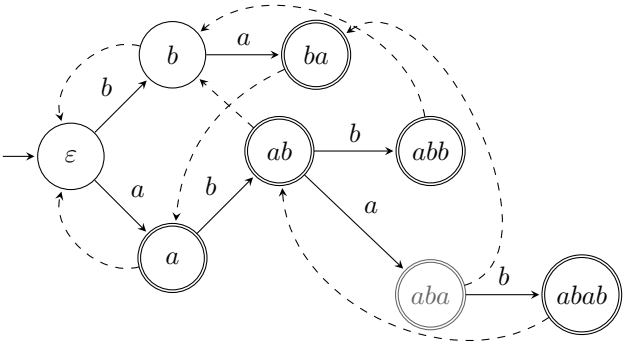
Заметки и задачи о регулярных языках и конечных автоматах

Александр Рубцов

alex@rubtsov.su



Оглавление

Предисловие

Эти заметки написаны по материалам моих семинаров по курсу «Теория и реализация языков программирования», которые я веду на факультете управления и прикладной математики МФТИ. В течение последних шести 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 семинаров. Сил хватило только на половину курса — на часть, посвящённую регулярным языкам и конечным автоматам. В процессе написания текста я осознал, что рассказ без доказательств утверждений и особенно корректности алгоритмов не складывается, так что в итоге получилась небольшая книжка, которая покрывает первую половину курса.

Учебников, посвящённых формальным языкам и конечным автоматам (в которые традиционно входит тема этой книжки), написано немало (см. список литературы). Однако, благодаря плодотворной работе над курсом, которую ведёт коллектив преподавателей, в материалы курса вошли темы, полезные и естественные для изучения формальных языков и их приложений, но часто не отражаемые в учебной литературе (особенно русскоязычной). А именно: алгоритмы обработки текста на основе конечных автоматов, теорема Майхилла–Нероуда (критерий регулярности языка), а также редко встречающееся описание регулярных языков уравнениями с регулярными коэффициентами.

При написании книжки 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 неслучайно. Естественные и интуитивные гипотезы в этой области могут оказаться ложными, поэтому обоснование словом *очевидно* здесь, пожалуй, более опасно, чем во многих других математических предметах.

Для того чтобы помочь студентам лучше работать с определени-

ями, я старался обратить внимание на тонкие моменты при помощи контрольных вопросов (к большинству которых приведены ответы и решения!). Прежде чем читать авторские ответы, стоит потратить силы и попытаться ответить самостоятельно. После чего свериться с автором, дабы убедиться, что вы не попались в ловушку.

Основная цель контрольных вопросов — показать подход к изучению математики, адептом которого является в том числе и автор. Увидев новое определение или формулировку теоремы, не нужно 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 — это часто естественные вопросы, которые возникают у меня при работе с определениями, а порой отражают встреченные мною за время преподавания трудности студентов, часто возникающие не на пустом месте.

Для достижения этой цели необходимо, чтобы всё было хорошо определено. Поэтому изложение начинается с вводного раздела по теории множеств. Отдельные определения встречаются редко, по этой причине определяемое понятие выделяется *курсивом*[1] по ходу текста.

Текст организован следующим образом. В каждом разделе идёт сначала теоретический материал, который снабжён контрольными вопросами и задачами по ходу изложения. Их лучше выполнять в процессе чтения. Далее идёт раздел с задачами и (для первых разделов) дополнительными (но обязательными) контрольными вопросами, после чего приведены ответы к контрольным вопросам и иногда решения избранных задач. Подробнее об организации задач в книжке и об ожиданиях об уровне подготовки читателя сказано в последующих подразделах предисловия.

Я благодарен коллегам, с которыми я вёл и продолжаю вести курс ТРЯП: Д. А. Голубенко, Д. Р. Гончару (который тщательно вычитал этот текст), Д. Лещёву, В. А. Серебрякову, К. Б. Теймуразову и особенно С. П. Тарасову, который в своё время привил мне любовь к формальным языкам и рассказал о многих интересных сюжетах в этой науке, часть из которых вошли и в эту книжку. Благодаря активной работе, которая ведётся коллективом преподавателей по данному курсу, он существенно отошёл от классического курса по формальным языкам (на мой взгляд, в правильную сторону), и эти изменения и отражены в этой книжке.

[1] 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.

Также я благодарен М. Н. Вялому, постоянные обсуждения педагогики с которым и привели к выбору формата этой книжки, а наша совместная научная работа в области формальных языков несомненно повлияла и на содержание.

Я благодарен В. А. Алексееву, который, будучи студентом, откликнулся на призыв подготовить иллюстрации к задачам, разобранным на семинарах. Благодаря ему моя работа над книжкой значительно упростилась.

Также я благодарен В. К. Филипенко и Я. Д. Томинину, которые, изучая ТРЯП по черновику этой книжки, нашли и помогли исправить немало неточностей.

Я прошу сообщать мне об опечатках, ошибках и неточностях, допущенных в книжке, по почте alex@rubtsov.su. Исправленные и обновлённые версии этого текста будут доступны в свободном доступе на сайте http://rubtsov.su (на данный момент на странице курса ТРЯП-2018).

О задачах

Разные задачи в этой книжке служат разным целям, и в этом разделе мы постараемся это прояснить. **Важно:** обратите внимание на обозначения ниже!

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

- **Контрольные вопросы** — вопросы, нужные для проверки того, 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 (ответы есть к большинству вопросов).

- **Задачи**, которые встречаются **по ходу изложения**, призваны акцентировать внимание читателя на происходящем. Их лучше решить, но можно и пропустить. Порой такие задачи нужны для дальнейшего изложения или прояснения происходящего, и к ним приводятся решения.

- **Задачи в конце каждого раздела** собраны в основном из задач, которые подобраны мною для обсуждения на семинаре и для домашних заданий. Я старался ко многим разделам добавить больше задач, пригодных для этих целей, чем реально использовал.

- **Задачи в конце книжки** призваны помочь закрепить весь материал и подготовиться к семестровой контрольной. Они взяты в основном из разных контрольных и экзаменов прошлых лет.

Многие из задач, вошедших в книжку, были придуманы мною за время работы над курсом. Также в книжке использовано много классических задач — многие из них встречаются, например, в [1]. Как часто бывает, часть придуманного оказалось классикой. Я решил не браться за ювелирную (и не очень благодарную) деятельность разделения задач на авторские и неавторские, поэтому, как и во многих учебниках, задачи указаны без авторства. Но некоторые задачи были составлены вместе с коллегами, а про некоторые я узнал от коллег и больше в такой формулировке нигде не встречал. Я старался отметить все такие факты и прошу прощения, если что-то упустил — сообщите об этом и текст будет поправлен.

Обозначения

Задачи в конце разделов отмечены просто номерами. В решениях используются сокращения «З.», «К.В.», «Упр.».

Задачи (и контрольные вопросы и упражнения), которые будут разобраны далее, помечены символом «○». Как правило, решение или ответ помещено в последний подраздел раздела, но бывают и исключения. Задачи на применения алгоритмов, разобранных на семинаре, часто разбиты на контрольные вопросы, и решение к ним приведено чуть дальше в разделе. Иногда изложение требует привести решение сразу после формулировки задачи, и в этих случаях я использую блок

,

который призывает читателя остановиться и попробовать решить задачу, прежде чем читать дальше. Или же о грядущем решении предупреждается в абзаце сразу после задачи.

Задачи повышенной трудности помечены символом «∗».

Ожидания от подготовки читателя

Изложение ориентировано на второкурсников ФУПМ МФТИ и отталкивается от их учебного плана. Предполагается, что читатель владеет базовой математической культурой. Однако при изучении курса ТРЯП математическая культура всё ещё насаждается. Этим обусловлено начало изложения с базовых фактов о теории множеств, на которые ожидаемый читатель мог не обратить достаточного внимания будучи первокурсником. Прошу второкурсников отнестись к этому разделу с вниманием: увы, ни раз выяснялось, что задача на контрольной или экзамене решена неправильно из-за неверного понимания описанного в условии множества. Я старался, чтобы всё в книжке было достаточно строго определено, поэтому в текст добавлены разделы о бинарных отношениях и отношениях эквивалентности.

Работа с книжкой может оказаться непростой, потому что для должного освоения материала нужно отвечать на контрольные вопросы и решать задачи. Если читатель понимает, что ему нужно более подробное и простое изложение, то я рекомендую обратиться к учеб-

нику Хопкрофта, Мотвани и Ульмана [2]. Также базовый материал изложен в классической книге М. Сипсера [1] на английском.

Предполагается, что читателю знакомы базовые алгоритмы (необходимо знать поиск в ширину) и асимптотические оценки сложности алгоритма. Если требуется восполнить пробелы, то для этих целей подойдут книги [3], [4] или [5].

Одной из стандартных проблем при изучении курса ТРЯП является плохое владение методом математической индукции. Призываю обратить внимание на его применение в ходе всего изложения и особое внимание уделить подразделу 1.5.

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 [6] и [7].

Формально определить понятие множества — 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

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

В последнем описании множества элементы повторяются, но в самом множестве нет — повторения в описании разрешены для удобства.

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 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 нечётных простых чисел, поэтому в данном случае, лучше указать в списке ещё и 9.

Другой важный способ задания множеств состоит в описании свойства, которым обладают все 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 $B \subseteq A$.

Замечание 1. При решении задач по формальным языкам, часто приходится доказывать равенство двух множеств, скажем A и B. 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 принадлежат множеству B;
- *симметрическая разность* $A \triangle B$ состоит из элементов, принадлежащих ровно одному из множеств.

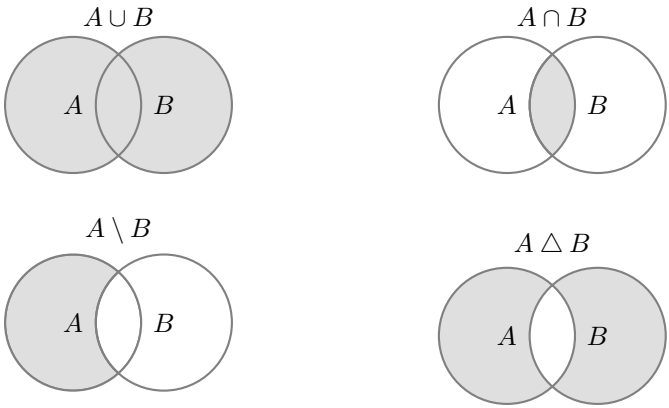


Рис. 1. Диаграммы Эйлера–Венна

Изобразим эти операции при помощи кругов (рис. 1[1]).
Говорят, что множества A и B *пересекаются*, если $A \cap B \neq \varnothing$.

Замечание 2. 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 $\inf \varnothing = +\infty$.

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

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

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

Мы используем наивную теорию 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 2^n — это число всех подмножеств множества из n элементов.

Напомним читателю, что множества A и 2^A неравномощны для любого множества A. 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 $\mathbb{N} \times \varnothing$.

Ответ на контрольный вопрос приведён в подразделе с ответами в конце раздела 1.

Мы будем считать для удобства операцию декартова произведения ассоциативной (если операций $\times$ несколько, то порядок их применения неважен):

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

$$= X \times Y \times Z.$$

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 в результате попытки 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, множество слов с операцией конкатенации (склейки) — достаточно простая алгебраическая структура (свободный моноид). Если же читатель не любит алгебру, хотя бы в той же степени, что 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*. Во всех разделах мы будем работать с алфавитом $\Sigma = \{a, b\}$, если не оговорено противное.

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, слово uuu записывают как u^3.

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 — последовательность 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 слове 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 представимо 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*[1]: $X^* = \varepsilon + X + X^2 + X^3 + \ldots + X^n + \ldots$

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

- *Плюс Клини*: $X^+ = X + X^2 + X^3 + \ldots + X^n + \ldots = XX^*$.

При возведении в степень мы считаем, что $n \geqslant 1$. При $n = 0$ мы считаем, что $X^0 = \{\varepsilon\}$, если только $X \neq \varnothing$. Операция возведения в степень переносится на слова как на одноэлементные множества: $a^3 = aaa$, $b^0 = \varepsilon$.

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 $w = u$.

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:

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

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.

Регулярное выражение — 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 $\{a\}$, $(a \mid b) = \{a, b\}$, $(a \mid b)^* = \{a, b\}^* = \Sigma^*$. В случае отсутствия скобок самый высокий приоритет у итерации, затем у конкатенации, а самый низкий у объединения. Приоритеты расставлены по аналогии с арифметикой: возведение в степень, умножение, сложение.

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 подходящие, так и неподходящие под определение объекты. 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 используя мощностной метод. Чтобы превратить объяснение в доказательство, читателю нужно выполнить следующее упражнение (или схалтурить и изучить матчасть по [7]).

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.

Итак, объясним существование нерегулярного языка. Заметим, что множество регулярных языков счётно: $\mathsf{REG} = \bigcup\limits_{k=1}^{\infty} R_k$, а счётное объединение конечных множеств не более чем счётно, в то время как множество всех языков 2^{Σ^*} имеет мощность континуум. Отсюда следует, что $\mathsf{REG} \subsetneq 2^{\Sigma^*}$, т. е. существуют и нерегулярные языки.

1.5. Предостережение перед решением задач

Среди задач этого раздела достаточно много задач вида «постройте регулярное выражение для языка L, заданного описанием. . . ». Для решения нужно построить РВ, задающее язык R, и доказать, что языки L и R совпадают. Если в задаче любого раздела требуется построить объект, то решение задачи включает доказательство корректности построения, если только в условии не оговорено противное! С этого места и начинаются стандартные трудности.

Во-первых, в качестве доказательства не годятся аргументы в духе «ну ясно же, что ...». Для доказательства нужно использовать схему рассуждения, описанную в замечании 1.

При использовании описанной схемы решения просто маханий руками (то очевидно, сё очевидно) недостаточно. Если вы сомневаетесь, достаточно ли убедительно ваше решение для неограниченного круга проверяющих, то используйте метод математической индукции.

Тут, увы, возникает основная проблема, которой и посвящён этот подраздел. Огромное число второкурсников применяют метод математической индукции неправильно. Метод математической индукции можно применять только к рассуждениям, зависящим от натурального параметра, вида $A(n) =$ «каждое слово $w \in R$ длины n принадлежит языку L». Если в доказательстве по индукции нет натурального параметра, то это не доказательство по индукции. Сам метод состоит в доказательстве *базы*: утверждения $A(0)$ и *перехода*: утверждения $A(n) \Rightarrow A(n+1)$.

Не обязательно каждый раз с официозом выделять доказываемое утверждение как это сделано в предыдущем абзаце. Достаточно просто указать параметр. Так, доказательство по индукции утверждения $A(n)$ можно объявить фразой: «докажем включение $R \subseteq L$ индукцией по длине слова».

Ожидаемую в этом курсе строгость можно оценить, взглянув на решение задачи 3(д). Всё сказанное выше относится и к другим задачам в курсе, в первую очередь на доказательство равенства двух объектов.

1.6. Задачи

Напоминаем, что если в задаче не указан алфавит, то подразумевается алфавит $\Sigma = \{a, b\}$ (это соглашение для всех разделов). 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)

а) для языка из (здесь и далее всех) слов, содержащих в качестве подслова слово aab;

б) для языка, слова которого не содержат подслово ab;

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

г) для языка всех слов чётной длины[2];

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

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

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

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

а) для языка десятичных записей натуральных чисел (над алфавитом $\{0, \ldots, 9\}$);

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

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

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.

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

1.7. Ответы и решения

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

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 сказано в решении предыдущего контрольного вопроса, язык Σ^n состоит из всех слов длины n. Для каждого $n \geqslant 1$ справедливо включение $\Sigma^n \subseteq \Sigma^*$, отсюда следует, что Σ^* содержит все слова ненулевой длины, а $\varepsilon \in \Sigma^*$ по определению. Получается, с одной стороны, Σ^* — язык всех слов, а с другой стороны, это 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 языков. Обратим внимание, что первое утверждение решения справедливо для любой операции, а не только для операции объединения. □

З. 3(д). Ответ: $R = (b^*ab^*ab^*)^* \mid b^*$. Докажем оба включения: $R \subseteq L$ и $L \subseteq R$. Включение $R \subseteq L$ очевидно: РВ b^* порождает слова с нулевым количеством a, РВ $b^*ab^*ab^*$ порождает слова ровно с двумя буквами a, а если слово w порождается итерацией

$$(b^*ab^*ab^*)^* = b^*ab^*ab^* \cup (b^*ab^*ab^*)^2 \cup \ldots \cup (b^*ab^*ab^*)^n \cup \ldots,$$

то $w \in (b^*ab^*ab^*)^n$ для некоторого n, а значит содержит ровно $2n$ букв a.

Включение $L \subseteq R$ докажем индукцией по количеству n букв a в слове w. База: при $n = 0$ слово w либо пусто, либо состоит только из букв b, а значит принадлежит $b^* \subseteq R$; при $n = 2$ слово w имеет вид $b^iab^jab^k$, $i, j, k \geqslant 0$ и принадлежит множеству $b^*ab^*ab^* \subseteq R$. Переход: пусть для $n = 2k$, $k \geqslant 1$ каждое слово с n буквами a принадлежит R, докажем, что тогда это справедливо и для $n = 2k + 2$. Пусть w содержит $2k + 2$ буквы a, тогда $w = xab^iab^j$ для некоторых $i, j \geqslant 0$. При этом слово x содержит $2k$, $k \geqslant 1$ букв a и принадлежит R по предложению индукции; $x \notin b^*$, а значит $x \in (b^*ab^*ab^*)^*$ и $x \in (b^*ab^*ab^*)^m$ для некоторого $m \geqslant 1$. Слово ab^iab^j также принадлежит множеству $(b^*ab^*ab^*)^*$, а именно его подмножеству $b^*ab^*ab^*$. Но тогда $xab^iab^j \in (b^*ab^*ab^*)^{m+1} \subseteq (b^*ab^*ab^*)^*$, а значит $w \in (b^*ab^*ab^*)^*$ и $w \in R$. □

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 записано слово, поданное на вход. Автомат последовательно считывает символы с ленты при помощи головки, которая может двигаться только вправо. Всевозможные конфигурации памяти образуют множество состояний автомата (конечное множество). В начале работы, автомат находится в начальном состоянии q_0.

Автомат проиллюстрирован на рис. 2. В последней клетке ленты записан технической символ $\triangleleft$ — маркер конца слова. В случае, когда на вход подаётся пустое слово, головка сразу стоит над маркером $\triangleleft$.

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

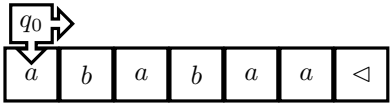


Рис. 2. Конечный автомат

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: слово считывается 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 рис. 2 и 3:

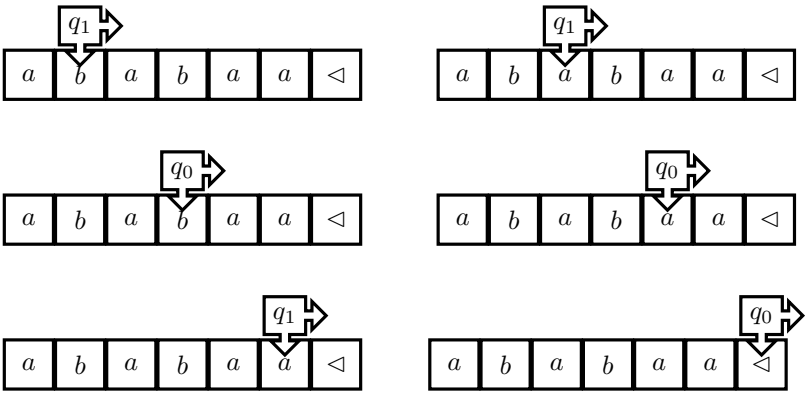


Рис. 3. Демонстрация работы автомата для К.В. 6

□

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;
- $F \subseteq Q$ – множество принимающих состояний.

Также мы будем ссылаться на описанные компоненты набора автомата, добавляя к введённым здесь обозначениям имя автомата в индексе: $F_{\mathcal{A}}$ — множество принимающих состояний автомата $\mathcal{A}$.

Замечание 3. Напомним, что 2^Q — множество всех подмножеств Q. Запись $f : A \to B$ означает, что функция определена на всех элементах множества A. 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 его переходы определены однозначно. Формально это означает, что, во-первых, для каждого состояния q и для каждого символа σ существует не больше одного состояния $q' \in \delta(q, \sigma)$, а, во-вторых, в автомате нет ни одного ε-перехода[1] ($\delta(q, \varepsilon) = \varnothing$). В общем случае автомат является *недетерминированным* (НКА). Иногда различают НКА с ε-переходами и без них, но нас эти различия интересовать не будут.

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

Детерминированный автомат начинает такт работы в состоянии q, считывает символ $\sigma \in \Sigma$ и вычисляет функцию перехода $\delta(q, \sigma)$. Если

[1] Это условие можно ослабить, потребовав, чтобы при наличии ε-перехода из состояния q больше никаких переходов из q не было. В случае конечных автоматов это ослабление только добавляет неудобства, но оно существенно в случае автоматов с магазинной памятью.

$\delta(q, \sigma) = q'$, то автомат переходит в состояние q', если же $\delta(q, \sigma) = \varnothing$, то автомат прекращает работу. Для удобства мы не различаем одноэлементное множество $\{q'\}$ и состояние q', как и в случае с языками из одного слова и словами.

Недетерминированный автомат начинает такт работы в состоянии q, *недетерминированно выбирает* состояние q' из множества $\delta(q, \sigma)$ и переходит в состоянии q', если $\delta(q, \sigma) \neq \varnothing$; иначе автомат прекращает работу. Перед тем как объяснить, что понимается под недетерминированным выбором, дадим ещё одно определение.

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. Если такое объяснение недетерминированного выбора кажется читателю недостаточно формальным, то предлагаем его не использовать, а пользоваться только определением приёма слова, которое изложено ниже в разделе 2.4 в терминах конфигураций.

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 (в разделе 3.2) мы приведём алгоритм построения ДКА по НКА, корректность которого доказывает равенство классов языков, распознаваемых автоматами.

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. Если с описанием предстоит работать человеку, а не машине, 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 читате-

ля решить этот контрольный вопрос и К.В. 8, прежде чем двигаться дальше.

При описании автомата графом (рис. 4) вершины графа соответствуют состояниям; возле начального состояния q_0 стоит входящая стрелка $\to$, а принимающие состояния (также q_0) помечены двойным кружком. На рёбрах записаны символы алфавита, по которым происходят переходы. Так ребро $q_0 \xrightarrow{a} q_1$ соответствует переходу $\delta(q_0, a) = q_1$ (переход определён однозначно, поскольку автомат детерминированный). В случае НКА, ребро $q \xrightarrow{a} p$ означает, что $p \in \delta(q, a)$.

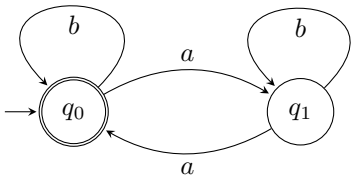


Рис. 4. Граф автомата $\mathcal{A}$

Также автомат удобно задать таблицей переходов (рис. 5). В первом столбце 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 состояния 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. 5. Описание автоматов через таблицы

Контрольный вопрос 9°. Проверьте, что граф автомата $\mathcal{A}$ (рис. 4) и таблица переходов автомата $\mathcal{A}$ (рис. 5) 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 — неслучайность. 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 отношение R — это подмножество 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 таково 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[2] отношению

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

В общем случае, под *отношением арности* k (или 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 $f : A \to 2^B$ взаимно однозначно 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 $\forall (a, b), (b, c) \in R : (a, c) \in R$.

[2] *Тернарное отношение* — это отношение арности 3.

Если выполняются все три этих свойства, то R — *отношение эквивалентности*. Отношения эквивалентности потребуются нам дальше, в разделе 5.4, где и приведём их более подробное описание.

Помимо обозначения $(a, b) \in R$ используют обозначение aRb. Бинарные отношения знакомы читателю со школьной скамьи: математические обозначения: $=, <, \geqslant$ и им подобные на самом деле — бинарные отношения. Так, запись $3 \leqslant 5$ означает, что 3 и 5 находятся в отношении $\leqslant$.

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

Итак, бинарному отношению R на множестве A соответствует граф и пара (a, b) в отношении R, если и только если в графе есть ребро из a в b. Или, иначе говоря, из a в b есть путь длины 1. В терминах графов выражаются следующие важные операции над бинарными отношениями.

Транзитивное замыкание (отношения R) — это отношение R^+, такое что $(a, b) \in R^+$ тогда и только тогда, когда в графе отношения R есть путь из вершины a в вершину b длины не меньше 1. *Рефлексивное замыкание* — это отношение $R \cup \{(a, a) \mid a \in A\}$. Эти операции дополняют отношение R до минимального транзитивного и минимального рефлексивного отношения, содержащего R как подмножество. Рефлексивное и транзитивное замыкание (одновременно) обозначают 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

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

Порядок операндов 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?

2.4. Язык конфигураций

Назовём *конфигурацией автомата* 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 $w \in \Sigma^*$ выполняется $(q, aw) \vdash (q', w)$. Таким образом, автомат *принимает слово*, если существует последовательность конфигураций

$$(q_0, w_1 w[2, n]) \vdash (q'_1, w[2, n]), (q'_1, w_2 w[3, n]) \vdash (q'_2, w[3, n]), \dots$$
$$\dots, (q'_{n-1}, w_n) \vdash (q'_n, \varepsilon),$$

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 работу из конфигурации (q, u) за некоторое число тактов.

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 (не путать с рёбрами графа автомата для которых мы используем схожее обозначение). Это обозначение можно формализовать с помощью конфигураций:

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

Обозначим через $L(\mathcal{A})$ множество всех слов, принимаемых автоматом $\mathcal{A}$. 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*».

Для описания множества всех состояний p, достижимых из q, в результате обработки слова u удобно использовать расширенную функцию переходов δ^*:

$$\delta^*(q, u) = \{p \mid q \xrightarrow{u} p\}.$$

Контрольный вопрос 10. Проверьте, что

$$L(\mathcal{A}) = \{w \mid \delta^*(q_0, w) \cap F_{\mathcal{A}} \neq \varnothing\}.$$

2.5. Построение ДКА по РВ

В этом разделе мы ответим на поставленный ранее вопрос: как алгоритмически проверить, порождает ли регулярное выражение 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 РВ по НКА мы изучим позже, в разделе 6.

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 в конец символы $\triangleright$ и $\triangleleft$, маркеры начала и конца порождённого слова, и занумеруем все символы:

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

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 множества позиций можно вычислить устройством с конечной памятью. В начале работы автомат ожидает, что ему на вход будут поданы символы, которые могут иметь позиции[3] $\{1_a, 3_b, 4_b\}$. После обработки каждого символа автомат меняет допустимое множество позиций: если первым был обработан символ a, то дальше может

[3]Запись i_a означает, что i-м символом РВ является a.

идти символ с позицией из множества $\{2_a\}$, то есть только a. После же первого символа b дальше может идти только символ с одной из позиций множества $\{3_b, 4_b, 5_a, 6_b, 7_a\}$, 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

$$A_\sigma = \bigcup_{i_\sigma \in A} \mathsf{followpos}(i_\sigma). \tag{2}$$

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 (2). 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 $L(R)$ (порождаемый РВ R).*

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

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

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 $i_{w_{n-1}} \in Q_{n-2}$, поэтому 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 $i_\triangleleft \in Q_n$, 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 символа w_1 принадлежит множеству $Q_0 = \mathsf{followpos}(0_\triangleright)$ по определению, а позиция символа w_i принадлежит множеству Q_{i-1} по построению (формула (2)). Поскольку Q_n — принимающее состояние, то $i_\triangleleft \in \mathsf{followpos}(i_{w_n})$. □

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 изучения алгоритма вычисления функции 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.

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 (синтаксическое) дерево как на рис. 6[4].

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:

- $\mathsf{firstpos}(u)$ — 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).

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

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 на рисунках ниже пример вычисления атрибутов в одном из случаев.

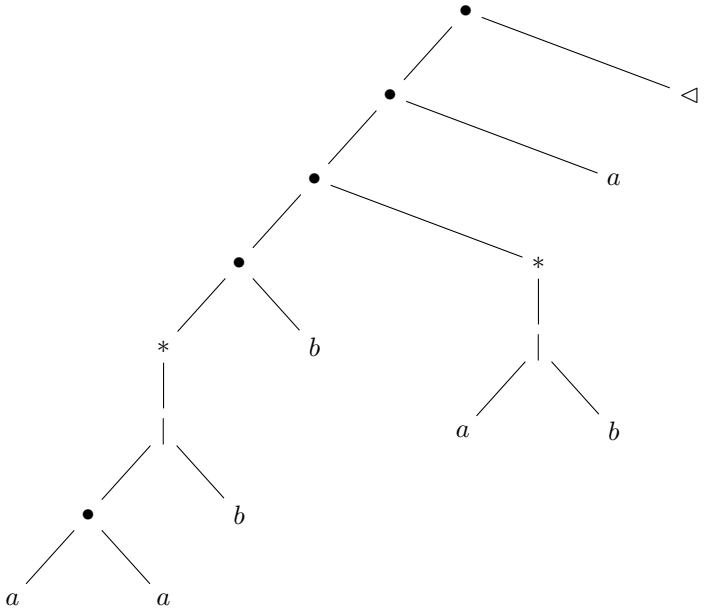


Рис. 6. Синтаксическое дерево

На рис. 7 приведены только атрибуты дочерних узлов u и v узла конкатенации $\bullet$ (атрибуты firstpos и lastpos обозначены как f и l):

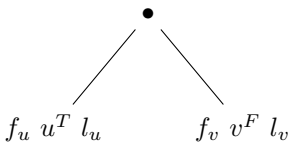


Рис. 7. Узел конкатенации

На рис. 8 вычислены атрибуты узла конкатенации:

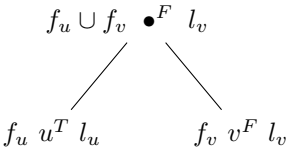


Рис. 8. Узел конкатенации с вычисленными атрибутами

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 для остальных случаев (всевозможных операций и значений атрибута $\mathsf{nullable}$) и сверьте получившийся свод правил с правилами из [8].

Контрольный вопрос 11°. Вычислите все атрибуты для дерева на рис. 6 самостоятельно и сверьте результат вычислений с рис. 9 в следующем подразделе.

Опишем алгоритм вычисления множества $\mathsf{followpos}(i)$; входом алгоритма является дерево с вычисленными атрибутами.

Алгоритм

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

Упражнение 11. Докажите корректность данного алгоритма.

Контрольный вопрос 12°. Вычислите множества $\mathsf{followpos}(i)$ для РВ (1) не заглядывая в таблицу на рис. 10 в следующем подразделе.

2.5.2. Построение автомата

Результаты вычислений атрибутов, оставленные выше читателю в качестве упражнений, приведены на рисунках 9 и 10.

Для построения автомата дело осталось за малым. Вычислить начальное состояние и выполнить построение ДКА по описанному рецепту (вычислить состояния и переходы по формуле (2)).

Контрольный вопрос 13°. Как вычислить начальное состояние?

Spoiler Alert!

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 $\lhd$. Заметьте, что для вычислений нам достаточно использовать только один из маркеров: $\rhd$ или $\lhd$, например маркер конца, — два маркера удобней для теоретических обоснований. □

Приведём теперь граф автомата, построенного по данному алгоритму (рис. 11).

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

Контрольный вопрос 14°. 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 язык $L(\mathcal{A}) \cup L(\mathcal{B})$, — обычно не сразу приходит в голову. Предлагаем читателю попробовать решить эту задачу, не заглядывая дальше.

Если это сразу не получается, то лучше сначала разобраться с замкнутостью относительно операции пересечения.

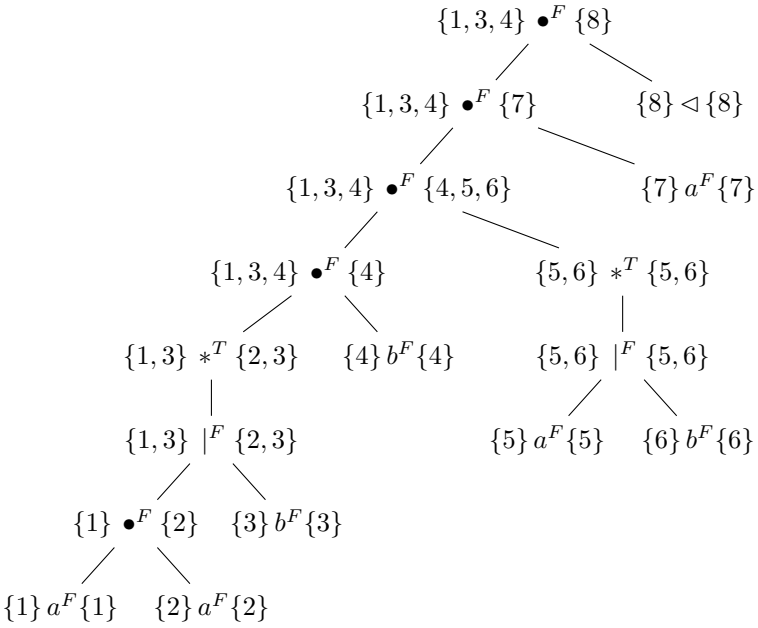


Рис. 9. Вычисление атрибутов firstpos, lastpos и nullable

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	$8_\lhd$

Рис. 10. Таблица followpos для РВ (1)

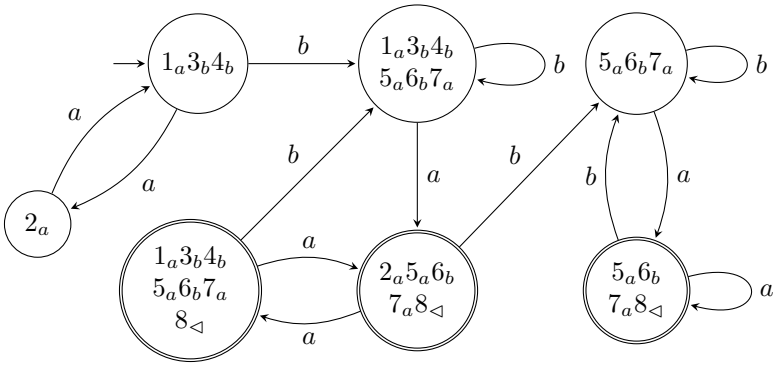


Рис. 11. Автомат, построенный по РВ 1

Контрольный вопрос 15°. 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. □

Решение следующей задачи поможет воплотить эту идею в жизнь.

Задача 5. Постройте ДКА $\mathcal{A}$, $\mathcal{B}$ и $\mathcal{C}$ над алфавитом $\{0, 1\}$, такие что

а) ДКА $\mathcal{A}$ распознаёт язык из всех слов с чётным числом нулей;

б) ДКА $\mathcal{B}$ распознаёт язык из всех слов с нечётным числом единиц;

в) ДКА $\mathcal{C}$ распознаёт язык из всех слов с чётным числом нулей и нечётным числом единиц.

Контрольный вопрос 16°. Преобразуйте эту идею в алгоритм.

Spoiler Alert!

Решение. Зафиксируем описание 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 автоматам $\mathcal{A}$ и $\mathcal{B}$ 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, (заново) изобрести которые мы оставили читателю в качестве задач.

Контрольный вопрос 17°. 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 $L(\mathcal{A}) \cup L(\mathcal{B})$?

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 18°. Верно ли, что ДКА $\bar{\mathcal{A}}$ распознаёт язык $\overline{L(\mathcal{A})}$?

Перед тем как привести решение обратим внимание читателя на смысл вопроса. Автомат $\mathcal{A}$ является параметром вопроса, и для разных автоматов ответы, вообще говоря, могут быть разными. Мы следуем правилам игры, принятым в математических текстах: при отсутствии квантора по параметру, следует полагать, что имеется ввиду квантор всеобщности. То есть вопрос можно переформулировать так: «Верно ли, что для всякого автомата $\mathcal{A}$, автомат $\bar{\mathcal{A}}$ распознаёт язык $\overline{L(\mathcal{A})}$?»

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 $\mathcal{A}$ на рис. 11;
- 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 19°. Предложите алгоритм, который строит по ДКА $\mathcal{A}$ полный эквивалентный ДКА $\mathcal{A}'$ ($L(\mathcal{A}) = L(\mathcal{A}')$).

Spoiler Alert!

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. Докажем эквивалентность автоматов. Поскольку переходы автоматов $\mathcal{A}$ и $\mathcal{A}'$ отличаются только в случае, когда у автомата $\mathcal{A}$ переход не определён, то автомат $\mathcal{A}'$ всегда оказывается в принимающем состоянии, если в нём оказался автомат $\mathcal{A}$, т. е. $L(\mathcal{A}) \subseteq L(\mathcal{A}')$. Для доказательства обратного включения заметим, что автомат $\mathcal{A}'$ попадает в принимающее состояние, только если он не попадает в состояние q_D, поскольку из него нет пути ни в одно принимающее состояние. Отсюда вытекает, что если автомат $\mathcal{A}'$ попал в принимающее состояние в результате обработки слова w, то каждый его такт работы соответствует такту работы автомата $\mathcal{A}$, который также попал в принимающее состояние в результате обработки w. □

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

6. Постройте РВ[5], порождающее язык $L(\mathcal{B})$; автомат $\mathcal{B}$ задан таблицей на рис. 5.

7. Построить РВ, порождающее язык, распознаваемый автоматом $\mathcal{B}_1$ на рис. 12.

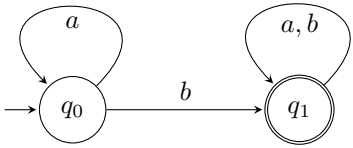


Рис. 12. Автомат $\mathcal{B}_1$

8. Постройте ДКА[6], распознающий язык $\Sigma^* aab \Sigma^*$.

[5] Требуемое в задаче РВ называют *эквивалентным* ДКА $\mathcal{B}$.

[6] Лучше построить ДКА «методом внимательного вглядывания», после чего доказать корректность.

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)** на рис. 12; **в)** на рис. 11.

11. ДКА $\mathcal{A}$ и $\mathcal{B}_1$ — автоматы на рис. 4 и 12 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. Постройте полиномиальный алгоритм, который, получив на вход ДКА $\mathcal{A}$ и $\mathcal{B}$, проверяет, совпадают ли языки $L(\mathcal{A})$ и $L(\mathcal{B})$.

13. Докажите, что несмотря на то, что алгоритм построения ДКА по РВ из раздела 2.5 применим не для всех РВ, из него следует, что для любого РВ R существует ДКА, распознающий язык $L(R)$.

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.

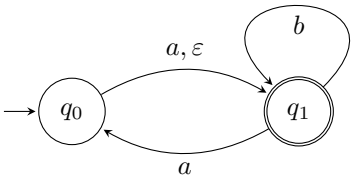
- $Q = \{q_0, q_1\}$;
- $\Sigma = \{a, b\}$;
- q_0;
- $\delta : (q_0, a) \mapsto q_1, (q_1, a) \mapsto q_0, (q_0, b) \mapsto q_0, (q_1, b) \mapsto q_1$;
- $F = \{q_0\}$.

Или можно записать весь набор как

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

□

К.В. 9. Граф автомата $\mathcal{B}$:



□

К.В. 14. 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:

$$R^+ = R \cup (R \circ R) \cup (R \circ R \circ R) \cup \dots$$

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

К.В. 17. *Нет!* Если вы ответили «да», то найдите в своих рассуждениях ошибку и постарайтесь придумать контрпример. Если самостоятельно придумать контрпример не получилось, то в качестве него сгодятся автоматы на рис. 11 и 12.

Постарайтесь исправить конструкцию произведения самостоятельно. Если этого не получилось сделать, используйте исправление из следующего подраздела для дополнения. □

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

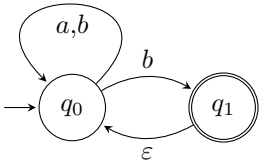


Рис. 13. НКА $\mathcal{A}$

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

Контрольный вопрос 20°. Опишите язык, распознаваемый автоматом $\mathcal{A}$ (например, в виде РВ).

Spoiler Alert!

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 воспользуемся определением регулярного языка. Напомним, что по определению любой регулярный язык можно построить 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) $X \cup Y \in \mathsf{REG}$; в) $X^* \in \mathsf{REG}$.

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

(i) НКА имеет ровно одно принимающее состояние;

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

(iii) Из принимающего состояния НКА нет ни одного перехода.

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

Контрольный вопрос 21°. Постройте НКА, удовлетворяющий условиям (i–iii), для языка **а)** $\varnothing$; **б)** $\{a\}$.

Автоматы для п. 3а и 3б 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.

Будем изображать НКА схематично, помечая в них только начальное и принимающее состояния (рис. 14):

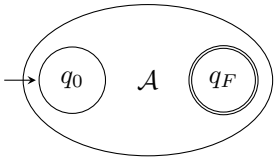


Рис. 14. Автомат, удовлетворяющий условиям (i–iii)

Далее, мы будем предполагать, что начальное состояние на схеме находится слева, а принимающее справа.

Допустим уже построены автоматы $\mathcal{A}$ и $\mathcal{B}$ (удовлетворяющие условиям (i–iii)) для регулярных языков X и Y соответственно. Построим явно автомат, распознающий $X \cdot Y$ (рис. 15).

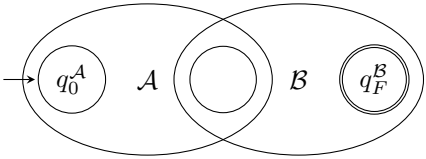


Рис. 15. Автомат для конкатенации

Как видно из рисунка, результирующий автомат получен из автоматов $\mathcal{A}$ и $\mathcal{B}$ следующим образом. Начальное состояние автомата $\mathcal{B}$ *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. Дабы определить склейку состояний для произвольных автоматов, скажем, что если бы в состояние $q_0^{\mathcal{B}}$ вели какие-то переходы, то их бы также направили в состояние, полученное в результате склейки.

Упражнение 13. Доказать, что автомат, построенный по схеме на рис. 15, распознаёт язык $X \cdot Y$.

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

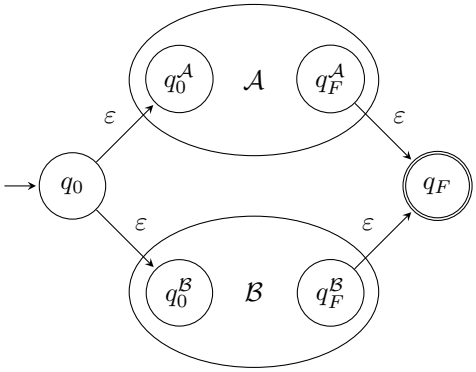


Рис. 16. Автомат для объединения

Упражнение 14. Доказать, что автомат, построенный по схеме на рис. 16, распознаёт язык $X \mid Y$.

И наконец перейдём к построению автомата для языка X^*:

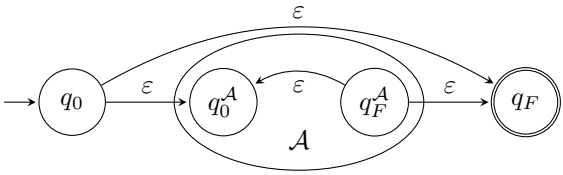


Рис. 17. Автомат для итерации

Упражнение 15. Доказать, что автомат, построенный по схеме на рис. 17, распознаёт язык X^*.

Приведённый алгоритм достаточно простой, но это не единственное его достоинство. Из алгоритма вытекает важное свойство. Легко видеть, что полученный автомат имеет примерно такой же размер, как и регулярное выражение: если у РВ длина n (число символов + число операций), то у построенного автомата $\Theta(n)$ состояний[1]. То есть мы

[1] Или не более чем cn состояний и не менее чем $c'n$, где c и c' — некоторые константы. Θ обозначает точную асимптотическую оценку, O — верхнюю, а Ω — нижнюю (на случай, если асимптотические обозначения не знакомы читателю).

установили, что для каждого РВ существует НКА с длиной описания того же порядка.

Упражнение 16. Докажите это.

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.

Задача 15°. Постройте алгоритм, который, получив на вход описание НКА $\mathcal{A}$ и слово w, отвечает на вопрос $w \stackrel{?}{\in} L(\mathcal{A})$.

Spoiler Alert!

Решение. Модифицируем под наши нужды поиск в ширину (на графе НКА). На k-м шаге алгоритма будем находить все состояния автомата $\mathcal{A}$, 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. Получившийся алгоритм приведён на рис. 18.

1. Построить множество $Q_0 = \varepsilon\text{-}\mathsf{closure}(q_0)$.

2. Для k от 1 до $|w|$ построить множества:

- $Q'_k = \{p \mid \exists q \in Q_k : p \in \delta(q, w_k)\}$;
- $Q_{k+1} = \varepsilon\text{-}\mathsf{closure}(Q'_k) = \bigcup\limits_{q' \in Q'_k} \varepsilon\text{-}\mathsf{closure}(q')$.

3. Если $Q_{|w|} \cap F_{\mathcal{A}} \neq \varnothing$, то ответ «Да», иначе «Нет».

Рис. 18. Алгоритм проверки $w \stackrel{?}{\in} L(\mathcal{A})$

Для доказательства корректности алгоритма осталось лишь 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. □

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

Приведённый алгоритм легко переделать в алгоритм построения ДКА по НКА.

Контрольный вопрос 23°. Приведите алгоритм построения ДКА по НКА.

Spoiler Alert!

Решение. Заметим, что число множеств Q_k, появлявшихся в алгоритме, конечно, поскольку $Q_k \subseteq 2^{Q_{\mathcal{A}}}$. Поэтому всевозможные множества можно хранить в 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 18:

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

Очевидно, что автомат $2^{\mathcal{A}}$ после обработки слова w оказывается в том же состоянии $Q_{|w|}$, что и алгоритм 18. Отсюда и следует корректность алгоритма построения ДКА по НКА. □

Приведём пример[2] работы данного алгоритма на рис. 19. Промежуточные вычисления удобно организовать в виде таблицы. Звёздочкой отмечены принимающие состояния.

3.3. О НКА и ДКА

Начнём с проверки внимательности читателя.

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

Spoiler Alert!

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

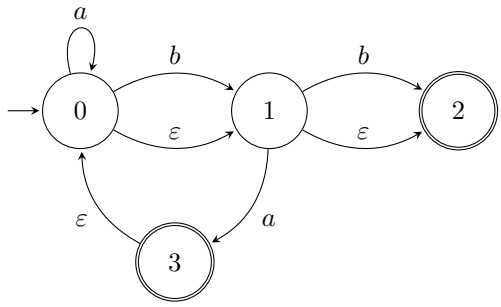
Поэтому из определения мы получаем, что НКА распознают все языки, распознаваемые ДКА, а из алгоритма построения ДКА по НКА мы получаем, что вычислительные силы этих моделей равны. Зачем же нам нужны обе?

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.

[2] Иллюстрации В. Алексеева.

НКА $\mathcal{A}$:



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

$Q_{2^{\mathcal{A}}}$	Состояния $\mathcal{A}$ в Q_i	$\delta(Q_k, a)$	$\delta(Q_k, b)$
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 $2^{\mathcal{A}}$:

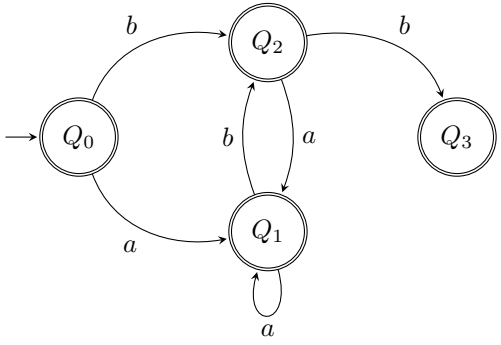


Рис. 19. Демонстрация алгоритма построения ДКА по НКА

Поставленный вопрос о достижимости верхней оценки 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. Более того, мы докажем, что для некоторых языков существуют (порождающие их) короткие регулярные выражения, но при этом для них не существует ДКА с небольшим числом состояний относительно длины РВ. Напомним, что для любого РВ длины n существует НКА с числом состояний $O(n)$.

Зафиксируем последовательность языков R_i над алфавитом $\Sigma = \{a, b\}$, состоящих из слов, в которых на i-м 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 потому 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 16°. Пусть ДКА $\mathcal{A}_n$ распознаёт язык R_n. Докажите, что $\mathcal{A}_n$ имеет не менее 2^n состояний.

Итак, детерминированные автоматы не стоит применять на практике при парсинге произвольных регулярных выражений. Однако для этих целей сгодится алгоритм из решения К.В. 15. 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. Случай уязвимости алгоритма анализа регулярного выражения из-за экспоненциального разрыва между длинами описаний в разных моделях (схожего описанному в этом разделе) приведён, например, в [9].

3.4. Замкнутость относительно операции обращения

Пусть $w = w_1 w_2 \ldots w_n$, $w_i \in \Sigma$, тогда $w^R = w_n w_{n-1} \ldots w_1$. То есть w^R — это слово w, записанное в обратном порядке. Обозначение R

здесь от слова «Reverse», хотя встречается и объяснение через связь со словом «miRror» (уму непостижимо). Операция *обращения* R естественным образом переносится на языки: $L^R = \{w^R \mid w \in L\}$.

Как ясно из названия раздела, регулярные языки замкнуты относительно операции обращения. Мы предлагаем читателю доказать это, решив следующую задачу.

Задача 17°. Постройте НКА, распознающий язык $L^R(\mathcal{A})$; автомат $\mathcal{A}$ приведён на рис. 20:

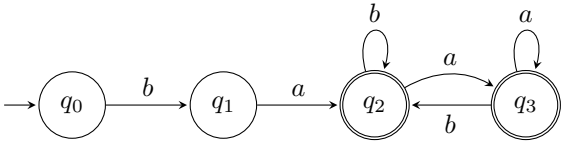


Рис. 20. Автомат $\mathcal{A}$

Spoiler Alert!

Решение. Построим НКА $\mathcal{A}^R$, распознающий $L^R(\mathcal{A})$. Чтобы автомат $\mathcal{A}^R$ читал слово в обратном порядке, нужно развернуть в автомате $\mathcal{A}$ все стрелочки в обратную сторону. Начальное состояние $\mathcal{A}$ нужно сделать принимающим состоянием для $\mathcal{A}^R$. По логике вещей нужно сделать каждое из принимающих состояний автомата $\mathcal{A}$ начальным состоянием $\mathcal{A}^R$, но это невозможно согласно определению НКА. Для решения этой проблемы добавим новое состояние, которое объявим начальным, и сделаем из него ε-переходы в бывшие принимающие состояния автомата $\mathcal{A}$. Получившийся НКА изображён на рис. 21.

Докажем корректность данного алгоритма для произвольного автомата $\mathcal{A}$. Каждому успешному пути $q_0 \xrightarrow{w} q_f$, $q_f \in F_{\mathcal{A}}$, автомата $\mathcal{A}$ соответствует успешный путь $q_0' \xrightarrow{\varepsilon} q_f \xrightarrow{w^R} q_0$ автомата $\mathcal{A}^R$ (второй путь получается добавлением шага $q_0' \xrightarrow{\varepsilon} q_f$ к развёрнутому первому пути). Отсюда $(L(\mathcal{A}))^R \subseteq L(\mathcal{A}^R)$. Очевидно, что указанное соответствие между путями взаимно однозначно (биекция) — значит $L(\mathcal{A}^R) \subseteq (L(\mathcal{A}))^R$. □

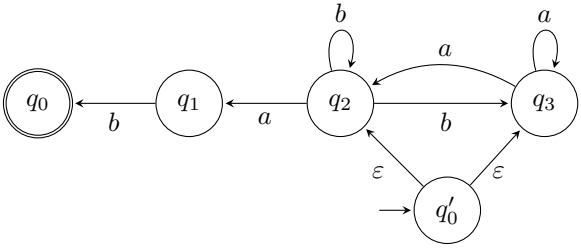


Рис. 21. Автомат $\mathcal{A}^R$

Решение задачи даёт алгоритм построения НКА $\mathcal{A}^R$, распознающего язык $L^R(\mathcal{A})$. Однако регулярность обращения регулярного языка явно вытекает из следующего наблюдения. Пусть язык L задан РВ. Потребуем, чтобы каждая подформула, к которой применялась итерация, была окружена скобками: $(\ldots)^*$. Заменим всюду $(\ldots)^*$ на $[\ldots]$, приложим к получившейся формуле зеркало и применим к «отзеркаленной» формуле обратную замену:

$$\begin{aligned} a(a)^*(aa(b)^*|b)^*(a|b)b \quad &\to \quad a[a][aa[b]|b](a|b)b, \\ a[a][aa[b]|b](a|b)b \quad &\to \quad b(b|a)[b|[b]aa][a]a, \\ b(b|a)[b|[b]aa][a]a \quad &\to \quad b(b|a)(b|(b)^*aa)^*(a)^*a. \end{aligned}$$

Ясно, что «отзеркаленное» РВ в точности порождает язык L^R.

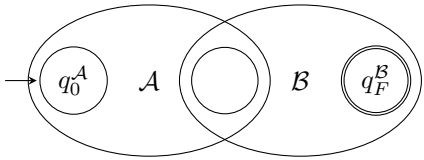
3.5. Задачи

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

19. Постройте НКА $\mathcal{A}$, распознающий слова с суффиксом $abaab$.

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

21. Докажите, что для любого НКА существует эквивалентный НКА, удовлетворяющий условиям (i–iii) раздела 3.1.



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

В результате склейки состояний все переходы, которые вели из начального состояния автомата $\mathcal{B}$, ведут теперь из бывшего принимающего состояние автомата $\mathcal{A}$, и все переходы, которые вели в начальное состояние $\mathcal{B}$, ведут теперь в бывшее принимающее состояние $\mathcal{A}$.

23. Обозначим через S_w язык слов с суффиксом w. Докажите или опровергните следующие утверждения:

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

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

3.6. Ответы и решения

К.В. 21.

□

З. 16. 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-м 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. □

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. 15. 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, автомат (КМП-автомат) будет распознавать язык $\Sigma^* w$. Такие автоматы также можно использовать для подсчёта числа вхождений образца в текст. Скажем, посчитать, сколько раз в романе «Война и мир» встречается 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, поскольку у слова w всего $|w|+1$ суффиксов.

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.

Обозначим искомый суффикс на k-м шаге (слова $t[0,k]$) через x_k. 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 на j-м шаге алгоритма:

$$t[0,j] = t[0,j-i]w[0,i]; \quad (x_j = w[0,i]).$$

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

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} — суффикс обработанной части текста $t[0,j+1] = t[0,j+1-i]w[0,i]t[j+1]$, получаем, что это также и суффикс слова $w[0,i]t[j+1]$ 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 непустого 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, $n > 0$; **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°. Построить КМП-автомат $\mathcal{A}_{abaab}$ (распознающий язык $\Sigma^* abaab$).

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. Несмотря 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, а ссылка определена.

Программисты справедливо обратят внимание на следующую 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. *ДКА с ссылками-исключениями* $\mathcal{A}^{\mathsf{exc}}$ задан 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, к которой будут добавлены переходы, реализуемые ссылками-исключениями.

Эти переходы определяются следующим образом. Если переход из состояния q по символу 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 $\delta_{\mathcal{A}}(q, a) = r$. 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 рис. 22. 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.

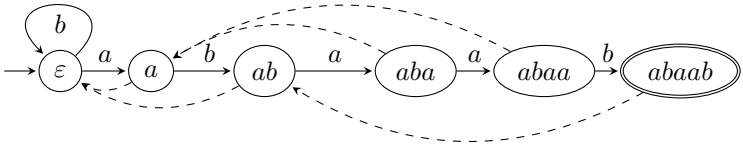


Рис. 22. КМП-автомат с суффиксными ссылками (ссылками-исключениями)

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

Контрольный вопрос 29°. По какому алгоритму нужно добавлять ссылки-исключения в автомат $\mathcal{A}^{\mathsf{exc}}_{w}$? Попробуйте сообразить, прежде чем читать дальше.

Для ответа на этот вопрос нам потребуются свойства детерминированного КМП-автомата, связанные со свойствами префикс-функции l. Обозначим через $l^m(u)$ слово, получающееся последовательным применением функции l к слову u, т. е.

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

При построении переходов детерминированного КМП-автомата мы использовали префикс-функцию:

$$w[0,i] \xrightarrow{a} l(w[0,i]a) = w[0,k] \text{ при } w[i+1] \neq a.$$

Заметим, что $w[0,k-1]$ является суффиксом слова $w[0,i]$, который одновременно является его же префиксом. 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 $i > k' - 1 > k - 1$. 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 $w[0, k-1] = l^m(w[0, i])$ для некоторого $m \geqslant 1$. 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-исключениями. 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, описать вычисление значений префикс-функции на всех префиксах слова 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. 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, по утверждению 3, $w[0, k{+}1] = l(w[0, i])$. Из того же утверждения получаем, что первое событие всегда наступает при условии $l(w[0, i]) \neq \varepsilon$, а значит при наступлении только второго события справедливо $l(w[0, i]) = \varepsilon$.

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 — будем платить по рублю за переход по суффиксной ссылке. 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 слова $l(w[0, i])$ не превосходит количество рублей $\$_i$ в бюджете после i-го шага.*

Доказательство. Докажем утверждение индукцией по i. База очевидна: $l(w[1]) = \varepsilon$ и в запасе есть целый рубль! Пусть утверждение верно для i. При вычислении значения $l(w[0, i+1])$ 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 не менее k рублей, а $(k+1)$-й рубль нам заплатят за вычисление значения $l(w[0, i+1])$. □

На случай, если рассуждения с бюджетом кажутся слишком мудрёными, мы приведём неравенство из книги А. Шеня [10], в которой

КМП-алгоритм описан с программистской точки зрения:

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

и предложим (как и в [10]) 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 является обобщением алгоритма 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. 23 показан пример словаря с содержимым $S = \{a, ba, ab, abb, abab\}$. ДКА $\mathcal{A}_S$ принимает слово x тогда и только тогда, когда $x \in S$.

Формально, множество состояний автомата-словаря состоит из префиксов всех слов, входящих в словарь (включая сами слова). Переходы определены по правилу $u \xrightarrow{a} ua$, а принимающие состояния — слова из словаря.

Задача 24°. Постройте алгоритмы, реализующие операции **in**, **out** и **test**. В результате этих операций должен получиться ДКА, который реализует словарь с соответственно изменившимся содержимым.

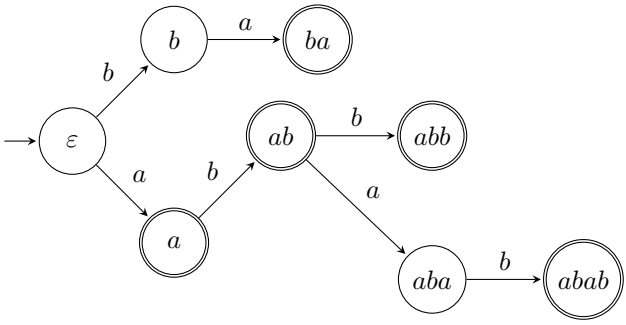


Рис. 23. ДКА $\mathcal{A}_S$ реализует словарь

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 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 приведённые выше алгоритмы работают 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!

Как и в случае КМП-алгоритма, алгоритму Ахо–Корасик соответствует одноимённый автомат $\mathcal{A}_S^{\mathsf{exc}}$ (рис. 24) — ДКА с ссылками-исключениями, построенный следующим образом. Он содержит все состояния и переходы, что и соответствующий автомат для словаря (рис. 23), 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 переходы 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.

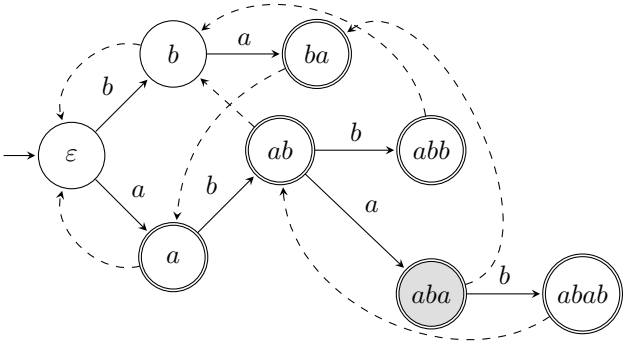


Рис. 24. Автомат Ахо–Корасик $\mathcal{A}_S^{\mathsf{exc}}$

На рис. 25 приведён ДКА Ахо–Корасик $\mathcal{A}_S$, полученный из автомата $\mathcal{A}_S^{\mathsf{exc}}$ (рис. 24) заменой суффиксных ссылок на явные переходы:

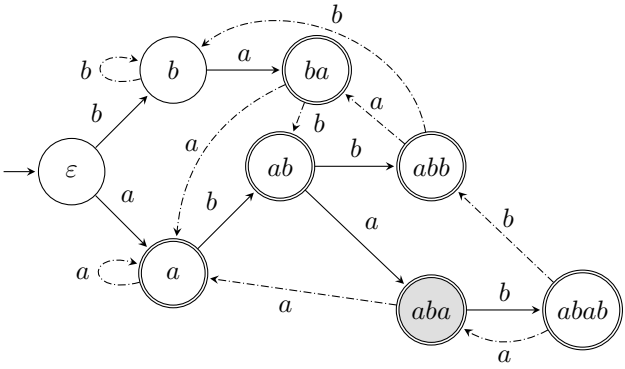


Рис. 25. ДКА Ахо–Корасик $\mathcal{A}_S$

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.

Построение ДКА Ахо–Корасик без ссылок-исключений, как правило, неэффективно для приложений, поскольку ссылки позволяют существенно сократить таблицу переходов и уменьшить тем самым длину описания автомата (как и в случае КМП-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 (который занимает линейное время 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 параметру 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 слова $t[0, j + 1]$ среди всех состояний. Иначе, если у слова $t[0, j + 1]$ есть более длинный суффикс $s't[j + 1]$, то отсюда получаем, что слово s' длиннее s и также является суффиксом $t[0, j]$, 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 и являются суффиксные ссылки). □

Для каждого состояния можно проверить (путешествуя по суффиксным ссылкам), какие суффиксы этого состояния содержатся в словаре, и сохранить эту информацию в таблице, если она нужна, на-

пример, для подсчёта всех вхождений слов из словаря в текст (считаем, что вхождения могут пересекаться).

При доказательстве 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 в 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 ссылке, начиная с перехода из s, со счёта вершины $u\sigma$ списывается рубль.

Для доказательства заявленной оценки достаточно доказать следующие утверждения:

1. Вершинам было выдано суммарно не более $\sum\limits_{u \in S} |u|$ 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. Перед тем как его сформулировать заметим, что если проводить только начисления и не учитывать траты, то все начисленные средства в итоге окажутся у *листьев* — вершин, из которых определены переходы только по суффиксным ссылкам. Слово может быть листом, только если оно входит в словарь, таким образом сумма длин листьев не превосходит $\sum\limits_{u \in S} |u|$.

Утверждение 6. *Если провести только начисления и не учитывать траты, то на счету каждого листа z окажется $|z|$ рублей.*

Доказательство. Докажем индукцией по глубине дерева n, что после начислений средств всем вершинам уровня n на счету каждой вершины u 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 по правилу iii. □

Для доказательства второго утверждения введём вспомогательное понятие. Назовём *суффиксной 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 вытекает из следующего наблюдения. Пусть $dl(ub) = n$. Зафиксируем последовательность

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

мы полагаем $\mathsf{exc}(s_{n-1}b) = \varepsilon$. В этой последовательности s_i — это некоторый суффикс слова u, который является одновременно суффиксом слова s_{i-1} и встречается среди состояний автомата. Значит, каждой, кроме быть может последней (при $s_{n-1} = \varepsilon$), вершине пути по ссылкам из ub в ε однозначно 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. Начнём со случая начисления денег вершине ub по правилу *ii*. По предложению индукции $\$_u \geqslant dl(u)$ (после вычисления ссылки из u), а значит перед вычислениями ссылки из вершины ub, на её счету не меньше $dl(u) + 1$-го рубля. Пусть $\mathsf{exc}(ub) = sb$ и пусть при вычислении sb было сделано k переходов из $\mathsf{exc}(u)$ (последний вёл в s). Отсюда следует, что $dl(s) = dl(u) - (k+1)$, поскольку если $y = \mathsf{exc}(x)$, то $dl(y) = dl(x) - 1$. Воспользовавшись вторым свойством утверждения 7 получаем, что

$$dl(ub) = dl(sb) \overset{(7.2)}{\leqslant} dl(s) + 1 \leqslant dl(u) - k \leqslant \$_{ub}.$$

Если деньги были начислены вершине ub по правилу *iii*, то на её счету (до вычисления ссылки) оказалось $|ub|$ рублей, что больше чем $dl(u)+1$ по свойству 1 утверждения 7 и этой суммы заведомо хватит — мы установили это при анализе начислений по правилу *ii*. □

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

25. 1. Постройте КМП-автомат для слова *babbabab* (над алфавитом $\{a, b\}$).

2. Постройте для того же слова КМП-автомат $\mathcal{A}^{\mathsf{exc}}$ с суффиксными ссылками.

3. Продемонстрируйте работу автомата $\mathcal{A}^{\mathsf{exc}}$ на словах:

а) *babbabbabab*; **б)** *babbabc*.

Под демонстрацией понимается последовательность конфигураций автомата $\mathcal{A}^{\mathsf{exc}}$, т. е. пар из состояния и необработанной части слова.

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

27. Постройте для словаря $S = \{ac, acb, b, ba, c, cbb\}$ (который вы строили в предыдущей задаче) автомат Ахо–Корасик. Посчитайте с его помощью количество различных вхождений слов из словаря S в слово *acbacbb* в качестве подслов.

28°. Постройте для словаря $S = \{aac, acb, b, ac, c\}$ автомат Ахо–Корасик. Посчитайте с его помощью (или с помощью ДКА Ахо–Корасик) количество различных вхождений слов из словаря S в слово *aacbacb* в качестве подслов.

При подсчёте (в последних двух задачах) нужно продемонстрировать работу автомата на слове. То есть привести последовательность конфигураций из состояния автомата, необработанной части входа и изменения счётчика на переходе.

Последовательности конфигураций удобно привести в виде таблицы.

29. Докажите, что язык из слов, содержащих хотя бы одно подслово из множества $\{aab, ba\}$, но не содержащих подслово *babb*, распознаваем некоторым ДКА с не более чем 50 состояниями.

4.4. Ответы и решения

К.В. 25. а) a^{n-1}; **б)** bab; **в)** $baba$; **г)** b; **д)** ε. □

К.В. 26.

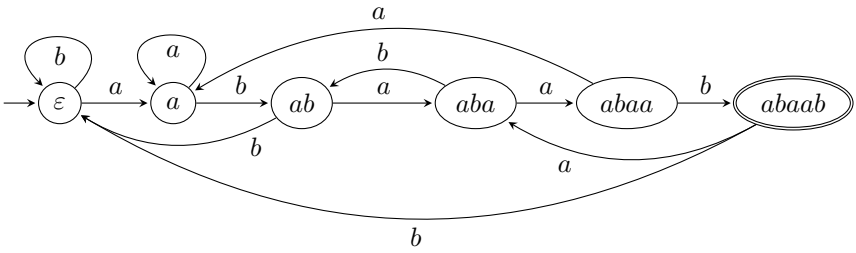


Рис. 26. Автомат $\mathcal{A}_{abaab}$ □

З. 28. На рис. 27 изображён автомат Ахо–Корасик для словаря из условия задачи. Демонстрация работы автомата приведена на рис. 28.

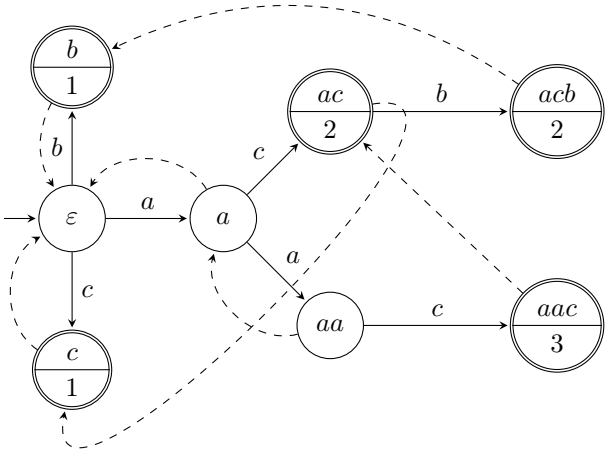


Рис. 27. ДКА $\mathcal{A}$ реализует словарь $S = \{aac, acb, b, ac, c\}$

состояние	необработанная часть входа	$+k$
ε	$aacbacb$	0
a	$acbacb$	0
aa	$cbacb$	0
aac	$bacb$	$+3$
ac	$bacb$	0!
acb	acb	$+2$
b	acb	0!
ε	acb	0!
a	cb	0
ac	b	$+2$
acb	ε	$+2$

Рис. 28. Демонстрация работы автомата Ахо–Корасик

Запись «0!» (в таблице на рис. 28) означает, что счётчик не увеличился, поскольку переход был по суффиксной ссылке, а это значит, что вклад состояния v (в которое был переход) был учтён при попадании в состояние uv по прямому переходу. Суммируя все увеличения счётчика, получаем численный ответ: 9. □

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 языка классу. До сих пор, изучая регулярные языки, мы не приводили ни одного примера нерегулярного языка, 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 регулярных 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 распознавания нужно помнить точное количество букв a, идущих перед b, сколь бы большим оно ни было. Это 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 30. Докажите лемму.

Задача 31. Докажите, что утверждение леммы справедливо не только для ДКА, но и для НКА.

Замечание 5. Эта лемма следует из теоремы Майхилла-Нероуда, изложенной дальше.

Из леммы 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 так же нелепо, как с леммой о накачке[1], а будем называть лемму 1 *диагональной леммой.*

5.2. Лемма о накачке

Лемма о накачке (Pumping Lemma) известна так же, как лемма о разрастании. Эта лемма легко вытекает из приведённого выше Ad hoc рассуждения. Основная идея доказательства: если слово w из регулярного языка L длиннее, чем число состояний распознающего его ДКА, то при обработке w встречаются два одинаковых состояния, а

[1] Я думал, что это название нельзя испортить ещё больше, пока один студент из Китая не назвал её «теоремой о каначке».

именно: $w = xyz$, 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 прояснить суть утверждения леммы.

Лемма 2. *Если L — регулярный язык, то существует такая константа $p \geqslant 1$, что для любого слова w из L длиннее p, существует разбиение $w = xyz$, такое что*

(1) $|y| \geqslant 1$,

(2) $|xy| \leqslant p$,

(3) $\forall i \geqslant 0$, $xy^iz \in L$.

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 длиннее 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 $w \in L$ длиннее p существует разбиение с указанными свойствами. □

Контрольный вопрос 31°. Докажите, что язык $L = \{a^nb^n \mid n \geqslant 0\}$ нерегулярный, используя лемму о накачке.

Spoiler Alert!

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

Воспользуемся леммой о накачке. Рассмотрим слово $w = a^pb^p$, 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 нужно применить контрапозицию: условие леммы имеет вид $L \in \mathsf{REG} \Rightarrow P$, а нужно $\overline{P} \Rightarrow L \notin \mathsf{REG}$; назовём P — *условием накачки*. Для получения отрицания условия накачки нужна замена кванторов, которая в этом случае даётся нелегко.

Контрольный вопрос 32°. Сформулируйте на естественным языке утверждение $\overline{P} \Rightarrow L \notin \mathsf{REG}$ (противоположное к обратному утверждению леммы).

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

Замечание 6. 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 и возможно получит другой ответ.

Также часто возникает следующие недоумение:

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

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

Задача 32. Покажите, что следующий язык удовлетворяет условию леммы о накачке, но сам регулярным не является:

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

Заметим, что противоречий тут никаких нет: сама лемма имеет вид импликации, заключение которой выполняется для каждого регулярного языка, но оно вполне может выполняться и для нерегулярного языка.

Во-вторых, неаккуратное применение леммы о накачке влечёт громоздкие выкладки.

Пример 1. Покажем с помощью леммы о накачке нерегулярность языка $L = \{a^n b^n \mid n \geqslant 0\}$, 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 символом b следует символ 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.

5.3. 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.

$$\forall z \in \Sigma^* : \left(q \xrightarrow{z} q_f, q_f \in F_{\mathcal{A}} \iff p \xrightarrow{z} p_f, p_f \in F_{\mathcal{A}}\right). \tag{1}$$

Интуиция подсказывает, что тогда можно избавиться, например, от p: само состояние удалить, а все ведущие в него переходы направить в q. Назовём эту процедуру *склейкой* p и q (не путать со склейкой из раздела 3). 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 (1), мы будем называть *неразличимыми*.

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

Доказательство. Воспользуемся следующим техническим трюком. Построим по ДКА $\mathcal{A}$ НКА $\mathcal{B}$, продублировав все переходы, ведущие в состояние p, в состояние q (само состояние p удалять не будем). 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[2] $q_0 \xrightarrow{x} p \xrightarrow{y} q_f$ 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 слова y из состояния q автомат $\mathcal{A}$ не встречает состояние 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 НКА $\mathcal{B}$ есть ход

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

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 $\mathcal{A}$.

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

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 $\delta(q, \sigma) = q_D$.

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-го шага алгоритма множество состояний Q разбито на j подмножеств $Q_1, \ldots, Q_j$. Зафиксируем символ $\sigma \in \Sigma$ и 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 в 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 ДКА с минимальным числом состоянием, 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). Обычно под *минимальным автоматом* понимают полный ДКА с минимально возможным числом состояний. Это соглашение удобно из-за связи с теоремой Майхилла–Нероуда; не смотря на то что мы придерживаемся этого соглашения, мы будем также напоминать читателю, что речь идёт именно о полном минимальном ДКА.

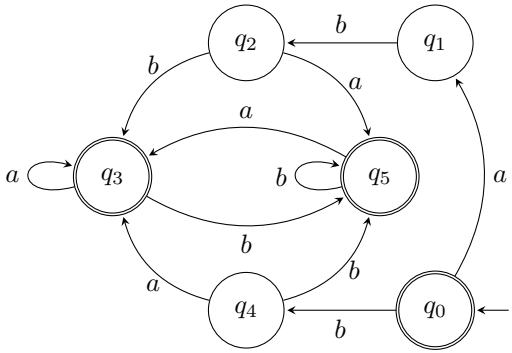
Пример 2. Минимизируем автомат $\mathcal{A}$, заданный графом на рис. 29[3].

Процесс разделения множеств проиллюстрирован на рис. 30 с помощью «ящиков». 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. 30 опущены стрелки из состояний, которые оказались единственными в ящиках. Обратите внимание, что алгоритм останавливается, когда после последовательного прохода по всем буквам алфавита разбиение не изменяется.

[3]Иллюстрации В. Алексеева.

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



Автомат $\mathcal{A}$ после пополнения:

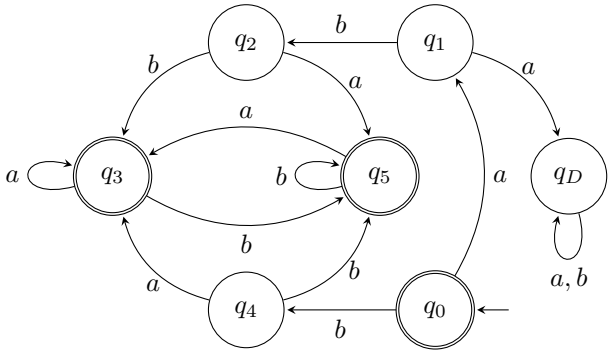


Рис. 29. Минимизация автомата: шаг 1

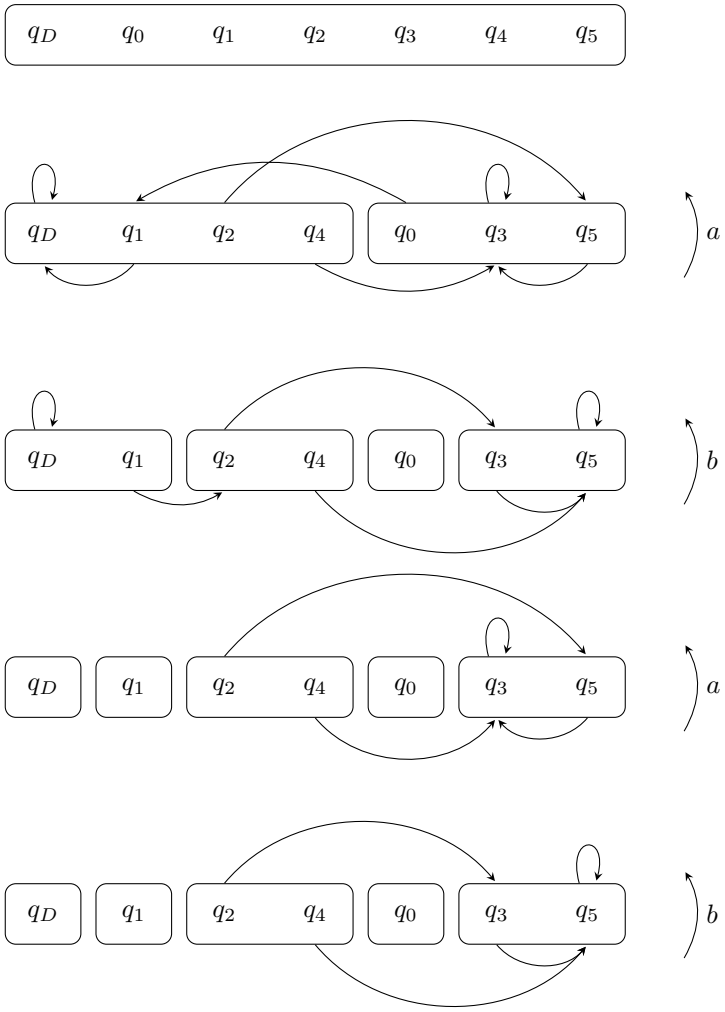
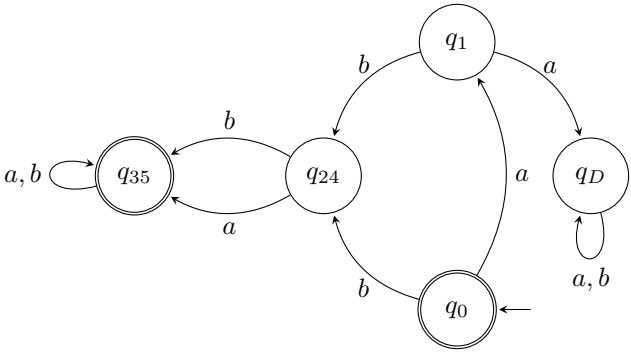


Рис. 30. Поиск различимых состояний

Полный минимальный ДКА
(в результате склейки неразличимых состояний):



ДКА с минимальным числом состояний
(после удаления тупикового состояния):

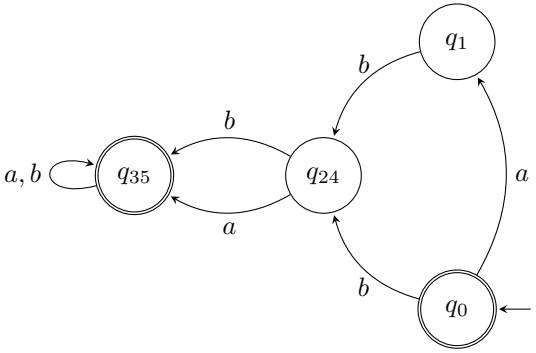


Рис. 31. Последние шаги алгоритма минимизации

Для доказательства корректности алгоритма нам придётся сначала признаться в обмане трудящихся.

Задача 33*. 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 33. 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 34 и 35, не решив задачу 33.

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

Задача 34°. Докажите, что в полном ДКА $\mathcal{B}$ не может быть меньше состояний, чем в полном «минимальном» автомате $\mathcal{A}$.

Задача 35°. 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[4].

5.3.1. Корректность алгоритма минимизации

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

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

[4] Автоматы $\mathcal{A}$ и $\mathcal{B}$ совпадают, если существует биекция $f : Q_{\mathcal{A}} \to Q_{\mathcal{B}}$, такая что $f(q_0^{\mathcal{A}}) = q_0^{\mathcal{B}}$, $f(F_{\mathcal{A}}) = F_{\mathcal{B}}$ и $q \in \delta_{\mathcal{A}}(p, \sigma) \iff f(q) \in \delta_{\mathcal{B}}(f(p), \sigma)$.

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 $p_i = \delta(p, z[1, i])$.

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. Индукцией по i получаем, что состояния q_i и p_i различимы словом $z[n-i, n]$ на i-м шаге алгоритма; а значит, что q и p будут найдены алгоритмом на $(n+1)$-м шаге. □

Перед тем как раскрыть следующую карту, начнём с ошибочного решения задачи 34.

Решение задачи 34 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 меньше.

Состояния q_x и q_y различимы (по утверждению 10), поэтому найдётся различающее их слово z. Но обработав z из состояния $q_{x,y}$, автомат $\mathcal{B}$ окажется либо в принимающем состоянии, либо в непринимающем: значит он либо примет оба слова xz и yz, хотя одно из них не принадлежит L, либо не примет оба слова, что также приведёт к противоречию. □

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

Решение задачи 33*. Вообще говоря, в ДКА могут быть состояния, недостижимые 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.

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. Однако ясно, что недостижимые состояния можно удалить, не поменяв тем самым язык. □

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 полный ДКА, построенный по алгоритму (теперь уже исправленному), «минимальным».

Исправление решения задачи 34. 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 35. Доказательство от противного. Пусть $\mathcal{A}$ — «минимальный» автомат с n состояниями. Пусть полный автомат $\mathcal{B}$ также имеет n состояний (и также не имеет недостижимых состояний, иначе удалим их и воспользуемся задачей 34). 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 $\mathcal{A}$, ведут в одно и то же состояние автомата $\mathcal{B}$.

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 34, получим, что автомат $\mathcal{A}$ был не «минимальным».

Если же случилось второе, то также получим, что состояния q_u и q_v неразличимы и должны были быть склеены в процессе минимизации автомата $\mathcal{A}$. □

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.*

5.4. Теорема Майхилла–Нероуда

Теорема Майхилла–Нероуда является критерием регулярности языка. Её 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.

5.4.1. Отношения эквивалентности

Напомним, что бинарное отношение $R \subseteq A \times A$ называется отношением эквивалентности, если оно рефлексивно, симметрично и транзитивно. 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 конгруэнтность.

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 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 $\sim \subseteq A \times A$ — отношение эквивалентности. Определим *класс эквивалентности* $[a]$ как множество всех таких элементов множества A, которые эквивалентны элементу a:

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

Контрольный вопрос 35°. Докажите, что два любых элемента из класса $[a]$ эквивалентны.

Ответ на контрольный вопрос вытекает из доказательства основной теоремы.

Теорема 3. *Классы эквивалентности* $[a]$ *и* $[b]$ *(по отношению эквивалентности* $\sim$*) либо не пересекаются*[5]*, 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 $a \sim x \sim b$ (эта запись значит «$a \sim x$ и $x \sim b$»). Значит $a \in [b]$ и из симметричности и транзитивности получаем, что каждый элемент y из класса $[a]$ также принадлежит классу $[b]$:

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

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

Эта теорема объясняет, что отношение эквивалентности разбивает[6] множество A на подмножества. С другой стороны, каждому разбиению множества A соответствует отношение эквивалентности «элементы принадлежат одному множеству из разбиения».

5.4.2. 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 из слов любого слова z превращает оба слова одновременно либо в слова из языка L, либо в слова не из языка L:

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

[5]Напомним, что множества A и B не пересекаются, если $A \cap B = \varnothing$.

[6]Под разбиением множества на подмножества понимают, что $A = \bigcup_{X \in \mathcal{F}} X$, где $\mathcal{F}$ — семейство попарно непересекающихся множеств, т. е. если $X, Y \in \mathcal{F}$, то $X \cap Y = \varnothing$ при $X \neq Y$.

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 слово x принадлежит языку L, а слово y — нет. Возможно ли, что $x \sim_L y$?

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

Задача 36. Найдите классы $L_=$-эквивалентности, где

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

язык задан над алфавитом $\Sigma = \{a, b\}$.

Spoiler Alert!

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 п. **b**. Пусть $i > 0$. Выбрав в качестве различающего слова b^i, получим, что $xb^i \in L_{=0} = L_=$, в то время как $yb^i \in L_{=j-i} \neq L_=$, а значит $yb^i \notin L_=$, поскольку $L_{=i} \cap L_{=j} = \varnothing$ при $i \neq j$.

Остался п. **c**, который 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 п. **b** при **а)** $i < 0$, **б)** $i = 0$?

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

Задача 37. Верно ли, что язык L всегда является классом L-эквивалентности?

5.4.3. 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.

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

$$L_q = \{x \mid q_0 \xrightarrow{x} q\}, \tag{2}$$

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. Регулярность языка L влечёт существование полного ДКА $\mathcal{A}$, распознающего L. Пусть у автомата $\mathcal{A}$ будет n состояний. Пусть $x_1, \ldots, x_{n+1}$ — 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 $[x] \subseteq L$, т. е. тогда и только тогда, когда $x \in L$. □

5.4.4. 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 (2), а также нам понадобятся языки

$$R_q = \{x \mid q \xrightarrow{x} q_f, q_f \in F_{\mathcal{A}}\}. \tag{3}$$

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

Доказательство. Доказательство от противного. Предположим, что 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. □

Это утверждение фактически устанавливает, что левые языки минимального автомата совпадают с классами эквивалентности Майхилла–Нероуда. Изучение общей картины оставим читателю в качестве задач.

5.5. Задачи

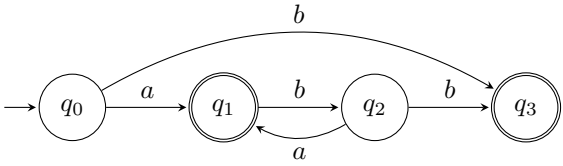
38. Пусть для языка L выполнено: $\exists n\ \forall w \in L : |w| > n$ существует разбиение w на $xuyvz$, причём $y \neq \varepsilon$ и $\forall i \geqslant 0\ xuy^ivz \in L$. Следует ли отсюда, что L – регулярный язык?

39. Укажите константу, начиная с которой для языка

$$L = \{a^n b^n \mid n \leqslant 2019\}$$

выполняется лемма о накачке.

40. Язык L распознаётся автоматом, заданным графом:



Построить ДКА с минимальным числом состояний для языка:

а) L; **б)** $\bar{L}$.

41. Опишите классы эквивалентности Майхилла–Нероуда для языка L. В случае конечности множества классов постройте минимальный полный ДКА, распознающий L, где L — язык

а) $\Sigma^* ab \Sigma^*$; **б)** $\mathsf{PAL} = \{w \mid w = w^R\}$[7]; **в)** $\{w \mid |w|_{ab} = |w|_{ba}\}$.

42. Докажите, что следующие языки нерегулярны:

а) $\{a^{2^n} \mid n \geqslant 0\}$; **б)** $\{a^n b^m a^n \mid m, n \geqslant 0\}$;

в) $\Sigma^* \setminus \mathsf{PAL}$ (см. предыдущую задачу).

43. Являются ли регулярными следующие языки:

а) $\{xy : |x| > |y|,\ x$ содержит букву $a\ \}$;

б) $\{xy : |x| < |y|,\ x$ содержит букву $b\ \}$?

44[8]**.** Язык L состоит из двоичных записей чисел (быть может с ведущими нулями), дающих остаток 2 по модулю 3. Например, $101 \in L$, поскольку $101_2 = 5_{10} = 3 + 2$ (индекс указывает основание системы счисления), а $1101 \notin L$, поскольку $01101_2 = 13_{10} = 3 \times 4 + 1$.

1. Является ли язык L регулярным?

2. Является ли обращение языка L (язык L^R) регулярным?

При положительных ответах на вопросы, постройте ДКА, распознающие языки L и L^R.

В задачах 45–47 зафиксирован полный ДКА $\mathcal{A}$, который распознаёт язык L. Обозначим $q_x = \delta^*(q_0, x)$.

45. Докажите, что

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

[7] Здесь R — операция обращения, введённая в разделе 3.4; язык PAL — это язык палиндромов, т. е. слов, которые читаются справа налево и слева направо одинаково, например «казак».

[8] Сообщил С. П. Тарасов.

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 $R_q = R_p$.

46°[9]**.** 1. Докажите, что $aR_{q_{xa}} \subseteq R_{q_x}$.

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

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

48. 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.

Замечание 7. Решив предыдущие задачи не используя свойства, доказанные для алгоритма минимизация, вы докажете корректность этого алгоритма.

49*. Обозначим через $R(\mathcal{A})$ автомат для языка $L^R(\mathcal{A})$ (обращения языка $L(\mathcal{A})$), построенный по алгоритму из раздела 3.4. Обозначим через $D(\mathcal{A})$ ДКА, полученный в результате детерминизации НКА $\mathcal{A}$.

Пусть $\mathcal{A}$ — полный ДКА. Докажите, что $D(R(D(R(\mathcal{A}))))$ — минимальный ДКА, распознающий $L(\mathcal{A})$. То есть двукратное последовательное выполнение процедур обращения и детерминизации для полного ДКА приводит его к минимальному.

[9] Решение в начале следующего раздела.

5.6. Ответы и решения

К.В. 32. Если для любой константы $p \geqslant 1$ существует слово w из языка L длиннее p, такое что для любого разбиения $w = xyz$, для которого выполняются условия (1) и (2), выполняется условие $\overline{(3)} = (\exists i \geqslant 0 : xy^i z \notin L)$, то язык L нерегулярный.

Точнее, это уже «причёсанное» условие, покажем как мы его получили. Чтобы воспользоваться контрапозицией, запишем условие леммы в кванторах:

$$L \in \mathsf{REG} \Rightarrow (\exists p\, \forall w \in L : (|w| \geqslant p) \Rightarrow (\exists x, y, z : (w = xyz) \wedge (1) \wedge (2) \wedge (3))),$$

избавимся от внутренней импликации преобразованием к дизъюнкции:

$$L \in \mathsf{REG} \Rightarrow (\exists p\, \forall w \in L : \overline{(|w| \geqslant p)} \vee (\exists x, y, z : (w = xyz) \wedge (1) \wedge (2) \wedge (3))),$$

и воспользуемся законами контрапозиции и Моргана:

$$(\forall p\, \exists w \in L : (|w| \geqslant p) \wedge (\forall x, y, z : \overline{(w = xyz)} \vee \overline{(1)} \vee \overline{(2)} \vee \overline{(3)})) \Rightarrow L \notin \mathsf{REG}.$$

Для получения «причёсанного вида» сделаем обратный переход от дизъюнкции к импликации выполнив замены:

$$\overline{(w = xyz)} \vee \overline{(1)} \vee \overline{(2)} \vee \overline{(3)} = \overline{((w = xyz) \wedge (1) \wedge (2))} \vee \overline{(3)} =$$

$$= ((w = xyz) \wedge (1) \wedge (2)) \Rightarrow \overline{(3)}. \qquad \square$$

К.В. 33. Лемма справедлива и для конечных языков. Если взять константу больше, чем длина самого длинного слова из L, то слов длиннее константы в L не найдётся. Условие накачки содержит импликацию (см. решение предыдущего К.В.). Поскольку слов, подходящих под условие посылки ($|w| \geqslant p$) нет, то импликация истинна. $\square$

К.В. 34. Нужно в процедуре склейки в качестве q взять начальное состояние (удалить нужно неначальное состояние). $\square$

К.В. 39. Если $i < 0$, выберем слово a^i, а при $i = 0$ выберем ε. $\square$

К.В. 40. 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 $i \leqslant 0$[10] 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 $L_{=i} = [\sigma^i]$. □

[10] $a^0 = \varepsilon$.

6. Построение РВ по НКА

В этом разделе мы закончим доказательство эквивалентности класса языков, заданных РВ, и класса языков, распознаваемых автоматами. Нам осталось только привести алгоритм построения РВ по НКА (и доказать его корректность). Мы приведём в этом разделе алгоритм, который опирается на правые языки, приведённые в предыдущем разделе (формула (3)). Начнём с решения задачи 46.

Решение. Напомним, что $q_x = \delta^*(q_0, x)$ — состояние, в которое ДКА попадает, обработав x из начального состояния. Формула

$$R_{q_x} = \bigcup_{a \in \Sigma} aR_{q_{xa}}$$

справедлива в случае, если состояние q_x непринимающее, а если же оно принимающее, то справедлива формула

$$R_{q_x} = \{\varepsilon\} \cup \bigcup_{a \in \Sigma} aR_{q_{xa}}.$$

Действительно, по определению правого языка $\varepsilon \in R_{q_x}$ если и только если состояние q_x принимающее. Каждое непустое слово $u \in R_{q_x}$ начинается с некоторой буквы, а потому $u = av$ и $v \in R_{q_{xa}}$. Значит, язык $R_{q_x} \setminus \{\varepsilon\}$ разбивается в объединение языков $aR_{q_{xa}}$. □

Из решения задачи следует, что правые языки удовлетворяют системе уравнений. Запишем эту систему для ДКА $\mathcal{A}$ на рис. 32.

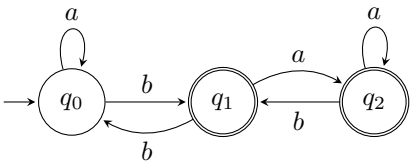


Рис. 32. Автомат $\mathcal{A}$

$$\begin{cases} R_{q_0} = aR_{q_0} + bR_{q_1}, \\ R_{q_1} = aR_{q_2} + bR_{q_0} + \varepsilon, \\ R_{q_2} = aR_{q_2} + bR_{q_1} + \varepsilon. \end{cases} \tag{1}$$

Мы использовали знаки «+» вместо знаков объединения для наглядности, как часто принято в формальных языках (это уже обсуждалось в разделе 1.2).

Алгоритм построения РВ по НКА состоит в решении системы уравнений (1). Осталось только сказать, что мы понимаем под решением, привести алгоритм решения и доказать его корректность.

Замечание 8. Хотя система и построена по полному ДКА, но сделано это было только для удобства обозначений. Правые языки имеют такое же определение для произвольных ДКА и НКА, а система для них строится аналогично. А именно: если состояние принимающее, то нужно добавить в правую часть ε, а в остальном, в правую часть для состояния q нужно добавить языки σR_p, такие что $q \xrightarrow{\sigma} p$, где $\sigma \in \Sigma \cup \{\varepsilon\}$.

Системе вида (1) могут удовлетворять многие наборы языков, в том числе и наборы, в которых есть нерегулярные.

Контрольный вопрос 41. Проверьте, что системе

$$\begin{cases} X = aX + Y, \\ Y = aY + X + \varepsilon. \end{cases}$$

удовлетворяют языки $X = Y = \{a^n b^k \mid n \geqslant k \geqslant 0\}$.

По этой причине, под *решением системы* с языками-переменными $X_1, \ldots, X_n$ мы будем понимать такой набор языков $A_1, \ldots, A_n$, удовлетворяющий системе, что для каждого набора языков $B_1, \ldots, B_n$, удовлетворяющего системе, выполнено $A_i \subseteq B_i$ для $1 \leqslant i \leqslant n$. То есть под решением системы мы будем понимать минимальное по включению решение. При доказательстве корректности ниже мы покажем, что правые языки образуют минимальное по включению решение.

Чтобы решить систему уравнений мы перейдём к более общей задаче.

6.1. Решение систем уравнений с регулярными коэффициентами

Под системой уравнений с регулярными коэффициентами мы понимаем следующую систему, в которой X_i — переменные-множества, а $\alpha_{i,j}$ и β_i — регулярные языки (коэффициенты):

$$\begin{cases} X_1 = \alpha_{11}X_1 + \alpha_{12}X_2 + \ldots + \alpha_{1n}X_n + \beta_1, \\ X_2 = \alpha_{21}X_1 + \alpha_{22}X_2 + \ldots + \alpha_{2n}X_n + \beta_2, \\ \quad \ldots \\ X_n = \alpha_{21}X_1 + \alpha_{22}X_2 + \ldots + \alpha_{2n}X_n + \beta_n. \end{cases} \tag{2}$$

Любая система уравнений вида (1), построенная по автомату, является системой уравнений с регулярными коэффициентами (языки α_{ij} и β_i могут быть пустыми).

В этом разделе мы приведём алгоритм решения системы (2) и решим с его помощью систему (1).

Алгоритм. Будем последовательно исключать переменные.

Случай 1. Если в системе есть строка вида $X_i = \alpha_{ii}X_i + \gamma$, $\alpha_{ii} \neq \varnothing$, то заменим всюду X_i на $\alpha_{ii}^*\gamma$, а саму строку удалим. Заметим, что γ может содержать другие переменные. После замены приведём систему к виду (2); в приведённой системе будет на одну переменную меньше.

Случай 2. Если в системе нет строк, подходящих под первый случай, то переменная X_n не встречается в правой части n-й строки, а значит после замены всюду переменной X_n на правую часть n-й строки переменная X_n исчезнет.

После того как все переменные будут исключены, нужно произвести последовательно подстановку полученных значений переменных в порядке, обратном к исключению.

Контрольный вопрос 42°. Решите систему уравнений (1).

Решение. Первая строка попадает под случай 1. Отсюда

$$R_{q_0} = aR_{q_0} + bR_{q_1} \Rightarrow R_{q_0} = a^*bR_{q_1}.$$

После подстановки получаем систему

$$\begin{cases} R_{q_1} = aR_{q_2} + ba^*bR_{q_1} + \varepsilon, \\ R_{q_2} = aR_{q_2} + bR_{q_1} + \varepsilon. \end{cases}$$

Опять исключим переменную из первой строки:

$$R_{q_1} = (ba^*b)^*aR_{q_2} + (ba^*b)^*.$$

После подстановки останется последнее уравнение:

$$R_{q_2} = (a + b(ba^*b)^*a)R_{q_2} + b(ba^*b)^* + \varepsilon,$$

из которого получаем, что

$$R_{q_2} = (a + b(ba^*b)^*a)^*(b(ba^*b)^* + \varepsilon).$$

Проведём подстановки полученных значений и найдём решение (минимальное по включению!):

$$R_{q_1} = (ba^*b)^*a(a + b(ba^*b)^*a)^*(b(ba^*b)^* + \varepsilon) + (ba^*b)^*,$$

$$R_{q_0} = a^*b(ba^*b)^*a(a + b(ba^*b)^*a)^*(b(ba^*b)^* + \varepsilon) + (ba^*b)^*.$$

□

Замечание 9. Заметив, что правый язык R_{q_0} совпадает с языком $L(\mathcal{A})$, получаем, что алгоритм решения системы уравнений с рациональными коэффициентами строит в процессе РВ для языка, распознаваемого НКА.

Замечание 10. Приведённый алгоритм построения РВ по НКА не полиномиальный. Полиномиального алгоритма не существует даже в случае построения ДКА по РВ над двухбуквенным алфавитом. В статье [12] доказана экспоненциальная нижняя оценка.

6.2. Корректность

Утверждение 12. *Построим по НКА $\mathcal{A}$ систему уравнений (2). Правые языки, соответствующие переменным, будут подмножествами любого набора языков $B_1, \ldots, B_n$, удовлетворяющего системе.*

Доказательство. Обозначим через X_q и B_q язык-переменную и множество из набора $B_1, \ldots, B_n$, соответствующие правому языку R_q.

Докажем по индукции следующее утверждение. Пусть $|w| = m$ и $w \in R_q$, тогда $w \in B_q$. Для $m = 0$ утверждение очевидно: q — принимающее состояние тогда и только тогда, когда ε есть в правой части уравнения, а потому $\varepsilon \in B_q$ при $q \in F$.

Докажем переход. Пусть $|w| = m > 0$ и $w \in R_q$. Тогда $w = av$ и $v \in R_p$, причём в НКА $\mathcal{A}$ есть переход $q \xrightarrow{a} p$. По предложению индукции $v \in B_p$, откуда получаем, что $w \in B_q$, поскольку в системе уравнений есть строка $X_q = aX_p + \ldots$ □

Итак, мы доказали, что правые языки образуют минимальное по включению решение системы, построенной по НКА. Осталось доказать корректность алгоритма решения уравнений с регулярными коэффициентами. Корректность второго случая алгоритма и этапа обратной подстановки сомнений не вызывают, поэтому сосредоточимся на первом случае.

Утверждение 13. *Минимальное по включению решение уравнения с регулярными коэффициентами*

$$X = \alpha X + \beta \tag{3}$$

есть $\alpha^*\beta$.

Доказательство. Пусть множество B удовлетворяет (3). Без сомнений $\beta \subseteq B$. Но если $w \in B$, то $\alpha w \subseteq B$ согласно (3). Отсюда и получаем, что $\alpha^*\beta \subseteq B$. С другой стороны ясно, что $\alpha^*\beta$ удовлетворяет уравнению (3): $\alpha^*\beta = \alpha(\alpha^*\beta) + \beta$. □

7. Заключение

Подытожим изученный материал. С помощью регулярных выражений удобно описывать спецификации (email адреса, почтовые индексы, имена переменных и т. п.), а конечные автоматы позволяют проверить, удовлетворяют ли данные (представленные в виде слов) описанной спецификации. Также с помощью конечных автоматов можно достаточно изящно описать эффективные алгоритмы поиска образцов в тексте (КМП и Ахо–Корасик).

Эта возможность неслучайна: детерминированные конечные автоматы реализуют онлайн алгоритмы, использующие конечную память. Онлайн-алгоритм — это алгоритм, который получает на вход последовательность данных $x_1, x_2, \ldots, x_n$; сколько элементов последовательности заранее неизвестно, и после обработки каждого элемента последовательности у алгоритма есть ответ на вопрос задачи на случай, если этот элемент последний. Например, онлайн-алгоритм, который ищет максимум на последовательности целых чисел, после обработки i-го числа должен хранить в памяти $\max(x_1, \ldots, x_i)$. Задачу, решаемую онлайн алгоритмом, можно представить как формальный язык. Например задаче о поиске вхождения образца w в текст соответствует язык $\Sigma^* w \Sigma^*$, состоящий из всех текстов, для которых ответ на вопрос задачи положительный.

Связь с НКА и замкнутость относительно различных операций позволяют доказать различные свойства таких алгоритмов. Например, если есть недетерминированный онлайн-алгоритм[1] с конечной памятью, решающий задачу, то есть и детерминированный. Правда последнему может потребоваться экспоненциально больше памяти, чем достаточно первому.

Замкнутость относительно теоретико-множественных операций, в частности, гарантирует существование онлайн-алгоритма (на конечной памяти — здесь и далее по тексту), решающего две задачи одновременно, если для каждой из задач есть онлайн-алгоритм. Замкнутость относительно операции обращения гарантирует удивительное свойство: если для задачи есть онлайн-алгоритм, то есть и онлайн-

[1] Мы не будем давать здесь описание недетерминированного алгоритма, полагаясь на интуицию читателя. Подробное объяснение недетерминизма можно найти, например, в [1], [3] или [4], [5].

алгоритм для задачи, в которой данные подаются в обратном порядке. В частности, если какое-то свойство чисел, записанных в k-ичной системе счисления, проверяемо онлайн-алгоритмом, то неважно, в каком порядке считывать запись числа: со старших или младших разрядов (см. задачу 44).

Нижние оценки, такие как в задаче 16, позволяют доказать, что некоторые задачи нельзя решить, используя меньшую память: чтобы проверить, что в последовательности $x_1, \ldots, x_n$ на позиции $n - i$ стоит единица, любой онлайн-алгоритм должен помнить все позиции, на которых стоят единицы, среди последних n обработанных.

Алгоритм минимизации позволяет не только построить самый эффективный онлайн-алгоритм, если есть хоть какой-то, но и сравнить, совпадают ли два онлайн-алгоритма или нет, что невозможно в общем случае: проверка эквивалентности программ — алгоритмически неразрешимая задача.

Теорема Майхилла–Нероуда даёт простой критерий проверки регулярности языка, что в терминах онлайн-алгоритмов значит, что по описанию задачи можно установить разрешима ли она онлайн-алгоритмом или нет.

И наконец, алгоритм построения РВ по НКА позволяет получить описание спецификации в виде РВ, если задан алгоритм проверки спецификации.

Эти свойства несомненно хороши для практики. Увы, лишь немногие из них выживают при усилении модели вычислений. Если к конечному автомату добавить стек, то получится магазинный автомат. Недетерминированные магазинные автоматы распознают контекстно-свободные языки, а детерминированные — их детерминированный подкласс. Последний широко используется на практике, в частности, в компиляторах. Однако многие вопросы, связанные с КС-языками алгоритмически неразрешимы, а алгоритм проверки эквивалентности детерминированных МП-автоматов был получен относительно недавно и его автор был удостоен премии Гёделя.

Автор надеется, что общая картина, описанная в заключении, побудит студентов, ориентированных на практику и рассматривающих ТРЯП как введение в теорию компиляторов, которые они никогда писать не планируют, пересмотреть своё отношение.

8. Задачи по всем темам

50°. Постройте для РВ $a^*(b|bb)(a^*bb^*|b^*)^*$ над алфавитом $\{a, b\}$ эквивалентный минимальный полный ДКА $\mathcal{A}$.

51. Язык R состоит из всех слов, в которых после каждой буквы a далее встречается (не обязательно следующим символом) буква b и нет двух b подряд. Запишите регулярное выражение или постройте конечный автомат для языка R.

52. Постройте конечный автомат, распознающий все слова над алфавитом $\{a, b\}$, в которые в качестве подслова входит ровно одно из слов aab, ba.

53. Язык L_1 объединили с конечным языком R и получили язык L ($L = L_1 \cup R$). Язык L оказался регулярным. Верно ли, что язык L_1 мог быть нерегулярным?

54. Верно ли, что если пересечение двух языков $L_1,\ L_2 \subseteq \{a, b\}^*$ содержит язык $F = \{a^n b^n \,|\, n \geqslant 1\} : F \subseteq L_1 \cap L_2$, то языки L_1 и L_2 нерегулярные?

55. Пусть R регулярный язык. Верно ли, что если языки $F \cap R$ и $F \cap \bar{R}$ являются регулярными, то и F тоже регулярный язык?

56. Верно ли, что язык, образованный конкатенацией нерегулярного языка L и регулярного языка R, является нерегулярным языком?

57. Верно ли, что для любых бесконечных регулярных языков X и Y, отличных от языка всех слов, язык

$$L = \{xy \mid x \in X, y \in Y, |x| = |y|\}$$

является нерегулярным?

58. Язык R называется ограниченным, если существует такой набор слов $w_1, w_2, \ldots, w_n$, что $R \subseteq w_1^* w_2^* \ldots w_n^*$. Верно ли, что

а) любой конечный язык является ограниченным;

б) если регулярные языки A и B ограниченные, то и язык $A \cdot B$ является ограниченным;

в) если регулярные языки A и B ограниченные, то и язык $A \cup B$ является ограниченным;

г) если регулярные языки A и B ограниченные, то и язык $A \cap B$ является ограниченным?

59. Пусть R_1 и R_2 бесконечные регулярные языки, $R_1 \neq R_2$ и при этом $R_1 \subseteq L \subseteq R_2$. Верно ли, что L — регулярный язык?

60. Для языка L над алфавитом Σ определим множество префиксов слов из языка L:

$$\mathrm{PREF}(L) = \{x \mid \exists y : xy \in L\}.$$

Верно ли, что множество $\mathrm{PREF}(R)$ является регулярным языком для любого регулярного языка R?

61. Для языка L над алфавитом Σ определим множество суффиксов слов из языка L:

$$\mathrm{SUFF}(L) = \{y \mid \exists x : xy \in L\}.$$

Верно ли, что множество $\mathrm{SUFF}(R)$ является регулярным языком для любого регулярного языка R?

62. Для языка $L = \{w \mid |w|_a + |w| = 2 \mod 3\}$ найти классы эквивалентности Майхилла–Нероуда и построить минимальный полный ДКА, распознающий язык L.

63[1]**.** Пусть $\Sigma = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$. Определим функцию $go\colon \Sigma^* \to (\mathbb{Z} \times \mathbb{Z})$, соответствующую операции перемещения в двумерной целочисленной плоскости из точки $(0, 0)$ в финальную точку маршрута, заданного последовательностью стрелок (словом в алфавите Σ).

Например $go(\rightarrow\uparrow\rightarrow\uparrow\rightarrow) = (3, 2)$, $go(\leftarrow\uparrow\rightarrow\downarrow) = (0, 0)$.

Язык L состоит из слов-путей w из точки $(0, 0)$ в $(6, 5)$ (т. е. $go(w) = (6, 5)$), не выходящих за пределы квадрата $[0, 10] \times [0, 10]$ и не пересекающих прямую $y = x + 1$.

[1] Задача составлена совместно с Д. Лещёвым.

1. Является ли язык L регулярным?

2. Опишите классы L-эквивалентности (Майхилла–Нероуда).

Замечание 11. Факт того, что описанные классы являются классами Майхилла–Нероуда для языка L, требует доказательства!

64. 1. Является ли язык $L_a = \{w : |w|_a \geqslant |w|/3, w \in \Sigma^*\}$ над алфавитом $\Sigma = \{a, b\}$ регулярным языком?

2. Является ли язык $L = L_a \cup \{w : |w|_b \geqslant |w|/3, w \in \Sigma^*\}$ регулярным?

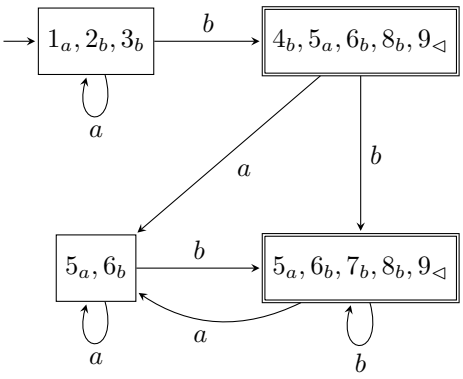
8.1. Ответы и решения

З. 50. Занумеруем все символы, входящие в РВ, и укажем результаты вычисления функции followpos, после чего построим получившийся по алгоритму «РВ → ДКА» автомат и соответствующий минимальный автомат.

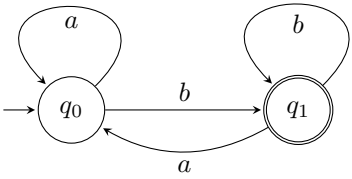
$$a_1^*(b_2|b_3b_4)(a_5^*b_6b_7^*|b_8^*)^* \lhd_9$$

№	Followpos
1	1,2,3
2	5,6,8,9
3	4
4	5,6,8,9
5	5,6
6	5,6,7,8,9
7	5,6,7,8,9
8	5,6,8,9

Автомат по РВ:



Минимальный автомат:



□

Литература

1. *Sipser M.* Introduction to the Theory of Computation. 1st edn. International Thomson Publishing, 1996.
2. *Хопкрофт Д., Мотвани Р., Ульман Д.* Введение в теорию автоматов, языков и вычислений. Москва : Вильямс, 2002.
3. *Дасгупта С., Пападимитриу Х., Вазирани У.* Алгоритмы. Москва : МЦНМО, 2012.
4. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. Москва : МЦНМО, 2002.
5. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. 2-е изд. Москва : Вильямс, 2005.
6. *Верещагин Н.К., Шень А.* Лекции по математической логике и теории алгоритмов. Начала теории множеств. 4-е изд. Москва : МЦНМО, 2012.
7. Лекции по дискретной математике [Электронный ресурс]: черновик лекций / М. Вялый, В. Подольский, А. Рубцов, Д. Шварц, А. Шень. ФКН ВШЭ, 2018. URL: http://rubtsov.su/public/DM-HSE-Draft.pdf
8. Теория и реализация языков программирования / В.А. Серебряков, М.П. Галочкин, Д.Р. Гончар, М.Г. Фуругян. Москва : МЗ-Пресс, 2006.
9. Analyzing Matching Time Behavior of Backtracking Regular Expression Matchers by Using Ambiguity of NFA / N. Weideman, B. van der Merwe, M. Berglund, B. Watson // Implementation and Application of Automata / eds. Y.-S. Han, K. Salomaa. Cham: Springer International Publishing, 2016. P. 322–334. ISBN 978-3-319-40946-7.
10. *Шень А.* Программирование: теоремы и задачи. 2-е изд. Москва : МЦНМО, 2004.
11. *Журавлёв Ю.И., Флёров Ю.А., Федько О.С.* Дискретный Анализ. Комбинаторика. Алгебра логики. Теория графов. Москва : МФТИ, 2012.
12. *Gruber H., Holzer M.*, Finite Automata, Digraph Connectivity, and Regular Expression Size // Proceedings of the 35th International Colloquium on Automata, Languages and Programming. Cham: Springer International Publishing, 2008. P. 39—50.