

[10] Или не более чем cn состояний, где c — некоторая константа (на случай, если асимптотические обозначения не знакомы читателю.)

Решение. Модифицируем под наши нужды поиск в ширину (на графе НКА). На k-ом шаге алгоритма будем находить все состояния $\mathcal{A}$, достижимые по префиксу w длины k из начального состояния q_0. Будем обозначать такие множества Q_k. Очевидно, что НКА $\mathcal{A}$ принимает слово w тогда и только тогда, когда множество $Q_{|w|}$ содержит принимающее состояние.

Из определения ясно, что множество Q_0 состоит из состояния q_0 и всех состояний, достижимых из q_0 по ε-переходам (возможно по нескольким!). Найдём последние выполняя поиск в ширину только по ε-переходам начиная от q_0. Обозначим множество $\{p \mid q \xrightarrow{\varepsilon} p\}$ состояний, достижимых из q по ε-переходам через ε-$\mathsf{closure}(q)$; в этом абзаце мы привели алгоритм построения данного множества, которое называют *ε-замыканием*.

Итак, мы показали, что $Q_0 = \varepsilon\text{-}\mathsf{closure}(q_0)$. Чтобы вычислить Q_{k+1} зная Q_k нужно сначала найти все состояния, достижимые из Q_k переходом по w_k, после чего взять ε-замыкание от получившегося множества. Приведём получившийся алгоритм:

1. Построить множество $Q_0 = \varepsilon\text{-}\mathsf{closure}(q_0)$.
2. Для k от 1 до $|w|$ построить множества
   - $Q'_k = \{p \mid \exists q \in Q_k : p \in \delta(q, w_k),\ q \in Q_k\}$;
   - $Q_{k+1} = \varepsilon\text{-}\mathsf{closure}(Q'_k) = \bigcup\limits_{q' \in Q'_k} \varepsilon\text{-}\mathsf{closure}(q')$.
3. Если $Q_{|w|} \cap F_{\mathcal{A}} \neq \varnothing$, то ответ «Да», иначе «Нет».

Рис. 14. Алгоритм проверки $w \overset{?}{\in} L(\mathcal{A})$.

Осталось показать корректность шага 2. Действительно, если $p \in Q_{k+1}$, то в графе автомата $\mathcal{A}$ есть либо путь

$$q_0 \xrightarrow{w[1,k]} q \xrightarrow{w_{k+1}} p, \text{ либо путь } q_0 \xrightarrow{w[1,k]} q \xrightarrow{w_{k+1}} q' \xrightarrow{\varepsilon} p,$$

в которых $q \in Q_k$, а $q' \in Q'_k$ по определению этих множеств. $\square$

Контрольный вопрос 22. Оцените сложность приведённого (или вашего) алгоритма.

Приведённый алгоритм легко переделать в алгоритм построения ДКА по НКА. Мы рекомендуем попробовать читателю выполнить это упражнение самостоятельно, прежде чем читать дальше.

Контрольный вопрос 23. Приведите алгоритм построения ДКА по НКА.

Решение. Заметим, что число множеств Q_k, появлявшихся в алгоритме, конечно, поскольку $Q_k \subseteq 2^{Q_\mathcal{A}}$. Поэтому, всевозможные множества можно хранить на конечной памяти, они и будут состояниями ДКА, который мы назовём $2^\mathcal{A}$. Начальным состоянием будет состояние Q_0, принимающими будут множества Q_k, содержащие принимающие состояния $\mathcal{A}$, а переходы между состояниями определены как и в алгоритме 14:

$$\delta_{2^\mathcal{A}}(Q_k, a) = \varepsilon\text{-}\mathsf{closure}(\{p \mid \exists q \in Q_k : p \in \delta(q, a)\}).$$

Очевидно, что автомат $2^\mathcal{A}$ после обработки слова w оказывается в том же состоянии $Q_{|}w|$, что и алгоритм 14. Отсюда и следует корректность алгоритма построения ДКА по НКА. $\square$

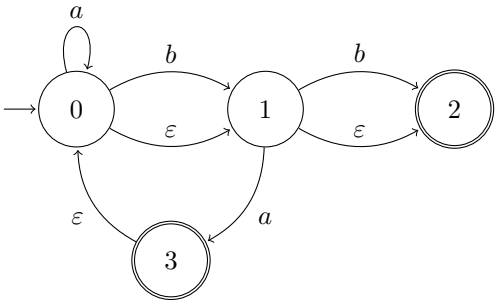
Приведём пример[11] работы данного алгоритма на рис. 15. Промежуточные вычисления удобно организовать в виде таблицы. Звёздочкой отмечены принимающие состояния.

3.3 О НКА и ДКА

Начнём с проверки внимательности читателя.

[11] Иллюстрации В. Алексеева.

НКА $\mathcal{A}$:



Вычисление таблицы переходов автомата $2^{\mathcal{A}}$:

$Q_{2^{\mathcal{A}}}$	$\delta(Q_k, a)$	$\delta(Q_k, b)$	
Q_0^*	q_0, q_1, q_2	Q_1^*	Q_2^*
Q_1^*	q_0, q_3, q_1, q_2	Q_1^*	Q_2^*
Q_2^*	q_1, q_2	Q_1^*	Q_3^*
Q_3^*	q_2	$\varnothing$	$\varnothing$

ДКА $2^{\mathcal{A}}$:

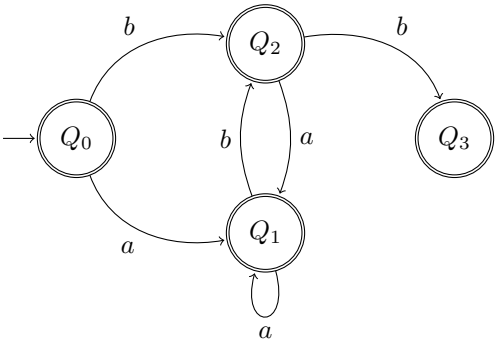


Рис. 15. Демонстрация алгоритма построения ДКА по НКА.

Контрольный вопрос 24. Верно ли, что если автомат $\mathcal{A}$ детерминированный, то он не является недетерминированным?

Решение. Мы выбрали определение 1 так, чтобы ДКА был частным случаем НКА, а потому ответ «Нет». □

Поэтому, из определения мы получаем, что НКА распознают все языки, распознаваемые ДКА, а из алгоритма построения ДКА по НКА мы получаем, что вычислительные силы этих моделей равны. Зачем же нам нужны обе?

Из алгоритма построения ДКА по НКА следует, что в получившемся ДКА $2^{\mathcal{A}}$ может оказаться $2^{|Q_{\mathcal{A}}|}$ состояний. Но достижима ли эта оценка на самом деле?

Чтобы дать ответ, нужно сначала формализовать вопрос. Если у НКА $\mathcal{A}$ два состояния, а у ДКА $2^{\mathcal{A}}$ восемь, то совершенно неясно есть ли тут экспоненциальный разрыв: если восемь — это два в кубе, то казалось бы да, а если дважды четыре, то вроде бы нет... Итак, экспоненциальный разрыв между НКА и ДКА достигается, если существует последовательность НКА $\mathcal{A}_1, \mathcal{A}_2, \ldots$, $|Q_{\mathcal{A}_i}| \to \infty$, такая что, если НКА $\mathcal{A}_i$ имеет n состояний, то любой ДКА, распознающий $L(\mathcal{A}_i)$, имеет не меньше 2^n состояний.

Вопрос этот не очень простой, поэтому мы сразу приведём ответ, но жаждущий самостоятельной работы читатель, может попробовать ответить на него самостоятельно.

Итак, между НКА и ДКА есть экспоненциальный разрыв и это означает, что некоторые языки, для которых существует короткое описание через регулярные выражения, невозможно распознать ДКА с небольшим числом состояний, относительно длины РВ. Тут мы немного забежали вперёд — оценку на длину РВ, построенного по НКА, мы обсудим позже. Но мы приведём сейчас пример РВ, на котором достигается экспоненциальный разрыв между РВ и ДКА, равно как и между НКА (построенному по РВ) и ДКА.

Зафиксируем последовательность языков R_i над алфавитом $\Sigma = \{a, b\}$, состоящих из слов, в которых на i-ом месте от конца

стоит b, т.е. $w[n-i+1] = b, n = |w|$. Каждый из языков последовательности регулярный в силу справедливости формулы $R_i = \Sigma^* b \Sigma^{i-1}$. При записи регулярного выражения для языка R_i множество Σ^{i-1} необходимо заменить на $(a \mid b) \cdot (a \mid b) \cdot \ldots \cdot (a \mid b)$, а посему длина РВ для R_i есть $O(i)$. Отсюда получаем, что НКА, построенный по этому РВ будет также иметь $O(i)$ состояний.

Для того, чтобы доказать экспоненциальный разрыв достаточно справиться со следующей задачей, к которой мы приведём решение, но рекомендуем пытливым умам попробовать решить её самостоятельно.

Задача 14. Докажите, что любой ДКА $\mathcal{A}_n$, распознающий язык R_n имеет не менее 2^n состояний.

Решение. Докажем от противного. Допустим, что найдётся ДКА $\mathcal{A}_n$ меньшего размера. Ясно, что в результате обработки произвольного слова w, автомат $\mathcal{A}_n$ должен оказаться в некотором состоянии $q_w : q_0 \xrightarrow{w} q_w$, потому что слово wba^{n-1} должно быть принято автоматом.

Тогда, по принципу Дирихле, среди всех слов длины n найдётся хотя бы два различных слова u и v, для которых $q_u = q_v$. Поскольку слова различаются, то $u[i] \neq v[i]$ для некоторого i. Пусть для определённости $u[i] = a$, а $v[i] = b$. Автомат $\mathcal{A}_n$ принимает слово va^{n-i+1}, поскольку оно принадлежит языку R_n, получаем отсюда, что $q_v \xrightarrow{a^{n-i+1}} q_f \in F$. Но тогда, автомат $\mathcal{A}_n$ принимает и слово ua^{n-i+1}, поскольку для него есть ход

$$q_0 \xrightarrow{u} q_v \xrightarrow{a^{n-i+1}} q_f.$$

Но в этом слове на n-ом месте от конца стоит символ a, а значит $ua^{n-i+1} \notin R_n$. Приходим тем самым к противоречию: автомат $\mathcal{A}_n$ принимает слово не из языка R_n. □

Итак, ДКА не стоит использовать на практике при парсинге произвольных регулярных выражений. Однако, для этих целей

сгодится алгоритм из решения К.В. 21. У него есть свои недостатки для практических нужд, поэтому программисты прибегают к ухищрениям решая задачу парсинга РВ. Иногда эти ухищрения приводят к экспоненциальному раздуванию используемых моделей как и в случае с ДКА. Это один из примеров того, почему при программировании важно понимать теорию. Небрежное к ней отношение может привести к уязвимостям в практических приложениях.

Ссылка на статью CIAA.

3.4 Задачи

15. Постройте НКА по регулярному выражению $(a(a|b))^*b$.

16. Постройте НКА $\mathcal{A}$, распознающий слова с суфиксом $abaab$.

17. Постройте по НКА $\mathcal{A}$ из предыдущей задачи эквивалентный ДКА $\mathcal{B}$ по алгоритму НКА $\to$ ДКА.

18. Обозначим через S_w язык слов с суфиксом w. Докажите или опровергните следующие утверждения:

а) ДКА, распознающий язык S_w имеет не менее $|w| + 1$ сосотяний;

б) Для каждого w существует ДКА с $|w| + 1$ сосотянием, распознающий язык S_w.

3.5 Ответы на контрольные вопросы

К.В. 20.

□

4 Автоматы и алгоритмы поиска образцов в тексте

Проблему поиска слова в тексте легко сформулировать в терминах регулярных выражений. Поскольку, с точки зрения формальных языков, и искомое слово и текст являются словами, то мы будем называть первое *образцом*.

Чтобы проверить, что в тексте t, слове над алфавитом Σ, встречается слово w достаточно проверить, принадлежит ли t языку $\Sigma^* w \Sigma^*$.

Для решения этой задачи можно построить НКА по РВ и воспользоваться алгоритмом проверки принадлежности слова языку, распознаваемому НКА, из решения К.В. 21. В автомате будет $O(|w|)$ состояний, но поскольку на каждом шаге алгоритм поддерживает подмножество состояний, то верхняя оценка на время работы этого алгоритма $O(|w| \times |t|)$.

В этом разделе мы изучим алгоритм Кнута-Морриса-Пратта, который решает эту задачу за время $O(|w| + |t|)$ и также основан на конечных автоматах, но уже детерминированных. Также мы изучим алгоритм Ахо-Корасик, который находит вхождение сразу нескольких образцов: на вход алгоритма подаются образцы $w_1, \ldots, w_n$ и текст t; алгоритм проверяет, что в t встречается хотя бы один из образцов за линейное время — $O(\sum |w_i| + |t|)$.

Точнее, мы исследуем более общую задачу. Построенные ниже автоматы будут попадать в принимающее состояние каждый раз когда в тексте встретился образец, то есть когда суффикс обработанного префикса текста t совпал с образцом. В случае одного образца (КМП-автомат), автомат будет распознавать язык $\Sigma^* w$. Такие автоматы также можно использовать для подсчёта числа вхождений образца в текст. Скажем, посчитать сколько раз в «Войне и мире» упоминается слово «дуб».

4.1 Алгоритм Кнута-Морриса-Пратта

Перед тем как изучать сам КМП-алгоритм, давайте обсудим почему эту задачу можно решить используя детерминированный автомат. Начнём с жадного алгоритма, который использует $O(|w|)$ памяти.

4.1.1 Жадный алгоритм и КМП-автомат

Алгоритм считывает посимвольно текст t и хранит в памяти самый длинный суффикс t, который совпадает с некоторым префиксом образца w. Ясно, что образец встречается в тексте каждый раз, когда этот суффикс совпадёт с самим w. А потому алгоритм просто проверяет на каждом шаге, совпал ли искомый суффикс текста с образцом. Для хранения искомого суффикса достаточно $O(|w|)$ памяти: достаточно хранить само слово w и его префикс — суффикс t.

Если каждый раз вычислять искомый суффикс полным перебором префиксов w, то получится алгоритм, работающий за $O(|w| \times |t|)$, что не лучше, чем использование НКА. Однако этот алгоритм легко улучшить до $O(|w|^2 + |t|)$-алгоритма благодаря следующему наблюдению.

Будем для удобства обозначать через $w[0]$ и $w[0,0]$ пустое слово, а при $i > 0$ считаем, что $w[0,i] = w[1,i]$. Пусть $w[0,i]$ — текущий искомый суффикс обработанной части текста:

$$t[0,j] = t[0,j-i]w[0,i].$$

Если следующий символ текста $t[j+1]$ совпадает с символом образца $w[i{+}1]$, то $w[0,i{+}1]$ и есть следующий искомый суффикс. Если же нет, то следующий искомый суффикс зависит только от символа $t[j+1]$ и префикса $w[0,i]$, но не от всего текста t. Докажем это.

Обозначим искомый суффикс на j-ом шаге (слова $t[0,j]$) через x_j. Поскольку $|x_j| = i$, а следующий символ текста не совпал

с образцом, то длина x_{j+1} будет строго меньше i. Добавив к этому, что x_{j+1} — префикс слова w, получаем, что x_{j+1} ещё и префикс слова $w[0,i]t[j+1]$ (т.к. $|w[0,i]t[j+1]| > |w[0,i]| > |x_{j+1}|$). Аналогично, из того, что x_{j+1} является суффиксом обработанной части текста $t[0,j+1] = t[0,j+1-i]w[0,i]t[j+1]$, получаем, что это также и суффикс $w[0,i]t[j+1]$ в силу соотношения длин

$$|t[0,j+1]| \geqslant |w[0,i]t[j+1]| > |x_j| > |x_{j+1}|.$$

Итак, мы получили, что x_{j+1} — это самый длинный суффикс слова $w[0,i]t[j+1]$, совпадающий с префиксом этого же слова, но не совпадающий с самим словом!

Определение 2. Назовём *префикс-функцией* $l : \Sigma^* \to \Sigma^*$ функцию, возвращающую самый длинный суффикс y слова x, совпадающий с некоторым префиксом x, отличным от x.

То есть, мы доказали, что если символ $t[j+1]$ не продолжает суффикс $x_j = w[0,i]$ до $w[0,i+1]$, то $x_j = l(w[0,i]t[j+1])$.

Контрольный вопрос 25. Вычислите значение префикс-функции $l(w)$ для слова w:

а) a^n; **б)** $babab$; **в)** $bababa$; **г)** bab; **д)** baa.

Ясно как построить ДКА, вычисляющий x_j, на каждом шаге обработки t. То есть, состояние автомата после обработки префикса текста $t[0,j]$ соответствует искомому суффиксу x_j.

Определение 3. КМП-автоматом $\mathcal{A}_w$ для слова $w \in \Sigma^*$ длины n называется ДКА, заданный набором $(Q,\ \Sigma,\ q_0,\ \delta,\ F)$, где

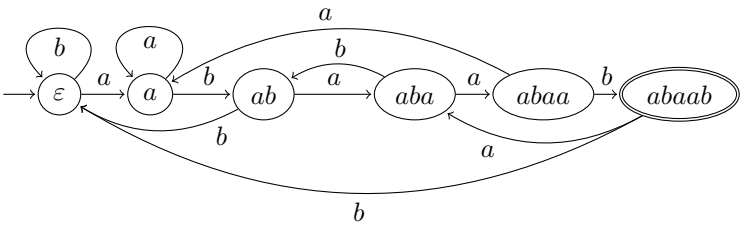
- $Q = \{\ w[0,0],\ w[0,1],\ w[0,2], \ldots,\ w[0,n]\ \}$;
- $q_0 = w[0,0]$;
- $\delta(w[0,k],a) = \begin{cases} w[0,k+1] & \text{при } w[0,k+1] = w[0,k]a \text{ и } k < n; \\ l(w[0,k]a) & \text{при } w[0,k+1] \neq w[0,k]a \text{ и } k < n; \\ l(w[0,n]a) & \text{при } k = n. \end{cases}$

- $F = \{w[0, n]\}$.

Этот автомат легко построить за $O(|w|^2)$ после чего обработка текста займёт $O(|t|)$ и суммарно весь алгоритм будет работать за $O(|w|^2 + |t|)$. Это уже лучше чем алгоритм на основе НКА, но всё ещё не линейное время. Алгоритм Кнута-Морриса-Пратта позволяет построить КМП-автомат за линейное время!

Контрольный вопрос 26. 19. Построить КМП-автомат, распознающий язык $\Sigma^* abaab$.

Решение.



□

4.1.2 КМП-алгоритм

Перед тем как перейти к изложению КМП-алгоритма обратим внимание на следующую проблему, с которой этот алгоритм также успешно справляется. Оценки на время работы алгоритмов в этом разделе были даны исходя из того, что размер алфавита фиксирован. Не смотря на то, что на практике это так, всё равно требуется вычислить $|\Sigma|$ переходов для каждого состояния, что делает автомат громоздким: на $|w| + 1$ состояние приходится $|\Sigma| \times (|w| + 1)$ переходов в таблице. В результате КМП-алгоритма получается сжатое описание автомата, в котором вместо обычных переходов используются переходы специального вида — суффиксные ссылки.

Суффиксные ссылки — это частный случай ссылок-исключений, к определению которых мы и переходим. Автомат переходит по ссылке-исключению, когда не определён переход по обрабатываемому символу (т.е. произошло исключение). Ссылки задаются функцией exc, которая ставит в соответствие состоянию q состояние p, в которое автомат переходит при обработке символа a из q, если $\delta(q, a) = \varnothing$. При этом, символ a, по которому произошло исключение, не считывается автоматом и, попав в состояние p, автомат либо переходит из него дальше по a, если такой переход определён, либо дальше переходит по ссылке-исключению, если такой переход не определён.

Программисты справедливо отметят в таком определении техническую проблему: если переход по a не определён ни из одного состояния на пути по ссылкам-исключениям, то автомат никогда не остановится. Но этот недостаток легко исправить на этапе построения автомата, поэтому мы не будем уделять этой проблеме внимание.

Определение 4. Назовём ДКА с ссылками-исключениями автомат $\mathcal{A}^{\mathsf{exc}}$, заданный набором $(Q,\ \Sigma,\ q_0,\ \delta, \mathsf{exc},\ F)$, в котором все компоненты, кроме exc такие же как и у обычного ДКА, а exc — частично определённая функция из Q в Q.

Ход автомата $\mathcal{A}^{\mathsf{exc}}$ на слове w определим через отношение $\vdash$ на множестве конфигураций $Q \times \Sigma^*$ — парах из состояния и необработанной части слова:

- $(q, aw) \vdash (p, w)$, если $\delta(q, a) = p$;
- $(q, aw) \vdash (p, aw)$, если $\delta(q, a) = \varnothing$ и $\mathsf{exc}(q) = p$.

Автомат $\mathcal{A}^{\mathsf{exc}}$ принимает слово w, если из начальной конфигурации достижима принимающая: $(q_0, w) \vdash^* (q_f, \varepsilon)$, $q_f \in F$. □

Ясно, что для каждого ДКА $\mathcal{A}^{\mathsf{exc}}$ с ссылками исключениями существует эквивалентный ДКА $\mathcal{A}$ без ссылок. ДКА $\mathcal{A}$ будет отличаться от $\mathcal{A}^{\mathsf{exc}}$ только изменением функции переходов,

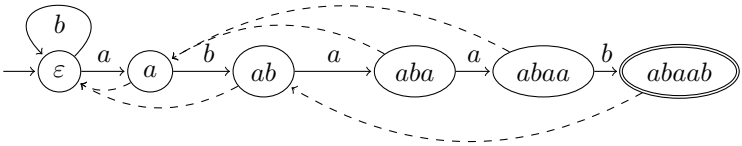


Рис. 16. КМП-автомат с суффиксными ссылками (ссылками-исключениями)

к которой будут добавлены переходы, реализуемые ссылками-исключениями.

Эти переходы определяются следующим образом. Если переход из q по a не определён, но определена ссылка-исключение, то для вычисления $\delta_{\mathcal{A}}(q, a)$ нужно пройти по цепочке ссылок-исключений из состояния q до первого состояния p, из которого будет определён переход по a, ведущий, например, в состояние r, после чего положить $\delta_{\mathcal{A}}(q, a) = s$. Если же функция $\mathsf{exc}(q)$ не определена или для каждого состояния p, достижимого из q по цепочке ссылок, нет перехода по a, то $\delta_{\mathcal{A}}(q, a) = \varnothing$.

Контрольный вопрос 27. Докажите, что ДКА $\mathcal{A}$, построенный описанным выше образом по ДКА с ссылками-исключениями $\mathcal{A}^{\mathsf{exc}}$ распознаёт язык $L(\mathcal{A}^{\mathsf{exc}})$.

Приведём в качестве примера ДКА $\mathcal{A}^{\mathsf{exc}}_{abaab}$, эквивалентный КМП автомату $\mathcal{A}_{abaab}$ на рисунке 16. Переходы по ссылкам-исключениям на нём изображены пунктиром На данном примере не видно особого выигрыша в размере описания автомата, однако если бы обычный ДКА был над троичным алфавитом, то его граф был бы совсем громоздким.

Контрольный вопрос 28. Проверьте, что ДКА $\mathcal{A}^{\mathsf{exc}}_{abaab}$ и ДКА $\mathcal{A}_{abaab}$ действительно эквивалентны.

Контрольный вопрос 29. Какой алгоритм построения ссылок-исключений для автомата $\mathcal{A}_w^{\mathsf{exc}}$? Попробуйте сообразить, прежде чем читать дальше.

При построении переходов детерминированного КМП-автомата мы использовали префикс функцию: $w[0,i] \xrightarrow{a} l(w[0,i]a) = w[0,k]$ при $w[i+1] \neq a$. Заметим, что $w[0,k-1]$ является суффиксом слова $w[0,i]$, который одновременно является его же префиксом. Обозначим через $l^m(u)$ слово, получающееся последовательным применением функции l к слову u, т.е.

$$l(u) = \underbrace{l(l(\ldots l}_{m}(u))).$$

Если $w[0,k-1] \neq l(w[0,i])$, то у слова $w[0,i]$ есть префикс $w[0,k'-1]$, являющийся одновременно и его суффиксом, причём $i > k'-1 > k-1$. Но отсюда ясно, что $w[0,k-1]$ является одновременно префиксом и суффиксом слова $w[0,k'-1]$. Поскольку длина $l(u)$ всегда меньше u. Значит, $w[0,k-1] = l^m(w[0,i])$. Этими рассуждениями мы доказали следующее утверждение.

Утверждение 3. *Если $va = l(ua)$, то $v = l^m(u)$ для некоторого $m \geqslant 1$. При этом, если $u[0,j] = l^k(u)$, $k < m$, то $u[j+1] \neq a$.*

Из этого утверждения ясно как построить КМП-автомат $\mathcal{A}_w^{\mathsf{exc}}$ с ссылками-исключения. Как и у автомата $\mathcal{A}_w$, множество его состояний — префиксы w, начальное состояние ε, единственное принимающее — w. Обычные переходы имеют вид

$$w[0,i] \xrightarrow{w[i+1]} w[0,i+1] \quad \text{при } i \geqslant 0,$$

а также из ε есть все переходы в себя по всем символам алфавита, отличным от $w[1]$. Переходы по ссылкам-исключениям определены через префикс функцию: $\mathsf{exc}(u) = l(u)$ при $u \neq \varepsilon$.

Схожие ссылки используются и в алгоритме Ахо-Корасик, которые принято называть суффиксными ссылками. Поскольку

КМП-автоматы (со ссылками и без) являются частным случаем автоматов Ахо-Корасик, то ссылки-исключения в КМП-автомате мы также будем называть суффиксными ссылками.

Итак, КМП-алгоритм сводится к построению КМП-автомата $\mathcal{A}_w^{\mathsf{exc}}$ по образцу w и прогону через него текста t. Приведём алгоритм построения автомата за линейное время и покажем, что прогон текста также осуществим за линейное время.

Фактически нужно описать вычисление ссылок-исключений или что то же самое для КМП-автомата, префикс-функции для всех префиксов w.

Алгоритм. Будем последовательно вычислять $l(w[0,i])$:

i=1. $l(w[0,1]) = \varepsilon$.

i>1. Пусть $w[0,i] = w[0,i-1]a$. Будем искать среди уже вычисленных значений $l^m(w[0,i-1])$ первое (в порядке возрастания $m \geqslant 1$), которое попадёт хотя бы под одно из условий:

- $l^m(w[0,i-1]) = w[0,k]$ и $w[k+1] = a$;
- $l^m(w[0,i-1]) = \varepsilon$.

Если произошли оба события, то алгоритм поступает как если бы произошло только первое: возвращает $w[0,k+1]$. Если же произошло второе (но не первое), то алгоритм возвращает $l(w[0,i]) = \varepsilon$.

Замечание 4. *Переход от $l^m(w[0,i-1])$ к $l^{m+1}(w[0,i-1])$ в процессе перебора соответствует переходу по уже вычисленной ссылке в графе автомата.*

Корректность. Во-первых, алгоритм возвращает некоторое значение для каждого i, потому что хотя бы одно из событий обязательно произойдёт: каждая последовательность $l^m(u)$ заканчивается пустым словом. Если произошло первое событие, то $w[0,k+1] = l(w[0,i])$ по утверждению 3. Из того же утверждения получаем, что первое событие всегда наступает при $l(w[0,i]) \neq \varepsilon$, а значит при наступлении только второго события $l(w[0,i]) = \varepsilon$.

Время работы алгоритма. На первый взгляд кажется, что этот алгоритм квадратичен, но более детальный анализ показывает, что он на самом деле линейный.

Воспользуемся амортизационным анализом; идея этого подхода неформально состоит в следующем. Представим, что у нас есть счёт в банке, и в начале работы алгоритма банк пуст. За вычисление $l(w[0,i])$ мы получаем по рублю для каждого i и можем эти деньги тратить на последующие вычисления. Ясно, что если мы вычислили $l(w[0,i])$ по приведённому алгоритму для каждого i и ни разу не оказались в долгах, то алгоритм линейный.

В ходе работы алгоритма поддерживается следующий инвариант.

Утверждение 4. *Длина $l(w[0,i])$ не превосходит количество рублей $\$_i$ на i-ом шаге.*

Доказательство. Докажем утверждение индукцией по i. Для $i = 1$ утверждение очевидно: $l(w[1]) = \varepsilon$ и в запасе есть целый рубль! Пусть утверждение верно для $l(w[0,i])$. При вычислении $l(w[0,i+1])$ произойдёт не более чем $|l(w[0,i])| \leqslant \$_i$ переходов по ссылкам из $l(w[0,i])$, поскольку каждый переход уменьшает длину хотя бы на 1. В результате вычисления окажется, что либо $l(w[0,i+1]) = \varepsilon$ и тогда с бюджетом всё в порядке, либо $l(w[0,i+1]) = w[0,k+1]$, причём в бюджете ещё остаётся k рублей, а $k+1$-ый рубль нам заплатят за вычисление $l(w[0,i+1])$. □

Если рассуждения с бюджетом кажутся слишком мудрёными, то приведём неравенство из книги А. Шеня [4], в которой КМП-алгоритм описан с программисткой точки зрения:

$$|l(w[0,i+1])| \leqslant |l(w[0,i])| - (\#\text{ число итераций на } i\text{-ом шаге»}) + 1$$

и предложим (как и в [4]) перенести число итераций в левую часть неравенства (а остальное в правую) и сложить получившиеся неравенства по всем i.

Те же рассуждения (с бюджетом или сложением неравенств) проходят и для доказательства линейного времени работы КМП-автомата $\mathcal{A}_w^{\mathsf{exc}}$ при обработке текста t.

4.2 Алгоритм Ахо-Корасик

Алгоритм Ахо-Корасик обобщает алгоритм Кнута-Морриса-Пратта на случай множества образцов. Он назван в честь Альфреда Ахо и Маргарет Корасик, поэтому склонениям не поддаётся (никаких Ахов-Корасиков!).

Для работы с множеством алгоритм использует структуру данных «Словарь», а точнее её автоматную реализацию.

4.2.1 Структура данных «Словарь»

Под словарём понимают структуру данных, с помощью которой можно проверять вхождение слова в множество. У этой структуры данных есть следующие операции:

- **in**: добавить слово x в словарь;
- **test**: проверить, входит ли слово x в словарь;
- **out**: удалить слово x из словаря.

Словарь можно изящно реализовать на основе детерминированных конечных автоматов. На рис. 17 показан пример словаря с содержимым $S = \{a, ba, ab, abb, abab\}$. ДКА $\mathcal{A}_S$ принимает слово x тогда и только тогда, когда $x \in S$.

Задача 20. Постройте алгоритмы, реализующие операции **in**, **out** и **test** для ДКА, реализующих словарь. В результате этих операций должен получиться ДКА, который реализует словарь с соответственно изменившимся содержимым.

Решение. Начнём с реализации **out**. Алгоритм подаёт на вход автомату $\mathcal{A}_S$ слово x. Если автомат не закончил обработку слова, поскольку не все префиксы x являются состояниями $\mathcal{A}_S$, то

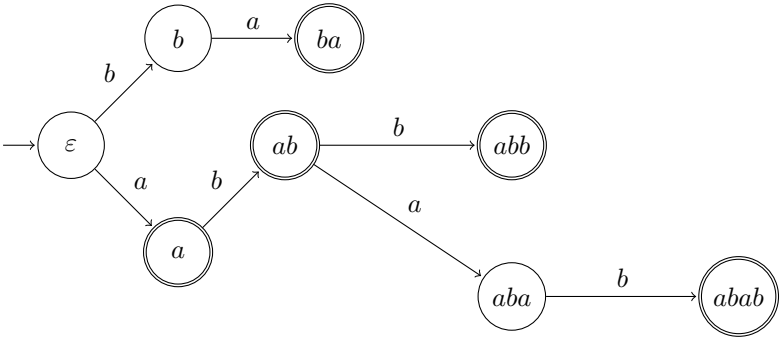


Рис. 17. ДКА $\mathcal{A}_S$ реализует словарь

алгоритм заканчивает работу. Если же слово обработано, то достигнуто состояние x, которое алгоритм делает непринимающим.

При выполнении операции **in** по слову x алгоритм обрабатывает x и добавляет недостающие префиксы x к состояниям автомата вместе с переходами $x[0,i] \xrightarrow{x[i+1]} x[0,i+1]$. После чего делает состояние x принимающим.

При выполнении операции **test**, алгоритм проверяет принимает ли ДКА слово x и возвращает соответствующий результат.

Корректность данных алгоритмов очевидна. После каждой операции сохраняется свойство $w \in S$ тогда и только тогда когда w — принимающее состояние автомата $\mathcal{A}_S$. □

Контрольный вопрос 30. Проверьте, что каждый приведённый выше алгоритм работает за линейное время по $|x|$ для каждой из операций.

4.2.2 Автомат Ахо-Корасик

Алгоритм Ахо-Корасик проверяет вхождение в текст t слов из заранее построенного словаря. Причём алгоритм находит все вхождения за линейное время!

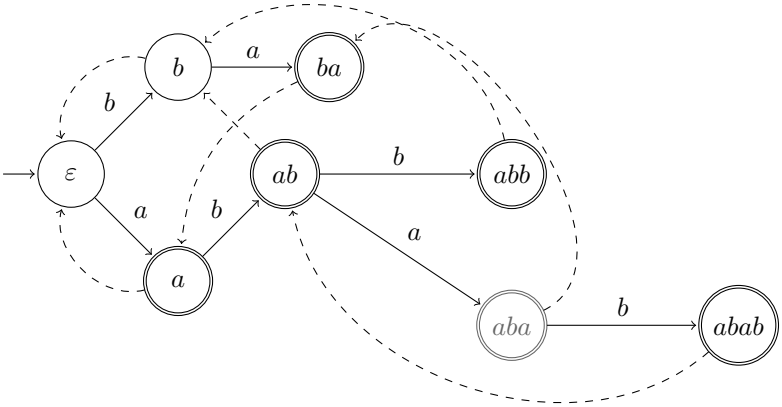


Рис. 18. Автомат Ахо-Корасик $\mathcal{A}_S^{\mathsf{exc}}$

Автомат для алгоритма Ахо-Корасик $\mathcal{A}_S^{\mathsf{exc}}$ (рис. 18) — это ДКА с сылками-исключениями, построенный следующим образом. Он содержит все состояния и переходы, что и соответсвующий автомат для словаря (рис. 17), но помимо этого к принимающим состояниям относятся те состояния, некоторый суффикс которых является принимающим: состояние aba выделено серым, поскольку слово aba не лежит в словаре, однако в словаре лежит слово ba. Пунктирные переходы как и в КМП-автомате — это пе-

реходы по ссылкам-исключениям. Переход по ссылке из состояния u добавляется согласно следующему правилу. Слово s, в которое ведёт переход, должно быть самым длинным суффиксом слова u, отличным от u, который является состоянием автомата Ахо-Корасик. Такие переходы называют *суффиксными ссылками*. Аналогично КМП-автомату, добавляются также переходы $\varepsilon \xrightarrow{a} \varepsilon$ для всех символов $a \in \Sigma$, которые не являются состояниями словаря.

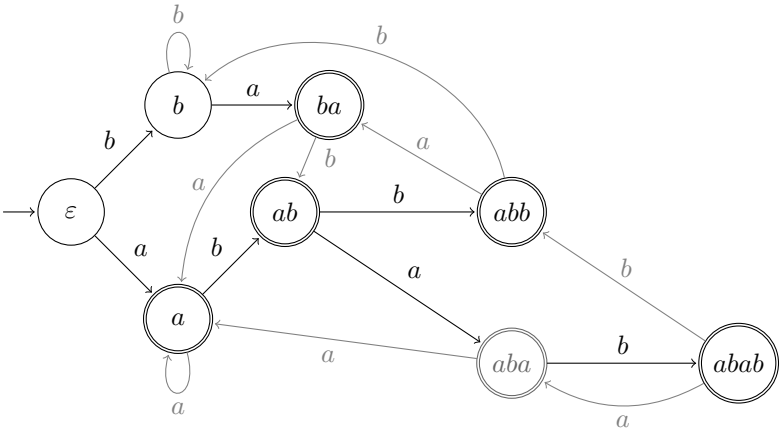


Рис. 19. ДКА Ахо-Корасик $\mathcal{A}_S$

На рис. 19 приведён ДКА Ахо-Корасик $\mathcal{A}_S$, полученный из автомата $\mathcal{A}_S^{\text{exc}}$ (рис. 18) заменой суффиксных ссылок на явные переходы. ДКА $\mathcal{A}_S$ оказывается в принимающем состоянии тогда и только тогда, когда некоторый суффикс обработанной части текста $t[0, j]$ является словом из словаря (таких слов может быть несколько одновременно). Заметим также, что количество одновременно встретившихся подслов текста из словаря зависит только от состояния автомата.

Как и в случае КМП-автомата, построение ДКА Ахо-Корасик без ссылок-исключений, как правило, неэффективно для приложений — ссылки позволяют существенно сократить таблицу переходов и уменьшить тем самым длину описания автомата.

Итак, алгоритм Ахо-Корасик сводится к построению словаря (выше было показано, что это занимает линейное время), построению суффиксных ссылок (мы покажем дальше, что их также можно добавить за линейное время) и к прогону получившегося автомата на тексте (и опять, линейное время там будет по тем же причинам, что и при добавлении суффиксных ссылок). Начнём с доказательства корректности общей конструкции.

Утверждение 5. *Автомат $\mathcal{A}_S^{\mathsf{exc}}$ оказывается в состоянии s тогда и только тогда, когда s — самый длинный из суффиксов обработанной части текста $t[0, j]$.*

Доказательство. Проведём доказательство индукцией по j. При $j = 0$ рассуждение очевидно, докажем индукционный переход. Если у автомата есть состояние $st[j+1]$, то произойдёт обычный переход и состояние $st[j+1]$ будет самым длинным суффиксом $t[0, j+1]$ среди всех состояний. Иначе, если есть более длинный суффикс $s't[j+1]$, то отсюда получаем, что слово s' длиннее s и также суффикс $t[0, j]$, что невозможно.

Если же в автомате нет состояния $st[j+1]$, то начнутся переходы по суффиксным ссылкам. Если ни один из суффиксов $st[j+1]$ не встречается среди состояний автомата, то эти переходы приведут в состояние ε, в котором автомат и останется после перехода по $t[j+1]$. Если же $s't[j+1]$ — самый длинный суффикс $st[j+1]$, среди состояний автомата, то s' является суффиксом s, а значит он достижим из s по суффиксным ссылкам. При этом s' — это самый длинный суффикс, из которого есть переход по символу $t[j+1]$, значит на нём переходы по суффиксным ссылкам и закончатся, по определению ссылок-исключений (которыми последние и являются). □

Для каждого состояния можно проверить (путешествуя по

суффиксным ссылкам) какие суффиксы состояния содержатся в словаре. И сохранить эту информацию в таблице, если она нужна, например, для подсчёта всех вхождений слов из словаря в текст (считаем, что вхождения могут пересекаться).

В доказательства утверждения 5 мы фактически доказали корректность следующего рекурсивного алгоритма добавления суффиксных ссылок.

Алгоритм. Для добавления ссылки из состояния ua нужно идти по суффиксным ссылкам из u до первого состояния s, из которого определён переход по a, после чего добавить ссылку в sa. Считаем, что для вычисления суффиксных ссылок из состояний $a \in \Sigma$ переходов по ссылкам не происходит (известно, что они ведут в ε).

Докажем теперь, что вычисление суффиксных ссылок в порядке обхода ширину от ε и есть линейный алгоритм добавления суффиксных ссылок.

Теорема 1. *Вычисление суффиксных ссылок в порядке увеличения* $|u|, u \in S$ *занимает* $O\left(\sum_{u \in S} |u|\right)$.

Доказательство. Также воспользуемся амортизационным анализом. Но теперь банковский счёт будет у каждого состояния — вершины графа автомата словаря. Средства будут начисляться вершинам при их открытии поиском в ширину (по уровням) после открытия со счёта каждой вершины будут списываться средства соразмерные числу операций для вычисления исходящей из неё суффиксной ссылки.

Итак, учёт средств организован следующим образом.

i. На счету вершины ε нет средств: $\$_\varepsilon = 0$.

ii. Если у вершины u осталось $\$_u$ рублей после вычисления ссылки, то её средства переходят к первой открытой дочерней вершине ua, которая в дополнение к наследству получает при открытии рубль; в итоге $\$_{ua} = \$_u + 1$.

iii. Если у вершины u больше одного ребёнка, то каждая следующая вершина ub получает при открытии $|ub|$ рублей.

iv. Вычисление суффиксных ссылок у дочерних вершин u начинается с вершины s, в которую ведёт суффиксная ссылка из u. За каждый переход по суффиксной ссылки, начиная с перехода из s со счёта вершины $u\sigma$ списывается рубль.

Для доказательства заявленной оценки достаточно доказать следующие утверждения:

1. Вершинам было выдано сумарно $\sum\limits_{u \in S} |u|$ рублей.
   (Обмен средств между вершинами тут не учитывается!)

2. В процессе вычислений с каждого счёта списывается не больше средств, чем на нём было: на каждом шаге $\$_u \geqslant 0$ для каждой вершины u.

Их доказательства вынесены отдельно. □

Начнём с доказательства утверждения, равносильного первому.

Утверждение 6. *Если провести только начисления и не учитывать траты, то на счету каждого листа z окажется $|z|$ рублей. Поскольку каждый лист и есть слово из словаря, то суммарно вершинам и будет выдана указанная сумма.*

Доказательство. Докажем индукцией по глубине дерева n, что после начислений средств всем вершинам уровня n, на счёту каждой вершины u этого этого уровня будет n рублей. База для $n = 0$ очевидно верна, согласно правилу i; докажем переход. Если у вершины u с уровня $n - 1$ лишь один потомок ua, то он унаследует $n - 1$ рубль у вершины u и получит ещё один на счёт, итого: $\$_{ua} = n$. В случае, если потомка больше одного, то каждый потомок ub получит n рублей согласно правилу *iii*. □

Для доказательства второго утверждения введём вспомогательное понятие. Назовём *суфиксной глубиной* $dl(v)$ вершины v количество переходов на пути из v в ε по суффиксным ссылкам.

Утверждение 7. *Для функции dl справедливы свойства:*

1) $dl(v) \leqslant |v|$

2) $dl(ub) \leqslant dl(u) + 1$

Доказательство. Первое свойство очевидно: каждый переход по суффиксной ссылке приводит к слову меньшей длины.

Второе неравенство получается из следующего факта. Пусть $dl(ub) = n$. Зафиксируем последовательность

$$s_1 b = \mathsf{exc}(ub),\ s_2 b = \mathsf{exc}(s_1 b),\ \ldots\ , s_{n-1} b = \mathsf{exc}(s_{n-2} b)$$

($\mathsf{exc}(s_{n-1} b) = \varepsilon$). В этой последовательности s_i — это некоторый суффикс u, который является одновременно суффиксом s_{i-1} и встречается среди состояний автомата. Значит каждой (кроме быть может одной) вершине пути по ссылкам из ub в ε соответствует вершина пути по ссылкам из u в ε, что и завершает доказательство. □

Докажем теперь второе утверждение: у каждой вершины u на счету хватит средств, чтобы добраться из неё по суффиксным ссылкам до корня (после начислений средств и их затрат на вычисления). Или его формальный вариант:

Утверждение 8. *Для каждой вершины u по завершению вычисления идущей из неё суффиксной ссылки выполняется неравенство*

$$\$_u \geqslant dl(u).$$

Доказательство. Докажем утверждение индукцией по $|u|$.

База (для $|u| = 1$) тривиальна — и так известно, что суффиксные ссылки из состояний $a \in \Sigma$ ведут в ε.

Докажем переход. Если деньги были начислены вершине ub по правилу iii, то на её счету (до вычисления ссылки) оказалось $|ub|$ рублей. Если ссылка из ub ведёт в s, то алгоритм совершил не более $|u| - |s|$ переходов по ссылкам из u, поэтому $\$_{ub} \geqslant |s| + 1$ и этих средств заведомо хватит, поскольку $dl(s) \leqslant |s|$ по утверждению 7, а значит

$$\$_{ub} \geqslant |s| + 1 \geqslant dl(s) + 1 = dl(ub).$$

Сосредоточимся на правиле ii. По предложению индукции $\$_u \geqslant dl(u)$ (после вычисления ссылки из u), а значит перед вычислениями ссылки из вершины ua, на её счету не меньше $dl(u) + 1$ рубля. Но $dl(ua) \leqslant dl(u) + 1$ по второму свойству из утверждения 7. □

Итак, с помощью амортизационного анализа мы доказали, что построение автомата Ахо-Корасик с суффиксными ссылками реализуется за линейное время. По тем же соображениям линейное время занимает и обработка входного слова: с ростом глубины текущего состояния увеличивается его банковский счёт: за счёт длины обработанного префикса текста $\$_u \geqslant |u|$. А мы уже установили, что на путешествия по суффиксным ссылкам из u тратится не больше $|u|$ рублей.

4.3 Задачи

Задача 21. Постройте ДКА для словаря $\{ac, acb, b, ba, c, cbb\}$. Добавьте в полученный словарь слово ab и удалите слово ac.

Задача 22. Постройте для словаря $S = \{ac, acb, b, ba, c, cbb\}$ (который вы строили в предыдущем разделе) автомат Ахо-Корасик. Посчитайте с его помощью количество различных вхождений слов из словаря S в подслово $acbacbb$.

5 Структурные свойства регулярных языков

Структурные свойства класса формальных языков часто используют не только для того, чтобы изучить структуру класса, но и как инструмент доказательства непринадлежности языку классу. До сих пор, изучая регулярные языки, мы не приводили ни одного примера нерегулярного, хотя и доказали неконструктивно, что они есть.

Мы начнём с доказательства «руками» нерегулярности языка. Техника этого примера легко обобщается до леммы о накачке — необходимого условия регулярности, которое часто используют для доказательства нерегулярности. Дальше мы изучим алгоритм минимизации ДКА: с его помощью можно проверить, эквивалентны ли два регулярых языка, заданных ДКА. Наличие алгоритма в этом разделе может показаться странным, но этому есть достойное объяснение: алгоритм минимизации напрямую вытекает из критерия регулярности языка — теоремы Майхилла-Нероуда, изложением которой мы и завершим этот раздел.

5.1 Ad-hoc рассуждение

Конечные автоматы — это устройства с конечной памятью, поэтому ясно, что они не могут распознать язык $\{a^n b^n \mid n \geqslant 0\}$, потому что для его распознования нужно помнить точное количество букв a идущих перед b, сколь бы большим оно ни было. Этот наблюдение можно преобразовать в честное доказательство следующим образом.

Докажем от противного. Допустим, что существует полный ДКА $\mathcal{A}$ с p состояниями, распознающий язык $\{a^n b^n \mid n \geqslant 0\}$. Пусть

$$q_0 \xrightarrow{a} q_1,\ q_0 \xrightarrow{a^2} q_2,\ \ldots,\ q_0 \xrightarrow{a^{p+1}} q_{p+1}.$$

Поскольку $\mathcal{A}$ — полный детерминированный автомат, то все

состояния q_i однозначно определены, а поскольку у ДКА $\mathcal{A}$ есть лишь p состояний, то $q_i = q_j$ при $i \neq j$ (по принципу Дирихле: среди $p+1$-го состояния обязательно хотя бы два совпадут). Заметим, что у ДКА $\mathcal{A}$ есть два следующих успешных пути вычислений:

$$q_0 \xrightarrow{a^i} q_i \xrightarrow{b^i} q_f \quad \text{и} \quad q_0 \xrightarrow{a^j} q_j \xrightarrow{b^j} q'_f,$$

где q_f и q'_f — принимающие состояния. Но раз $q_i = q_j$, то у $\mathcal{A}$ есть и такой путь вычисления:

$$q_0 \xrightarrow{a^i} q_i \xrightarrow{b^j} q'_f,$$

то есть $a^i b^j \in L(\mathcal{A}) = \{a^n b^n \mid n \geqslant 0\}$, что неверно.

Приведённое рассуждение обобщается заменой слов a_i и b_i на произвольные последовательности x_i, y_i с «диагональным свойством»:

Лемма 1. *Пусть $x_i, y_i, 1 \leqslant i \leqslant p$ — последовательности слов, такие что $x_i y_j \in L$ тогда и только тогда, когда $i = j$. Тогда, если ДКА $\mathcal{A}$ распознаёт язык L, то у него не меньше p состояний.*

Задача 23. Докажите это утверждение.

Задача 24. Докажите, что утверждение верно не только для ДКА, но и для НКА.

Ясно, что если существует бесконечная последовательность x_i, y_i с диагональным свойством ($x_i y_j \in L \iff i = j$), то L — не регулярный язык.

Последовательности x_i, y_i называют «fooling set», а технику из содержания леммы 1 «fooling set technique». Мы не будем переводить это выражение буквально, дабы не получилось также нелепо как с леммой о накачке[12], а будем называть лемму 1 диагональной леммой.

[12] Я думал, что это название нельзя испортить ещё больше, пока один китайский студент ни назвал её «теоремой о каначке».

5.2 Лемма о накачке

Лемма о накачке (Pumping Lemma) известна также как лемма о разрастании. Эта лемма также легко вытекает из приведённого выше ad-hoc рассуждения. Основная идея доказательства: если слово w из регулярного языка L длинее, чем число состояний распознающего его ДКА, то при обработке w встречаются два одинаковых состояния: то есть, $w = xyz$, где $q \xrightarrow{y} q$. А значит слово y можно повторять или исключить и опять получить слово из L: $xy^iz \in L$ при $i \geqslant 0$.

Надеемся, что идея доказательства поможет в понимании несколько запутанной формулировке леммы.

Лемма 2. *Для любого регулярного языка L существует такая константа $p \geqslant 1$, что для любого слова w из L длиннее p, существует разбиение $w = xyz$, такое что*

- $|y| \geqslant 1$
- $|xy| \leqslant p$
- $\forall i \geqslant 0$, $xy^iz \in L$.

Доказательство. Дополним идею доказательства перед леммой до доказательства. Пусть у ДКА $\mathcal{A}$, распознающего L, $p-1$ состояние. Тогда при обработке некоторого префикса u слова $w = uz$ не длинее p, автомат дважды окажется в состоянии q, обработав между посещениями состояний непустое слово y, т.е. $u = xy$.

Итак, мы доказали, что для каждого слова $w \in L$ длинее p существует разбиение с указанными свойствами. □

Контрольный вопрос 31. Докажите нерегулярность языка $L = \{a^nb^n \mid n \geqslant 0\}$ используя лемму о накачке.

Решение. Докажем от противного, допустив, что L — регулярный язык.

Воспользуемся леммой о накачке. Рассмотрим $w = a^pb^p$, где p — константа леммы. Тогда существует указанное разбиение и

раз $|xy| \leqslant p$, то $xy = a^m$ и $y = a^k$, где $m \geqslant k > 0$. Взяв $i = 0$ получаем, что слово $a^{p-k}b^p$ также должно принадлежать L, что невозможно. □

Лемма о накачке обычно непросто даётся студентам, потому что для доказательства от противного приходится применять её отрицание, а замена кванторов в этом случае даётся нелегко.

Контрольный вопрос 32. Сформулируйте отрицание условия леммы о накачке.

К тому же, студенты часто не понимают, что слово w стоит под квантором для p.

Замечание 5. *Для доказательства того, что некоторый язык является нерегулярным, при использовании леммы о накачке необходимо предъявить не одно конкретное слово, а последовательность слов, зависящих от p (константы леммы). Если вы предъявите всего одно слово, то кто-то очень умный сможет просто взять бóльшую константу.*

Также часто возникает следующие недоумение:

Контрольный вопрос 33. Справедлива ли лемма о накачке для конечных языков? Ведь xy^iz — бесконечная последовательность слов из L!

У этой леммы слишком много недостатков. Во-первых, она не всегда применима: утверждение леммы справедливо для некоторых нерегулярных языков.

Задача 25. Покажите, что следующий язык удовлетворяет условиям леммы о накачке, но сам регулярным не является:

$$L = \{ab^{2^i} \mid i \geqslant 0\} \cup \{b^j \mid j \geqslant 0\} \cup \{a^m b^n \mid m > 1, n \geqslant 0\}.$$

Во-вторых, неаккуратное применение леммы влечёт громоздкие выкладки.

Пример 1. Покажем с помощью леммы о накачке, что язык $L = \{a^n b^n \mid n \geqslant 0\}$ нерегулярный, выбрав менее удачное слово.

Рассмотрим слово: $w = a^r b^r$, $|w| > p$. Если обещанное разбиение существует, то y имеет вид a^k или b^k, $k \geqslant 1$ – в противном случае, если $y = a^k b^l$, то $y^2 = a^k b^l a^k b^l$, но в L нет слов, в которых за b следует a. Допустим, что $y = a^k$. Тогда $x = a^m$, $z = a^l b^r$, $k + m + l = r$. Но тогда, по лемме о накачке $xy^2z \in L$, а значит, слово $a^{m+2k+l} b^l \in L$, но $m + 2k + l > r$, т.к. $m + k + l = r$ и $k > a$, поскольку $|y| \geqslant b$. Аналогично приходим к противоречию когда $y = b^k$. □

Даже для такого простого примера неподкованному в этой науке человеку потребуется применить много усилий, а в более сложных случаях перебор возможных y может оказаться ещё более громоздким. Обычно лемма о накачке работает в тех же случаях, когда срабатывает диагональная лемма. Тем не менее, у этой леммы есть очень важный плюс — учебный. Во-первых, лемма о накачке показывает структуру регулярного языка: разность длин двух последовательных слов из регулярного языка ограниченна линейной функцией. Во-вторых, существует ещё лемма о накачке для контекстно-свободных языков, для понимания которой стоит изучить более простую лемму о накачке для регулярных языков. В случае КС-языков, доказательство непринадлежности языка классу КС уже куда менее очевидно, так что лемма о накачке становится мощным и одним из основных инструментов.

Для регулярных же языков чаще проще использовать диагональную лемму или теорему Майхилла-Нероуда, с которой мы познакомимся дальше.

6 Алгоритм минимизации ДКА

Начнём с общей идеи. Допустим, что q и p — такие состояния полного ДКА $\mathcal{A}$, что при обработке слова z из q ДКА $\mathcal{A}$ оказывается в принимающем состоянии тогда и только тогда, когда он оказывается в принимающем состоянии обработав z из p, т.е.

$$\forall z \in \Sigma^* : q \xrightarrow{z} q_f, q_f \in F_{\mathcal{A}} \iff p \xrightarrow{z} p_f, p_f \in F_{\mathcal{A}}. \tag{4}$$

Интуиция подсказывает, что тогда можно избавиться, например, от p: само состояние удалить, а все ведущие в него переходы направить q. Назовём эту процедуру *склейкой* p и q. В результате склейки p и q получится автомат $\mathcal{A}'$, распознающий тот же язык, что и $\mathcal{A}$. Докажем этот факт в следующем абзаце. Состояния q и p, удовлетворяющие условию (4), мы будем называть *неразличимыми*.

Утверждение 9. $w \in L(\mathcal{A}) \iff w \in L(\mathcal{A}')$

Доказательство. Для доказательства воспользуемся следующим техническим трюком. Построим по ДКА $\mathcal{A}$ НКА $\mathcal{B}$, добавив ε-переход из p в q. Докажем, что если слово w принимается автоматом $\mathcal{A}$, то для него есть успешный ход автомата B, в котором не встречается состояние p.

Если при обработке w состояние p не встретилось, то всё в порядке, если же ход $\mathcal{A}$ на w имеет вид[13] $q_0 \xrightarrow{x} p \xrightarrow{y} q_f$ и при обработке x состояние p не встречалось, то воспользовавшись тем, что $q \xrightarrow{y} q'_f$ в силу условия неразличимости, получим, что у НКА $\mathcal{B}$ есть ход

$$q_0 \xrightarrow{x} q \xrightarrow{y} q'_f.$$

Если при обработке y из q автомат $\mathcal{A}$ не встречает p, то всё в порядке. Если же встречает, то $y = x_1 y_1$, где $q \xrightarrow{x_1} p \xrightarrow{y_1} q'_f$, но тогда $q \xrightarrow{y_1} q''_f$ и у $\mathcal{B}$ есть ход

$$q_0 \xrightarrow{x} q \xrightarrow{x_1} q \xrightarrow{y_1} q''_f.$$

[13] Здесь и далее состояния с индексом f — принимающие состояния.

Продолжая так и дальше (заменяя y_i на $x_{i+1}y_{i+1}$ и т.д.) получаем, что у НКА $\mathcal{B}$ есть успешный ход на $w \in L$, на котором состояние p не встречается. Осталось отметить, что построенный ход НКА $\mathcal{B}$ является успешным ходом ДКА $\mathcal{A}'$. Итак, мы доказали, что $L(\mathcal{A}) \subseteq L(\mathcal{A}')$.

Докажем теперь, что $L(\mathcal{A}') \subseteq L(\mathcal{A})$. Отметим, что любой успешный ход ДКА $\mathcal{A}'$ либо не отличается от хода ДКА $\mathcal{A}$ либо совпадает с ходом НКА $\mathcal{B}$, построенным выше. Но эти преобразования можно проводить и в обратную сторону: если ход $\mathcal{A}'$ (и $\mathcal{B}$) имеет вид

$$q_0 \xrightarrow{x} q \xrightarrow{x_1} q \xrightarrow{x_2} q \to \cdots \to q \xrightarrow{x_n} q \xrightarrow{y_n} q_f,$$

то отрезок $q \xrightarrow{x_n} q \xrightarrow{y_n} q_f$ можно заменить на $q \xrightarrow{x_n} p \xrightarrow{y_n} q'_f$, взять $y_{n-1} = x_n y_n$ и продолжить обратные замены, пока не будет построен успешный ход ДКА $\mathcal{A}$. □

Приведённое доказательство относительно непростое. Однако корректность процедуры минимизации можно объяснить гораздо проще, используя теорему Майхилла-Нероуда, приведённую ниже.

Контрольный вопрос 34. В процедуре склейки оставлена лакуна. Как провести склейку, если одно из состояний начальное?

Алгоритм минимизации сводится к склейке неразличимых состояний. Но чтобы найти неразличимые, проще найти различимые: именно это и делает следующий алгоритм.

Алгоритм поиска различимых состояний. На вход алгоритма подаётся ДКА $\mathcal{A}$.
1. В случае, если автомат $\mathcal{A}$ не является полным, пополним автомат $\mathcal{A}$, добавив состояние q_D, такое что $\forall \sigma \in \Sigma : q_D \xrightarrow{\sigma} q_D$, и если $\delta(q, \sigma) = \varnothing$, то теперь $\delta(q, \sigma) = q_D$. Если в $\mathcal{A}$ есть состояния, не достижимые из начального состояния, удалим их.

2. Разделим множество состояний на два подмножества: множество принимающих состояний $F = Q_1$ и его дополнение $Q \setminus F = Q_2$.

i + 1. Пусть после i-ого шага алгоритма множество состояний Q разбито на j подмножеств $Q_1, \ldots, Q_j$. Зафиксируем символ $\sigma \in \Sigma$ сделаем следующее. Если для $q_k \in Q_k$ $q_k \xrightarrow{\sigma} q_l \in Q_l$, поместим состояние q_k в множество $Q_{k,l}$. Получили новое разбиение множества Q на подмножества $Q_{k,l}$ и повторяем для него эту процедуру для оставшихся символов $\sigma \in \Sigma$. Если после $|\Sigma|$ разбиений мы получили разбиение, в котором столько подмножеств, сколько и было (j), то алгоритм останавливается, а в противном случае переходит к шагу **i + 2**.

Алгоритм минимизации состоит поиске всех пар различимых состояний приведённым выше алгоритмом, после чего неразличимые состояния (все состояния, попавшие в одно подмножество) склеиваются (согласно процедуре, описанной в начале раздела). Чтобы получить минимальный ДКА, но не полный ДКА, нужно удалить состояние из которого недостижимо ни одно принимающее — все такие состояния попали в одну группу с состоянием q_d (если оно было добавлено).

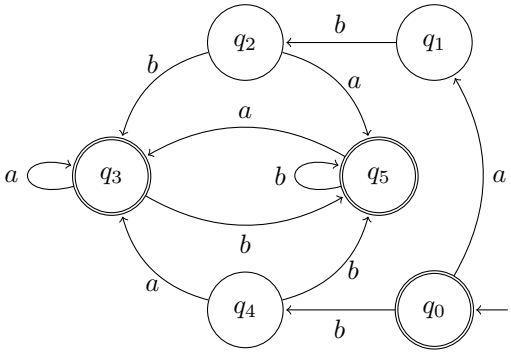
Пример 2. Минимизируем автомат $\mathcal{A}$, заданный графом на рис. 20[14].

Процесс разделения множеств можно проиллюстрировать с помощью «ящиков» (рис. 21). Каждое множество Q_i — отдельный ящик, переходы из состояний в ящиках изображены стрелочками. Состояния в одном ящике нужно разделить, если стрелки из них ведут в разные ящики (справа указана буква по которой происходит разделение).

Для наглядности на рис. 21 опущены стрелки из состояний, которые оказались одними в ящиках. Обратите внимание, что

[14]Иллюстрации: В. Алексеев.

Автомат $\mathcal{A}$, подлежащий минимизации:



Автомат $\mathcal{A}$ после пополнения:

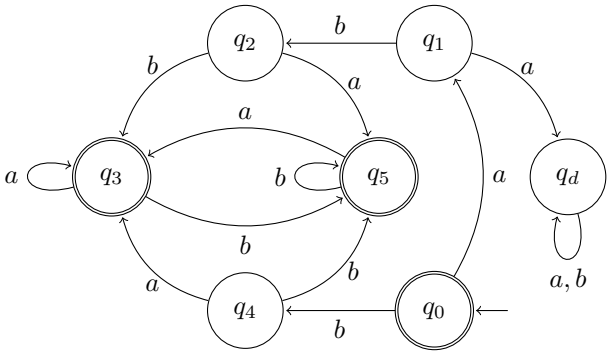


Рис. 20. Минимизация автомата: шаг 1.

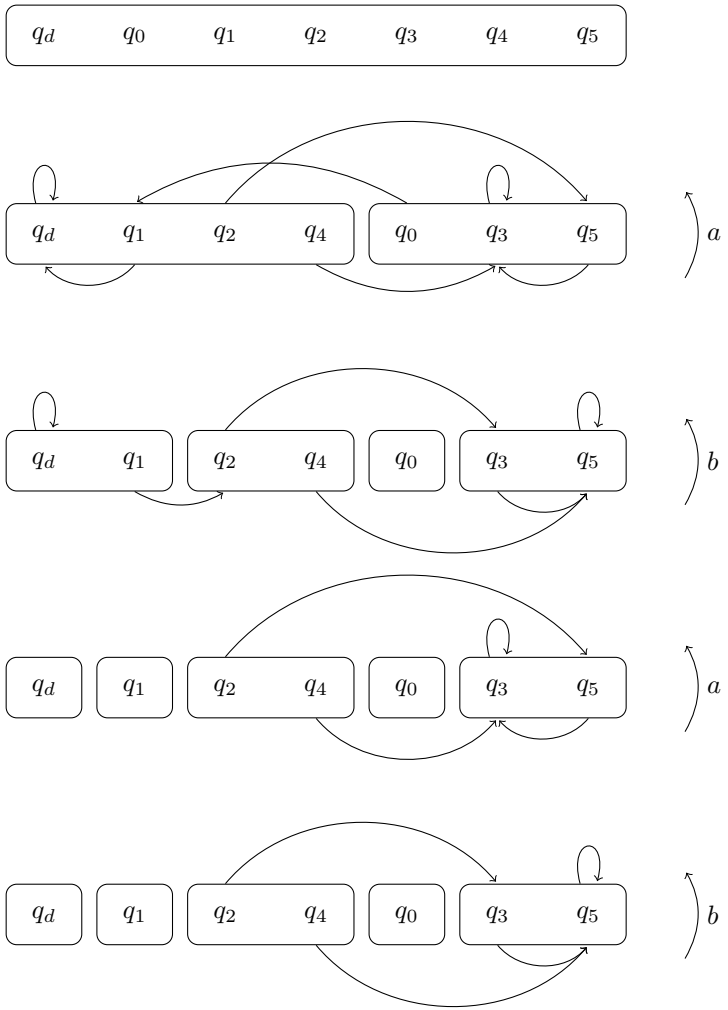
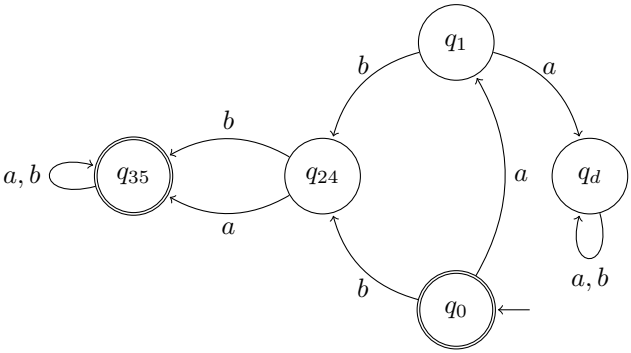


Рис. 21. Поиск различимых состояний

Автомат в результате склейки неразличимых состояний:



Автомат в результате минимизации (и удаления тупикового состояния):

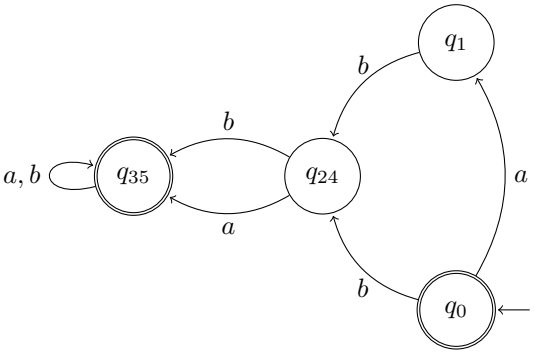


Рис. 22. Последние шаги алгоритма минимизации

алгоритм останавливается когда после последовательного прохода по всем буквам алфавита разбиение не изменяется.

Для доказательства корректности алгоритма, мы должны прежде всего признаться в обмане трудящихся.

Задача 26*. В описании алгоритма минимизации оставлен пробел (как и в некоторых учебниках, например, в учебнике Хопкрофта Мотвани и Ульмана). Приведите пример ДКА, для которого применение указанного алгоритма не приводит к минимальному автомату. Исправьте этот пробел.

Предлагаем читателю следующий эксперимент, если не получилось решить задачу 26. Поверьте на время, что алгоритм минимизации верно находит все различимые состояния. Попробуйте сами решить следующие две задачи, которые кажутся простыми, и являются промежуточными задачами в доказательстве корректности алгоритма минимизации, после чего изучите их решение в разделе с доказательством корректности и проверьте свои решения на ошибки, если решили задачи 27 и 28, не решив задачу 26.

Пока что будем называть полный автомат, построенный по алгоритму из предыдущего раздела «минимальным».

Задача 27. Докажите, что в полном ДКА $\mathcal{B}$ не может быть меньше состояний, чем в полном «минимальном» автомат $\mathcal{A}$.

Задача 28. Докажите, что минимальный автомат единственный. То есть, если полный ДКА $\mathcal{B}$ распознаёт язык L и имеет то же число состояний, что и «минимальный» ДКА $\mathcal{A}$, распознающий L, то автоматы $\mathcal{A}$ и $\mathcal{B}$ совпадают[15].

[15] Под совпадением автоматов мы понимаем, что их графы изоморфны (совпадают при наложении друг на друга)

6.1 Корректность алгоритма минимизации

В этом разделе мы заделаем пробел и докажем корректность, но будем раскрывать наши карты последовательно. Признаемся, что алгоритм действительно находит верно все различимые состояния.

Утверждение 10. *Если состояния q и p ДКА $\mathcal{A}$ различимы, то это будет установлено в результате выполнения алгоритма поиска различимых состояний — состояния q и p окажутся в разных множествах.*

Доказательство. Если состояния q и p различимы, то есть различающее слово z; пусть $\delta(q, z) \in F$, а $\delta(p, z) \notin F$. Обозначим через q_i состояние, в которое автомат $\mathcal{A}$ попадает при обработке префикса z длины i из q: $q_i = \delta(q, z[1, i])$; аналогично $p_i = \delta(p, z[1, i])$. Считаем, что $\mathcal{A}$ — полный ДКА.

Пусть $|z| = n$. Тогда q_{n-1} и p_{n-1} различимые состояния последней буквой слова z и будут найдены на первом шаге алгоритма. Но тогда q_{n-2} и p_{n-2} также различимы словом $z[n-2, n]$ и будут найдены на втором шаге. Далее обратной индукцией по i получаем, что состояния q_i и p_i различимы словом $z[n-i, n]$ на i-ом шаге алгоритма; а значит, что q и p будут найдены алгоритмом на $n+1$-ом шаге. □

Перед тем как раскрыть следующую карту, начнём с ошибочного решения задачи 27.

Решение задачи 27 с ошибкой. Доказательство от противного. Пусть у автомата $\mathcal{B}$ меньше состояний, чем у «минимального» автомата $\mathcal{A}$. Тогда есть два слова x и y, обработав которые автомат $\mathcal{A}$ приходит в разные состояния q_x и q_y, а автомат $\mathcal{B}$ после обработки каждого приходит в состояние $q_{x,y}$.

Слова x и y можно найти просто взяв пути без циклов из начального состояния автомата $\mathcal{A}$ до каждого состояния q: вдоль каждого пути написано слово w_q. Подав каждое слово w_q на вход

автомату $\mathcal{B}$, выяснится, что какие-то два слова (а может и больше) ведут в одно и то же состояние, потому что у $\mathcal{B}$ состояний на единицу меньше.

Поскольку состояния q_x и q_y различимы (по утверждению 10), то найдётся различающее их слово z. Но обработав z из состояния q_xy автомат $\mathcal{B}$ окажется либо в принимающем состоянии, либо не в принимающем; значит он примет оба слова xz и yz, но одно из них не принадлежит L. □

Пора, наконец, раскрыть все карты и указать на ошибку в алгоритме.

Решение задачи 26*. Вообще говоря в ДКА могут быть состояния, не достижимые из начального. И такие состояния и приведут к ошибке, если из них достижимо принимающее состояние. Такие состояния возникают при применении конструкции произведения: представьте, что конструкцию произведения применили к двум одинаковым автоматам. Тогда пара из принимающего и непринимающего состояния не будет достижима из пары начальных, хотя такие пары и будут.

Так в чём же ошибка в алгоритме? Если не удалить недостижимые (из начального) состояния ДКА, то они могут оказаться различимыми с состояниями из каждой оставшейся группы. Но ясно, что недостижимые состояния можно удалить, не поменяв тем самым языка. □

Патч для алгоритма минимизации: удалить вначале все состояния, недостижимые из начального. Все достижимые состояния можно найти, например, поиском в ширину по графу.

По-прежнему будем называть автомат, построенный по алгоритму (теперь уже исправленному) «минимальным».

Исправление решения задачи 27. Чтобы решение стало верным нужно добавить, что различимые состояния q_x и q_y обязательно достижимы из начального. В начале решения было неверно заявлено, что $\mathcal{A}$ попадёт в них после обработки слов x и y. □

Решение задачи 28 Доказательство от противного. Пусть $\mathcal{A}$ — полный «минимальный» автомат с n состояниями. Пусть автомат $\mathcal{B}$ также имеет n состояний (и также не имеет недостижимых состояний, иначе удалим их и воспользуемся задачей 27). Если графы $\mathcal{A}$ и $\mathcal{B}$ не совпадают, то подав на вход каждому из них все слова длины от 0 до n обнаружим, что либо два слова x и y, ведущих в одно и то же состояние автомата $\mathcal{A}$ ведут в разные состояния q_x и q_y автомата $\mathcal{B}$, либо два слова u и v, которые ведут в разные состояния q_u и q_v автомата $\mathcal{B}$, ведут в одно и то же состояние автомата $\mathcal{A}$.

Если случилось первое, то состояния q_x и q_y неразличимы: иначе, если их различает слово z, то автомат $\mathcal{A}$ должен принять только одно из слов xz, yz. Но если состояния q_x и q_y неразличимы и $\mathcal{B}$ распознаёт язык L, то склеив их, применив алгоритм минимизации к $\mathcal{B}$ получим меньшее число состояний, чем у $\mathcal{A}$. Но тогда, согласно задаче 27, получим, что автомат $\mathcal{A}$ был не «минимальным».

Если же случилось второе, то также получим, что состояния q_u и q_v неразличимы и повторим те же аргументы. □

Решив задачи и доказав утверждение 10 мы доказали следующую теорему.

Теорема 2. *Полный ДКА $\mathcal{A}$, построенный по (исправленному) алгоритму минимизации, имеет наименьшее число состояний среди полных ДКА, распознающих язык L. Если полный ДКА $\mathcal{B}$, распознающий L, имеет столько же состояний, сколько и $\mathcal{A}$, то эти автоматы совпадают.*

7 Теорема Майхилла-Нероуда

Теорема Майхилла-Нероуда — это критерий регулярности языка. Его изложение опирается на понятие отношения эквивалентности — важный частный случай бинарных отношений, поэтому мы предлагаем читателю обратиться к разделу 2.3, если свойства бинарных отношений подзабылись.

7.1 Отношения эквивалентности

Бинарное отношение $R \subseteq A \times A$ называется отношением эквивалентности, если оно рефлексивно, симметрично и транзитивно одновременно. Отношения эквивалентности постоянно встречаются в математике: отношения равенства и подобия треугольников в геометрии, равенство обыкновенных дробей в алгебре, эквивалентность формул, задающих одну и ту же функцию... Отношения эквивалентности встречаются столь часто, что со временем были повсеместно вытеснены знаком равенства, а потому и не знакомы многим читателям: до середины XX века, а то и позже, вместо термина равенства треугольников использовали термин конгуэнтность.

Как видно из названия, отношения эквивалентности определяют какие объекты эквивалентны (согласно отношению), а какие нет. Поэтому отношения эквивалентности часто обозначают символами «$\sim$» и «$\equiv$» (быть может с индексами) или и вовсе обманывают читателя, заменяя отношение эквивалентности знаком равенства.

Интуитивно ясно, что все объекты можно разбить на множества попарно эквивалентных между собой объектов — классы эквивалентности: подобные между собой треугольники образуют отдельный класс, также отдельный класс образуют обыкновенные дроби, равные $\frac{1}{2}$ (после сокращения).

Эта интуиция отражает основную теорему об отношениях эквивалентности. Чтобы её сформулировать, формализуем сначала понятие класса эквивалентности. Пусть $\sim \subseteq A \times A$ — отноше-

ние эквивалентности. Определим класс эквивалентности $[a]$ всех элементов множества A, эквивалентных элементу a:

$$[a] = \{x \mid x \in A, x \sim a\}.$$

Контрольный вопрос 35. Докажите, что два любых элемента $x, y \in [a]$ эквивалентны между собой.

Ответ на контрольный вопрос вытекает из доказательства основной теоремы.

Теорема 3. *Классы эквивалентности $[a]$ и $[b]$ (по отношению эквивалентности $\sim$) либо не пересекаются, либо совпадают. Множество A разбивается в объединение классов эквивалентности.*

Доказательство. Ясно, что

$$A = \bigcup_{a \in A} [a],$$

поскольку каждый класс $[a]$ содержит элемент a в силу рефлексивности. Докажем теперь первую часть теоремы от противного.

Допустим $x \in [a] \cap [b]$ и $[a] \neq [b]$. Раз $x \in [a]$ и $x \in [b]$ то $x \sim a$ и $x \sim b$. В силу симметричности получаем, что $a \sim x$, а по транзитивности получаем, что $a \sim b$, раз $a \sim x \sim b$. Значит $a \in [b]$ и из симметричности и транзитивности получаем, что каждый y из $[a]$ также принадлежит $[b]$:

$$y \in [a] \Rightarrow y \sim a \Rightarrow a \sim y \Rightarrow b \sim a \sim y \Rightarrow b \sim y \Rightarrow y \sim b.$$

То есть, мы показали, что $[a] \subseteq [b]$. Симметричные рассуждения показывают, что $[b] \subseteq [a]$, а значит, классы $[a]$ и $[b]$ совпадают, если перескаются. $\square$

Эта теорема объясняет, что отношение эквивалентности разбивает[16] множество A на подмножества. С другой стороны, каждому разбиению множества A соответствует отношение эквивалентности «элементы принадлежат одному множеству из разбиения».

7.2 Отношения эквивалентности Майхилла-Нероуда

Каждому языку $L \subseteq \Sigma^*$ (не обязательно регулярному!) соответствует отношение эквивалентности Майхилла-Нероуда $\sim_L$ на множестве всех слов Σ^*, которое будем называть *L-эквивалентностью*.

Определение 5. Слова x и y L-эквивалентны тогда и только тогда, когда приписывание справа к каждому слову любого z превращает оба слова одновременно либо в слова из языка L, либо в слова не из языка L:

$$x \sim_L y \iff \forall z \in \Sigma^* : (xz \in L \iff yz \in L)\,.$$

Контрольный вопрос 36. Проверьте, что $\sim_L$ действительно отношение эквивалентности.

Если слова x и y не эквивалентны, то по определению найдётся слово z, приписывание которого к каждому слову приведёт к тому, что одно получившееся слово будет лежать в языке, а другое нет. Такое слово z будем называть *различающим* словом для слов x и y (относительно отношения $\sim_L$).

Контрольный вопрос 37. Пусть $x \in L$, а $y \notin L$. Возможно ли, что $x \sim_L y$?

[16] Под разбиением множество на подмножество понимают, что $A = \bigcup_{X \in \mathcal{F}} X$, где $\mathcal{F}$ — семейство множеств, причём если $X, Y \in \mathcal{F}$, то $X \cap Y = \varnothing$ при $X \neq Y$.

Контрольный вопрос 38. Докажите, что приписав к любому слову $y \in [x]$ букву a получится слово из класса $[xa]$.

Задача 29. Найдите классы $L_=$-эквивалентности, где

$$L_= = \{w \mid |w|_a = |w|_b\};$$

язык задан над алфавитом $\Sigma = \{a, b\}$.
Решение. Покажем, что классами эквивалентности языка $L_=$ являются множества

$$L_{=i} = \{w \mid |w|_a - |w|_b = i\}, i \in \mathbb{Z}.$$

Для того, чтобы это проверить установим следующее:

a) Все слова из множества $L_{=i}$ попарно эквивалентны;

b) Если $x \in L_{=i}$, а $y \in L_{=j}$, то $x \not\sim_{L_=} y$ при $i \neq j$;

c) Каждое слово $w \in \Sigma^*$ попадает в некоторое множество $L_{=i}$.

Проверим первое. Пусть $x, y \in L_{=i}$, покажем что для каждого слова z слова xz и yz либо одновременно принадлежат $L_=$, либо не принадлежат. Это очевидно: разность между числом букв a и b у слов x и y одинаковая, а при приписывании к ним любого слова z, эта разность также меняется одинаково — на $|z|_a - |z|_b$.

Перейдём к пункту **b**). Пусть $i > 0$. Выбрав в качестве различающего слова b^i, получим, что $xb^i \in L_{=0} = L_=$, а $yb^i \in L_{=j-i} \neq L_=$, а значит $yb^i \notin L_=$, поскольку $L_i \cap L_j = \varnothing$ при $i \neq j$.

Пункт **c**) тривиален: для каждого слова w определена разность $|w|_a - |w|_b = i$, а значит $w \in L_{=i}$. □

Контрольный вопрос 39. Какое различающее слово нужно выбрать для пункта **b**) при **а**) $i < 0$, **б**) $i = 0$?

Решение. Если $i < 0$ выберем слово a^i, а при $i = 0$ выберем ε. □

Контрольный вопрос 40. Почему условия **a)**, **b)** и **c)** необходимы и достаточны для доказательства того, что $L_{=i}$ — классы $L_=$-эквивалентности?

Решение. Условие **a)** следует из К.В. 35, а потому необходимо.

Теорема 3 утверждает, что классы эквивалентности не пересекаются или совпадают, а потому необходимо условие **b)**. Поскольку классы эквивалентности образуют разбиение множества (в данном случае) всех слов, то условие **c)** также необходимо.

Вместе условия **a)**, **b)** и **c)** утверждают, что $L_{=i}$ являются классами $L_=$-эквивалентности: из них вытекает, что каждое слово из $L_{=i}$ эквивалентно σ^i, где $\sigma = a$ при $i \leqslant 0$[17] и $\sigma = b$ при $i > 0$ (условие **a)**); при этом, все слова эквивалентные σ^i попадают в множество $L_{=i}$ (условия **b)** и **c)**). Отсюда, $L_{=i} = [\sigma^i]$.

□

Задача 30. Верно ли, что язык L всегда является классом L-эквивалентности?

7.3 Доказательство теоремы

Теорема 4 (Дж. Майхилл, Э. Нероуд)**.** *Язык L регулярен тогда и только тогда, когда число классов L-эквивалентности конечно.*

Перед доказательством теоремы обсудим как оно связано с предыдущими наблюдениями. Диагональная лемма (лемма 1) объясняет, что если классов L-эквивалентности бесконечно много и для каждого класса $[x_i]$ есть слово y_i, которое различает x_i и x_j при $i \neq j$, то язык L — нерегулярный. Теорема Майхилла-Нероуда (в случае нерегулярности L) ослабляет диагональное условие: достаточно, чтобы различающее слово нашлось для каждой пары слов (x_i, x_j), а вовсе не для всех x_j сразу, при фиксированном i.

[17] $a^0 = \varepsilon$

Условие неразличимости состояний минимального автомата напоминает условие L-эквивалентности и это не случайно. Если язык L регулярный, то для него есть полный минимальный ДКА $\mathcal{A}$, все состояния которого неразличимы. Тогда определив для каждого состояния q автомата $\mathcal{A}$ множество

$$L_q = \{x \mid q_0 \xrightarrow{x} q\} \tag{5}$$

получим, что множества L_q и есть классы эквивалентности Майхилла-Нероуда для языка L.

Эти связи описывают идею доказательства теоремы. Разобьём доказательство теоремы на две леммы.

Лемма 3. *Если классов L-эквивалентности бесконечно много, то язык L нерегулярный.*

Доказательство. Докажем от противного. Если L — регулярный язык, то есть полный ДКА $\mathcal{A}$, распознающий L. Пусть у автомата $\mathcal{A}$ будет n состояний. Пусть $x_1, \dots x_{n+1}$ — представители разных классов эквивалентности и пусть обработав слово x_i из начального состояния автомат $\mathcal{A}$ оказывается в состоянии q_i.

По принципу Дирихле найдётся два различных слова x_i и x_j, для которых состояния q_i и q_j совпадают. Возьмём различающее слово z и пусть, например, $x_i z \in L$, а $x_j z \notin L$. Но обработав z из состояния q_i ДКА $\mathcal{A}$ попадёт либо в принимающее состояние, либо в непринимающее — значит слова x_i и x_j неразличимы — противоречие. □

Лемма 4. *Если число классов L-эквивалентности конечно, то язык $L \subseteq \Sigma^*$ распознаётся полным ДКА $\mathcal{A}_{\sim_L} = (Q, \Sigma, q_0, \delta, F)$:*

- $Q = \{[x] \mid x \in \Sigma^*\}$;
- $q_0 = [\varepsilon]$;
- $\delta([x], a) = [xa]$;
- $F = \{[x] \mid x \in L\}$;

Доказательство. Начнём с того, что автомат $\mathcal{A}_{\sim_L}$ корректно определён. Множество его состояний — это классы L-эквивалентности, число которых конечно. Переходы определены корректно, потому что приписав к любому слову $y \in [x]$ букву a получится слово из класса $[xa]$ (К.В. 38). Справившись с К.В. 37 читатель доказал, что все слова класса $[x]$ либо одновременно принадлежат L, либо нет, значит множество принимающих состояний тоже определено корректно.

Из конструкции ясно, что автомат $\mathcal{A}_{\sim_L}$ распознаёт язык L: поскольку $q_0 \xrightarrow{x} [x]$, то слово x принимается ДКА $\mathcal{A}_{\sim_L}$ тогда и только тогда, когда $[x] \subseteq L$, а из последнего условия следует, что $x \in L$. □

7.4 Связь с ДКА

В этом разделе мы докажем, что автомат $\mathcal{A}_{\sim_L}$, построенный в лемме 4, является минимальным полным ДКА, распознающим язык L и изучим связь L-эквивалентности с ДКА. Для зафиксированного полного ДКА $\mathcal{A}$ мы будем использовать языки L_q, определённые формулой (5), а также нам понадобятся языки

$$R_q = \{x \mid q \xrightarrow{x} q_f, q_f \in F_{\mathcal{A}}\} \tag{6}$$

Языки L_q и R_q называют соответственно *левыми* и *правыми* языками для состояний q автомата $\mathcal{A}$. Если нужно уточнить автомат, то будем указывать его в качестве верхнего индекса.

Утверждение 11. *$\mathcal{A}_{\sim_L}$ — полный минимальный ДКА, распознающий L.*

Доказательство. Доказательство от противного. Пусть полный ДКА $\mathcal{B}$ имеет меньше состояний, чем автомат $\mathcal{A}_{\sim_L}$.

Левых языков $L_q^{\mathcal{B}}$ меньше, чем классов L-эквивалентности (первых столько же, сколько и состояний $\mathcal{B}$). Поэтому, какой-то из языков $L_q^{\mathcal{B}}$ содержит хотя бы два слова x и y из разных классов. Взяв для них различающее слово z получим, что оба слова xz и yz принимаются автоматом $\mathcal{B}$, что невозможно. □

Опишем теперь связь с ДКА в общем случае. В предыдущем утверждении мы фактически установили, что левые языки минимального автомата совпадают с классами эквивалентности Майхилла-Нероуда. Изучение общей картины оставим читателю в качестве задач.

В следующих задачах зафиксирован полный ДКА $\mathcal{A}$, распознающий язык L. Обозначим $q_x = \delta(q_0, x)$.

Задача 31. Докажите, что

а) каждый левый язык L_q является подмножеством некоторого класса L-эквивалентности: $x \in L_q \Rightarrow L_q \subseteq [x]$.

б) для каждого класса эквивалентности $[x]$ существует подмножество состояний $Q_x \subseteq Q_{\mathcal{A}}$, такое что

$$[x] = \bigcup_{q \in Q_x} L_q.$$

в) если $x \in L_q$, то $L_p \subseteq [x]$ тогда и только тогда, когда $R_q = R_p$.

Задача 32.

1. Докажите, что $aR_{q_{xa}} \subseteq R_{q_x}$.
2. Верно ли, что $R_{q_x} = \bigcup_{a \in \Sigma} aR_{q_{xa}}$?

Задача 33. Докажите, что склеив между собой все состояния q_x и q_y, у которых совпадают правые языки, получится ДКА $\mathcal{A}_{\sim_L}$, т.е. минимальный автомат.

Задача 34. Докажите, что склейка двух состояния q_x и q_y, у которых совпадают правые языки, приводит к ДКА, эквивалентному $\mathcal{A}$.

Замечание 6. *Решив предыдущие задачи не используя свойства, доказанные для алгоритма минимизация, вы докажете корректность этого алгоритма.*

Список литературы

1. *Верещагин Н.*, *Шень А.* Лекции по математической логике и теории алгоритмов. Начала теории множеств. — 4-е издание. — М.: МЦНМО, 2012.
2. Лекции по дискретной математике / М. Вялый, В. Подольский, А. Рубцов, Д. Шварц, А. Шень. — Черновик: http://rubtsov.su/public/DM-HSE-Draft.pdf, 2018.
3. Теория и реализация языков программирования. / В. Серебряков, М. Галочкин, Д. Гончар, Ф. М.Г. — М.: МЗ-пресс, 2006.
4. *Шень А.* Программирование: теоремы и задачи. — 2-е издание. — М.: МЦНМО, 2004.
5. *Хопкрофт Д.*, *Мотвани Р.*, *Ульман Д.* Введение в теорию автоматов, языков и вычислений. — М.: Вильямс, 2002.
6. *Sipser M.* Introduction to the Theory of Computation. — 1st. — International Thomson Publishing, 1996.
7. *Журавлёв Ю. И.*, *Флёров Ю. А.*, *Федько О. С.* Дискретный Анализ. Комбинаторика. Алгебра логики. Теория графов. — М.: МФТИ, 2012.