Теоретическая информатика, осень 2020 г. Лекция 3. Минимальные DFA. Алгоритм минимизации DFA. Двухсторонние конечные автоматы (2DFA). Преобразование 2DFA к DFA*

Александр Охотин

18 сентября 2020 г.

Содержание

1 Минимальные DFA

Задача нахождения DFA с наименьшим числом состояний, распознающего тот же язык, что и данный DFA: *минимизация DFA*. Будет показано, что минимальный DFA всегда единственен, а также приведён алгоритм для его построения.

Если для каждого состояния DFA рассмотреть множество всех строк, принимаемых, начиная из этого состояния, и если для двух состояний эти множества совпадают, то эти два состояния взаимозаменяемы, и их можно объединить в одно.

Для DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, множество строк, принимаемых из состояния $q \in Q$, обозначается через $L_{\mathcal{A}}(q) = \{\, w \mid \delta(q, w) \in F \,\}$.

Лемма 1. *Пусть в DFA $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, начиная из двух состояний, q и q', принимается одно и то же множество строк: $L_{\mathcal{A}}(q) = L_{\mathcal{A}}(q')$, где $q \neq q'$. Пусть $q \neq q_0$. Тогда автомат $\mathcal{B} = (\Sigma, Q \setminus \{q'\}, q_0, \delta', F \setminus \{q'\})$, полученный удалением q' и перенаправлением всех переходов, ведущих в q', в состояние q, распознаёт тот же язык.*

Если же эти множества не совпадают, то существует строка, принадлежащая одному из них и не принадлежащая другому — *разделяющая строка*.

*Краткое содержание лекций, прочитанных студентам 2-го курса факультета МКН СПбГУ в осеннем семестре 2020–2021 учебного года. Страница курса: `http://users.math-cs.spbu.ru/~okhotin/teaching/tcs_fl_2020/`.

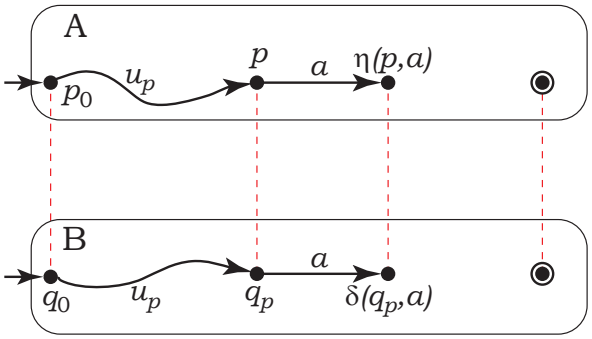

Рис. 1: Построение иоморфизма между двумя минимальными DFA для одного и того же языка.

Лемма 2. *Пусть $L \subseteq \Sigma^*$ — регулярный язык. Если для двух строк $u, v \in \Sigma^*$ существует такая строка $w \in \Sigma^*$, что одна из строк uw, vw принадлежит языку L, а другая не принадлежит, то во всяком DFA, распознающем L, состояния $\delta(q_0, u)$ и $\delta(q_0, v)$ должны быть различны.*

Доказательство. Если такие два состояния совпадут, то автомат или примет обе строки uw, vw, или обе отвергнет. □

С помощью этого нехитрого свойства, например, можно доказывать нижние оценки числа состояний DFA, распознающих данный язык L: если построить такое множество $\{u_1, \ldots, u_n\}$, что для всякой пары различных строк из него найдётся разделяющая строка, то у всякого автомата, распознающего L, должно быть не менее n состояний.

Два DFA *изоморфны*, если между их множествами состояний есть взаимно однозначное отображение, сохраняющее переходы и сопоставляющее начальное состояние начальному и принимающие — принимающим.

Теорема 1. *Минимальный DFA для данного регулярного языка единственен с точностью до изоморфизма.*

Доказательство. Пусть $\mathcal{A} = (\Sigma, P, p_0, \eta, E)$ и $\mathcal{B} = (\Sigma, Q, q_0, \delta, F)$ — два минимальных DFA, распознающие один и тот же язык L. Нужно построить взаимно однозначное соответствие между P и Q, сохраняющее переходы.

Для всякого состояния первого автомата, $p \in P$, пусть u_p — кратчайшая строка, по которой оно достижимо (если состояние недостижимо, его можно удалить, и тогда автомат не был бы минимальным). Тогда из состояния p принимается язык $L_{\mathcal{A}}(p) = \{\, v \mid u_p v \in L \,\}$. В силу минимальности $\mathcal{A}$, никакие два из этих языков не совпадают — иначе соответствующие состояния можно было бы объединить по лемме 1.

Читая эту же строку u_p, автомат $\mathcal{B}$ приходит в некоторое состояние q_p, из которого принимается тот же самый язык $L_{\mathcal{B}}(q_p) = \{\, v \mid u_p v \in L \,\}$. Эти языки также попарно не совпадают. Тогда состоянию p ставится в соответствие состояние q_p, и, в частности, p_0 отображается в q_0.

Поскольку у $\mathcal{B}$ столько же состояний, сколько и у $\mathcal{A}$, никаких других состояний у него нет, и получено взаимно однозначное соответствие между P и Q. Это соответствие сохраняет переходы: а именно, если p соответствует q, то состояния $p' = \eta(p, a)$ и $q' = \delta(q, a)$

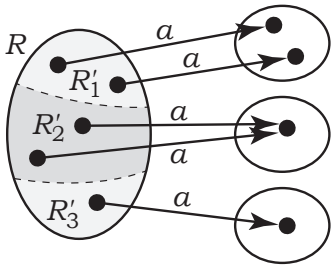

Рис. 2: Разбиение класса R на три класса на основе переходов по символу a.

также должны соответствовать друг другу. Действительно, из каждого из этих состояний должен приниматься язык $\{ v \mid u_p av \in L \}$, и в каждом из автоматов есть только одно такое состояние. □

2 Алгоритм минимизации DFA

Для данного DFA, лемма 2 определяет отношение эквивалентности на множестве Q.

Алгоритм Мура [1956] для минимизации автоматов находит это отношение, разбивая Q на классы эквивалентности. Это делается путём определения некоторых неэквивалентных пар состояний.

Сперва, если одно состояние — принимающее, а другое — отвергающее, то они, очевидно, неэквивалентны (ведь из одного $w = \varepsilon$ принимается, а из другого отвергается). Поэтому в качестве начального разбиения берутся два класса эквивалентности: $R_0 = Q \setminus F$, $R_1 = F$.

Алгоритм поддерживает следующий инвариант: для текущего разбиения Q на непересекающиеся классы $R_0, \ldots, R_{m-1}$ верно, что для каждой пары классов R_i, R_j существует строка w, которая принимается из всех состояний одного класса и ни из одного из состояний другого класса.

На каждом шаге просматриваются все имеющиеся классы, и для каждого класса R и для всякого символа $a \in \Sigma$, рассматриваются переходы из каждого состояния из R по a. Если все эти переходы ведут в состояния из одного и того же класса, то никаких противоречий не обнаружено, оснований разбивать класс R нет. Если же, однако, по a из R можно перейти в разные классы, то пусть $R = R'_1 \cup \ldots \cup R'_k$, где все переходы из R'_j по a идут в один из существующих классов R_j. Тогда R заменяется на классы $R'_1, \ldots, R'_k$, как показано на рис. 2. Действительно, если классы R_{j_1} и R_{j_2} различаются по некоторой строке w, то R'_{j_1} и R'_{j_2} различаются по строке aw.

Это продолжается, пока дальнейшее измельчение классов возможно. Когда далее измельчать становится некуда, строится новый автомат, состояниями которого станут построенные классы эквивалентности.

Теорема 2 (Мур [1956])**.** *Алгоритм 1 строит автомат, распознающий тот же язык, что и исходный, и минимальный среди всех автоматов, распознающих этот язык. Алгоритм работает за время $O(|\Sigma| \cdot n^2)$, где n — число состояний в исходном автомате, а Σ — алфавит.*

Доказательство. **(тот же язык)** Обозначение: $[q]$ — класс эквивалентности, в который попало состояние q. Поскольку алгоритм завершил измельчения классов, выполняется условие, что $\delta(p, a) = r$, то $\delta'([p], a) = [r]$. Тогда, для всякой входной строки, всякому вычислению

Алгоритм 1 Минимизация DFA, алгоритм Мура [1956]

Дано: DFA $(\Sigma, Q, q_0, \delta, F)$.
В каждый момент времени алгоритм помнит разбиение Q на классы эквивалентности $R_0, \ldots, R_{m-1}$.

```
1: Пусть R_0 = Q \ F, R_1 = F
2: while можно разделить какой-то класс do
3:     for all класс R do
4:         for all символ a ∈ Σ do
5:             for all q ∈ R do
6:                 Определить класс, в котором находится δ(q, a)
7:             if эти классы не все совпадают then
8:                 Пусть R = R'_1 ∪ ... ∪ R'_k, где переходы по a из каждого R'_j идут
                   в один класс
9:                 Заменить класс R на классы R'_1, ..., R'_k
```

Пусть $R_0, \ldots, R_{m-1}$ — полученные классы. Алгоритм выдаёт DFA $(\Sigma, \{R_0, \ldots, R_{m-1}\}, [q_0], \delta', \{\, R_i \mid R_i \subseteq F \,\})$, где $[q_0]$ — это класс, в который попало состояние q_0, а $\delta'(R_i, a) = [\delta(r, a)]$ для $r \in R_i$ — класс, в который ведут все переходы по a из состояний из R_i.

исходного автомата соответствует вычисление нового, проходящее через соответствующие классы эквивалентности.

$$q_0, q_1, q_2, \ldots$$
$$[q_0], [q_1], [q_2], \ldots$$

Поскольку каждый класс состоит целиком из принимающих или целиком из отвергающих состояний, в конце выдаётся одно и то же решение.

Отсюда следует, что строка w принимается исходным автоматом тогда и только тогда, когда её принимает новый автомат.

(минимальность) Пусть p и q — состояния исходного DFA, попадающие в разные классы эквивалентности. Утверждается, что есть строка, принимаемая из одного из них и отвергаемая из другого. Индукция по номеру шага алгоритма, на котором эти два состояния будут распределены в различные классы.

Базис: в разных классах с самого начала. Тогда одно из них принимающее, а другое — отвергающее, и разделяющая строка — ε.

Переход: пусть на каком-то шаге оба состояния p, q принадлежат некоторому классу R, рассматриваются переходы из R по a, и выясняется, что $[\delta(p, a)] \neq [\delta(q, a)]$. Тогда состояния $\delta(p, a)$ и $\delta(q, a)$ попали в различные классы ещё прежде, и потому, по предположению индукции, есть строка w, принимаемая из одного из них и не принимаемая из другого. Тогда строка aw разделяет p и q.

Отсюда, согласно лемме 2, классы $[p]$ и $[q]$ должны быть различными состояниями построенного автомата.

Время: $O(|\Sigma| \cdot n^2)$, так как число шагов измельчения — $O(n)$, и для каждого надо просмотреть $O(n)$ состояний и вычислить переходы по каждому символу. □

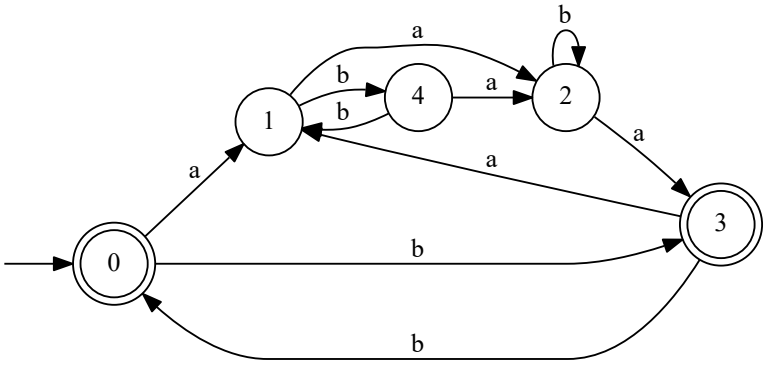

Рис. 3: Неминимальный DFA из примера 1.

Пример 1. *Автомат с 5 состояниями, приведённый на рис. 3, распознаёт язык* $\{ w \mid w \in \{a,b\}^*, |w|_a \equiv 0 \pmod 3 \}$*, хотя с виду этого не скажешь. Но стоит его минимизировать, как всё становится ясно.*

Алгоритм начинает с разбиения $R_0 = \{1,2,4\}$*,* $R_1 = \{0,3\}$*, приведённого на рис. 4(левом верхнем). Класс* R_1 *разбить не удаётся ни по* a*, ни по* b*, а вот класс* R_0 *разбивается по* a*, поскольку из состояний 1 и 4 переходы по* a *ведут в* R_0 *же, а переход по* a *из состояния 2 уходит в* R_1*. Поэтому* R_0 *заменяется на* $R_2 = \{1,4\}$ *и* $R_3 = \{2\}$*, полученное разбиение дано на рис. 4(правом верхнем). Ничего больше разбить не удаётся, так что получается минимальный автомат с 3 состояниями на рис. 4(нижнем).*

Как показали Бассино и др. [2012], если случайно выбрать автомат с n состояниями, то математическое ожидание времени работы алгоритма 1 будет порядка $n \log n$.

Известен более эффективный алгоритм минимизации DFA — алгоритм Хопкрофта [1971], работающий за время $O(|\Sigma| \cdot n \log n)$. Алгоритм Хопкрофта точно так же последовательно расщепляет классы эквивалентности, но за счёт хитроумных структур данных он меньше тратит времени на просмотр классов, которые расщепить не получается или пока не получается.

Для задачи минимизации NFA полиномиальный по времени алгоритм неизвестен. Вычислительная сложность этой задачи будет рассмотрена далее в курсе.

Применение минимизации DFA к распознаванию свойств автоматов. Как проверить для двух DFA, задают ли они один и тот же язык? Сперва минимизировать оба. Если вышло разное число состояний, то языки разные. Иначе пытаться построить взаимно однозначное соответствие между множествами состояний, начиная с начальных состояний и далее следуя функциям переходов — это делается за линейное время.

Другая задача: проверить, задаёт ли данное регулярное выражение язык Σ^*. Перевести в DFA с экспоненциальным ростом, минимизировать, и затем проверить, получился ли автомат с одним принимающим состоянием. Далее в курсе будет показано, что более эффективные алгоритмы неизвестны.

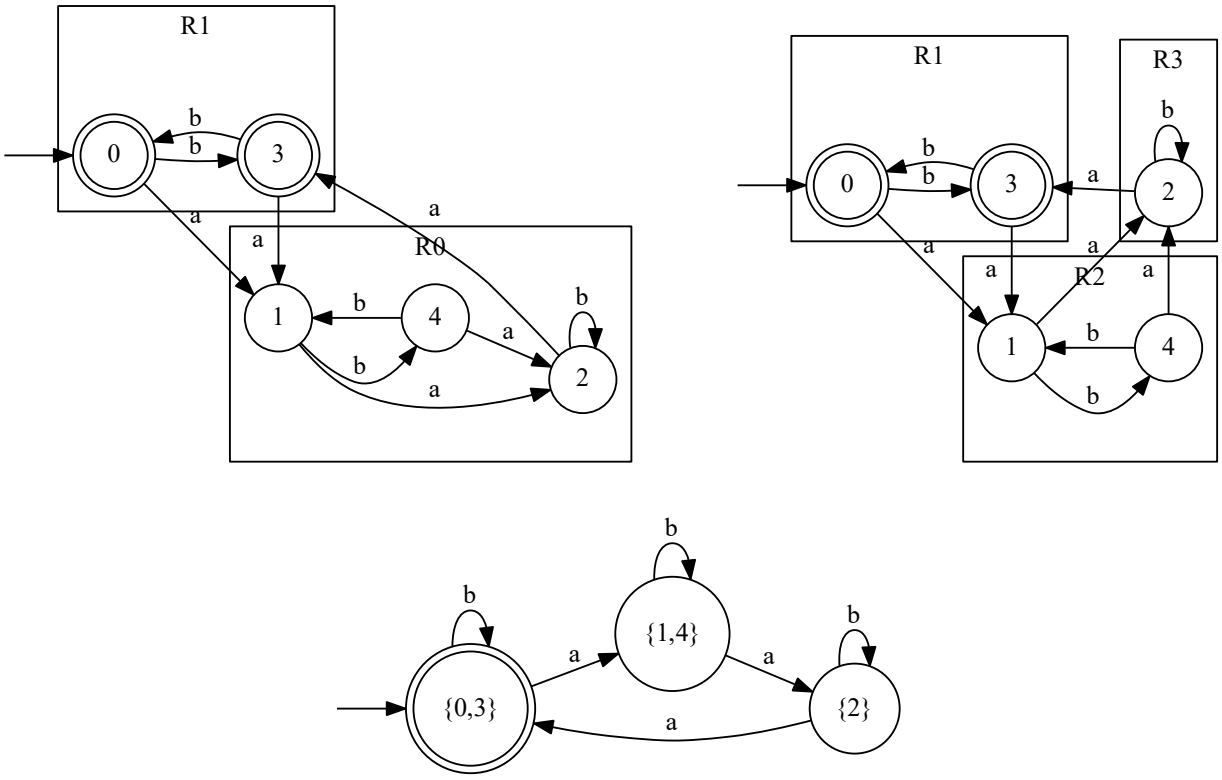

Рис. 4: Минимизация автомата из примера 1: (сверху слева) начальное разбиение; (сверху справа) измельчённое разбиение; (снизу) соответствующий минимальный DFA.


Рис. 5: Эдвард Мур (1925–2003); Джон Хопкрофт (род. 1939).

3 Двухсторонние конечные автоматы (2DFA)

Определение 1 (Рабин и Скотт [1959]). *Двухсторонний детерминированный конечный автомат (2DFA: two-way deterministic finite automaton) — это пятёрка* $A = (\Sigma, Q, q_0, \delta, F)$, *состоящая из следующих компонентов.*

- Σ — *алфавит, не содержащий двух особых символов* $\vdash, \dashv \notin \Sigma$ *(меток начала и конца строки).*
- Q — *конечное множество состояний.*
- $q_0 \in Q$ — *начальное состояние.*
- $\delta\colon Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{-1, +1\}$ — *частичная функция переходов.*
- $F \subseteq Q$ — *множество* ***принимающих состояний****, применяемых на правой метке конца* $(\dashv)$.

Для входной строки $w = a_1 \dots a_\ell \in \Sigma^*$, *пусть* $a_0 = \vdash$ *и* $a_{\ell+1} = \dashv$. *Вычисление 2DFA на* w *— это самая длинная, возможно бесконечная, последовательность* $(p_0, i_0), \dots, (p_j, i_j), \dots$, *где:*

- $p_j \in Q$ *и* $0 \leqslant i_j \leqslant \ell + 1$ *на каждом* j*-м шаге;*
- *начальная конфигурация —* $(p_0, i_0) = (q_0, 0)$*;*
- *всякая следующая конфигурация* (p_j, i_j), *если она определена, удовлетворяет* $\delta(p_{j-1}, a_{i_{j-1}}) = (p_j, d_j)$ *и* $i_j = i_{j-1} + d_j$.

Вычисление всегда определено однозначно. Строка w ***принимается****, если вычисление конечно и его последняя конфигурация —* $(p_f, \ell{+}1)$, *для некоторого* $p_f \in F$. *Также вычисление может заканчиваться неопределённым переходом (ведь* δ — частичная *функция) или же продолжаться бесконечно; и в том и в другом случае* A *не принимает* w.

Язык, распознаваемый автоматом — $L(A) = \{\, w \mid A \text{ принимает } w \,\}$.

Вычисление 2DFA на данной строке w удобно представлять в виде графа, множество вершин которого, $V = Q \times \{0, 1, \dots, |w|, |w|+1\}$ — это множество конфигураций автомата, и из каждой вершины исходит не более одной дуги, соответствующей переходу в этой конфигу-

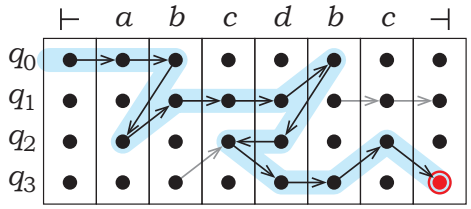

Рис. 6: Принимающее вычисление 2DFA.

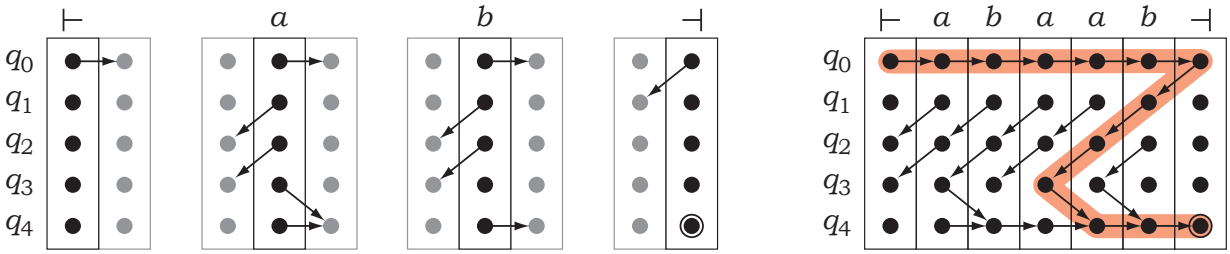

Рис. 7: 2DFA из примера 2: переходы по символам и по меткам начала и конца; вычисление на строке $w = abaab$, а также переходы из всех возможных конфигураций.

рации. Тогда вычисление — это путь, начинающийся в вершине $(q_0, 0)$. На рис. 6 показано принимающее вычисление.

Пример 2. *Язык* $(a \mid b)^* a (a \mid b)^{n-1}$ *распознаётся 2DFA со множеством состояний* $Q = \{q_0, q_1, \ldots, q_n, q_{n+1}\}$. *Сперва автомат едет в конец в состоянии* q_0.

$$\delta(q_0, \vdash) = \delta(q_0, a) = \delta(q_0, b) = (q_0, +1)$$

Затем, доехав до метки конца, он разворачивается и отсчитывает n *символов.*

$$\begin{aligned} \delta(q_0, \dashv) &= (q_1, -1) \\ \delta(q_i, a) = \delta(q_i, b) &= (q_{i+1}, -1) \qquad (i \in \{1, \ldots, n-1\}) \end{aligned}$$

Если n*-й символ с конца окажется символом* a*, то автомат вновь отправится направо к метке конца, чтобы там принять.*

$$\begin{aligned} \delta(q_n, a) &= (q_{n+1}, +1) \\ \delta(q_{n+1}, a) = \delta(q_{n+1}, b) &= (q_{n+1}, +1) \end{aligned}$$

Множество принимающих состояний — $F = \{q_{n+1}\}$.

Если строка — не требуемого вида, то в ходе описанного вычисления автомат натолкнётся на неопределённый переход.

Переходы по $\vdash$, a, b *и* $\dashv$ *в автомате, построенном для* $n = 3$, *показаны на рис. 7. Также приведено принимающее вычисление автомата на строке* $w = abaab$, *в котором указаны переходы из всех возможных конфигураций на этой входной строке.*

Рис. 8: Джон Шепердсон (1926–2015) и Христос Капуцис (род. 1974).

4 Преобразование 2DFA к DFA

Оказывается, что возможность ходить по ленте туда и обратно не даёт двухстороннему автомату возможности распознать какой-нибудь регулярный язык.

Теорема 3 (Рабин и Скотт [1959]; Шепердсон [1959])**.** *Для всякого 2DFA существует DFA, распознающий тот же язык.*

Лемма 3 (Капуцис [2005])**.** *Для всякого 2DFA с n состояниями существует DFA с $n(n^n - (n-1)^n) + 1$ состояниями, распознающий тот же язык.*

Доказательство. Выехав с метки начала ($\vdash$), 2DFA какое-то время движется направо; ничего не мешает DFA точно так же повторить его вычисление. Но потом 2DFA может развернуться и поехать налево, вновь обследуя символы, которые DFA уже прочитал и забыл. Как одностороннему автомату проследить это вычисление? Если знать наперёд, в каких состояниях 2DFA поедет назад, то DFA мог бы заранее проследить его обратный путь, читая строку в первый и единственный раз. Но узнать это заранее неоткуда — тем более, что 2DFA может сновать взад-вперёд по одним и тем же символам по сложной траектории. С другой стороны, для DFA неважно, какие именно зигзаги 2DFA выписывал по уже прочитанным символам — важно, вернётся ли 2DFA когда-нибудь к последнему прочитанному символу и перейдёт ли с него направо на следующий символ; если же это случится, важно знать, в каком состоянии он туда перейдёт.

Чтобы всё это проследить, DFA, прочитав строку u, будет помнить исход *всех возможных вычислений 2DFA на $\vdash u$, начинающихся в начальной конфигурации или на последнем символе строки u, и заканчивающихся выходом за пределы u направо.* Возможных начальных и конечных конфигураций конечное число, и потому можно запомнить исходы всех таких вычислений в состоянии построенного DFA. Сведения о просчитанных вычислениях на более короткой строке можно использовать для определения исхода вычислений на продолжении этой строки, и к моменту прочтения всей входной строки DFA вычислит, в числе прочего, итог искомого вычисления, начинающегося в начальной конфигурации.

Пусть $(\Sigma, Q, \delta, q_0, F)$ — 2DFA. Состояния DFA имеют вид $(\widehat{q}, f)$, где $\widehat{q} \in Q$ — состояние

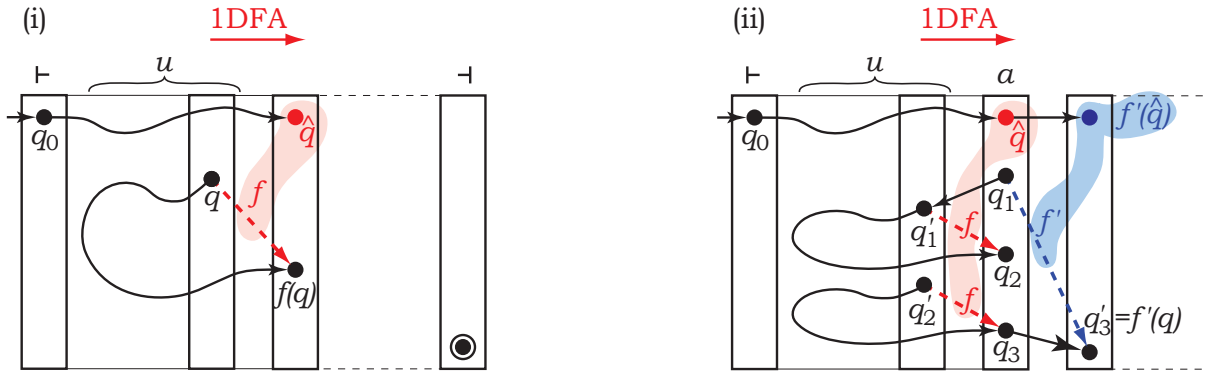

Рис. 9: Доказательство леммы 3: (i) сведения о поведении 2DFA на начальном участке строки, которые DFA запоминает в своём состоянии $(\widehat{q}, f)$; (ii) как DFA преобразует сведения для строки u в подобные же сведения для строки ua при переходе по символу a.

2DFA, а $f\colon Q \to Q$ — частичная функция, отображающая состояния в состояния[1]. На входной строке $u \in \Sigma^*$, получаемый DFA достигает состояния $(\widehat{q}, f)$, для которого

I. в вычислении 2DFA на $\vdash u$, когда он впервые переходит из самого правого символа $\vdash u$ направо, он переходит в состояние $\widehat{q}$;

II. если 2DFA начинает своё вычисление на самом правом символе $\vdash u$ в состоянии q, то он со временем переходит с этого самого правого символа направо в состоянии $f(q)$; если же это вычисление 2DFA отвергает или зацикливается, то $f(q)$ определяется как $\widehat{q}$: эта функция описывает *поведение* 2DFA на этом начальном участке строки.

Условия, определяющие состояние $(\widehat{q}, f)$, показаны на рис. 9(i).

Остаётся определить DFA, вычисляющий эти данные. Его начальное состояние — пара (q_0', f_0), где q_0' определяется переходом $\delta(q_0, \vdash) = (q_0', +1)$ (по этому переходу автомат выйдет направо из пустой строки) а функция f_0 записывает все переходы по метке начала.

$$f_0(q) = \begin{cases} q', & \text{если } \delta(q, \vdash) = (q', +1) \\ \text{не определено}, & \text{в противном случае} \end{cases}$$

Определение функции переходов показано на рис. 9(ii). Пусть DFA, прочитав префикс w входной строки, находится в состоянии $(\widehat{q}, f)$ — то есть, вычислил сведения о вычислениях 2DFA на u, удовлетворяющие условиям I–II. Тогда на следующем шаге DFA читает следующий символ $a \in \Sigma$ и должен вычислить такие же сведения о вычислениях 2DFA на строке ua. Для этого DFA переходит из состояния $(\widehat{q}, f)$ по символу a в состояние $(f'(\widehat{q}), f')$, где значение f' для каждого состояния $q_1 \in Q$ определяется так.

Пусть 2DFA начал работу на последнем символе строки ua в состоянии q_1.

- Тогда, если переход в q_1 по a перемещает головку направо, то есть, $\delta(q_1, a) = (q_1', +1)$, то значение $f'(q_1)$ определяется как q_1'.

- Если же переход ведёт налево, то есть, $\delta(q_1, a) = (q_1', -1)$, то автомат перейдёт к последнему символу u в состоянии q_1'. О вычислении, начинающегося с этого места, известно, что через какое-то время оно перейдёт обратно на символ a в состоянии $q_2 = f(q_1')$. Далее рассматривается переход в состоянии q_2 по a, и если он ведёт к перемещению головки направо в состоянии q_2', то значение $f'(q_1)$ можно определить

[1] Идея использования такой функции принадлежит Шепердсону [1959], но лишь Капуцис [2005] определил необходимое количество этих функций, а также их области определения и значений.

как q'_2; если же головка идёт налево, то продолжение вычисления выясняется далее по тому же образцу.

Так определяется последовательность состояний $q_1, \ldots, q_k, \ldots$, для $k \geqslant 0$, где $\delta(q_i, a) = (q'_i, -1)$ и $q_{i+1} = f(q'_i)$ для всех $i \in \{1, \ldots, k\}$. Есть три возможности: во-первых, эта последовательность может быть бесконечной, что означает, что 2DFA зацикливается, возвращаясь к символу a вновь и вновь в тех же состояниях; в этом случае значение $f(q_1)$ не определено. Во-вторых, последовательность может закончиться, натолкнувшись на неопределённое значение f или на неопределённый переход $\delta(q_i, a)$; тогда $f(q_1)$ также остаётся неопределённым. В-третьих, автомат может однажды придти на a в очередном состоянии q_{k+1}, после чего перейти направо по переходу $\delta(q_{k+1}, a) = (q'_{k+1}, +1)$; и тогда $f'(q_1)$ определяется как q'_{k+1}.

Первый компонент пары определяется применением полученной функции f' к состоянию $f'(\widehat{q})$. Если это значение остаётся неопределённым, то DFA переходит в особое мёртвое состояние, в котором дочитывает строку до конца и отвергает.

Принимающие состояния DFA определяются аналогично определению значения функции f' в точке $\widehat{q}$. А именно, для состояния $(\widehat{q}, f)$ одностороннего автомата рассматривается состояние $q_1 = \widehat{q}$ исходного автомата, в котором он впервые приходит к метке конца ($\dashv$). Для этого состояния вычисляется последовательность состояний $q_1, \ldots, q_k, \ldots$, где на каждом шаге 2DFA или перемещается с метки конца налево, или натыкается на неопределённый переход, или же приходит в принимающее состояние. Если последовательность заканчивается принимающим состоянием 2DFA, то в DFA состояние $(\widehat{q}, f)$ помечается как принимающее.

Правильность построения доказывается индукцией по длине строки. Приведённое выше описание работы DFA — это набросок такого доказательства.

Подсчёт числа состояний. Приведённое выше множество состояний DFA было построено с некоторым запасом, чтобы его легче было объяснять. Чтобы установить точный вид требуемых состояний, необходимо заметить следующие две вещи. Во-первых, нет нужды использовать частичную функцию f: если значение $f(q)$ по смыслу неопределённое, то его можно перенаправить в состояние $\widehat{q}$, и тогда при определении переходов вместо неопределённого значения будет находиться бесконечный цикл, с теми же последствиями. Тогда $f\colon Q \to Q$ — это полностью определённая функция. Во-вторых, образ f должен содержать состояние $\widehat{q}$, поскольку оно достигается при переходе с предыдущего символа в основном вычислении 2DFA. Поэтому из общего числа функций n^n можно вычесть $(n-1)^n$ функций, образ которых не содержит этого состояния. Таких пар $(\widehat{q}, f)$ ровно $n(n^n - (n-1)^n)$, и ещё используется одно мёртвое состояние. □

Пример 3. *Построение из теоремы 3, применённое к 2DFA из примера 2 (рис. 7), даёт DFA, приведённый на рис. 10.*

Каждое состояние вида $(\widehat{q}, f)$ представлено картинкой, на которой всякое значение $f(q) = q'$ обозначается стрелкой из состояния q в правом столбце в состояние q' в левом столбце, а состояние $\widehat{q}$ помечено звёздочкой. Иными словами, в левом столбце приведены состояния, в которых 2DFA начинает вычисление на последнем символе прочитанного участка строки, а в правом столбце — состояния, в которых 2DFA перейдёт направо, покидая уже прочитанный участок.

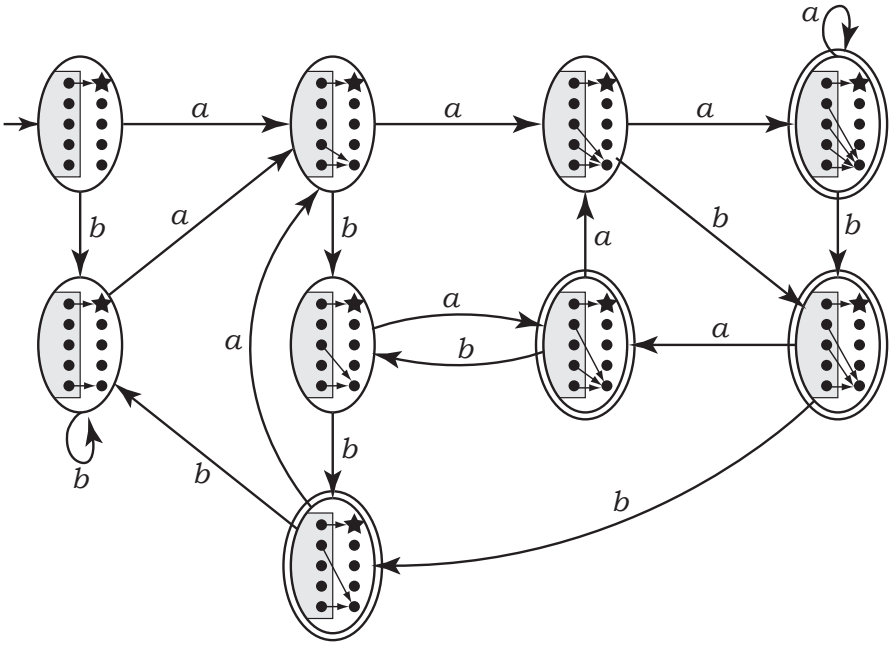

Рис. 10: DFA, построенный по 2DFA для языка $\{a,b\}^*a\{a,b\}^2$.

Преобразование 2DFA к NFA. Если задачу моделирования работы двухстороннего автомата (2DFA) вместо детерминированного автомата (DFA) поручить недетерминированному (NFA), то ему не потребуется запоминать все возможные вычисления — достаточно правильно угадать куски одного-единственного принимающего вычисления и запоминать только их. Оказывается, что чтобы прослеживать ход одного предполагаемого вычисления на строке w, читая её слева направо, не потребуется запоминать произвольную функцию f из Q в Q, и можно ограничиться хранением меньшего объёма сведений о фрагментах предполагаемого вычисления. Немного упрощая, можно сказать, что достаточно хранить *два подмножества* Q — состояния, в которых 2DFA проходит текущий символ слева направо, и состояния, в которых он проходит этот символ справа налево. Точное множество состояний даётся в нижеследующей теореме, приводимой без доказательства.

Теорема 4 (Капуцис [2005]). *Для всякого 2DFA с n состояниями существует NFA с $\binom{2n}{n+1}$ состояниями, распознающий тот же язык.*

А сколько это, $\binom{2n}{n+1}$? Это $\frac{(2n)!}{(n-1)!(n+1)!}$, и, если оценить факториалы по формуле Стирлинга, то множители порядка n^n сократятся, и останутся только экспоненциальные функции, так что $\binom{2n}{n+1} = O(\frac{1}{\sqrt{n}}4^n)$ — и это существенно меньше, чем n^n.

Капуцис [2005] также показал, что оба построения оптимальны по числу состояний — коль скоро размер алфавита неограничен. Однако, нижняя оценка справедлива лишь для алфавита экспоненциального размера. Для алфавитов меньшего размера оба построения можно немного улучшить, с сохранением порядка $2^{\Theta(n \log n)}$.

Также можно определить двухсторонние недетерминированные конечные автоматы (2NFA) с функцией переходов $\delta\colon Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q\times\{-1,+1\}}$. Преобразование в NFA работает для них точно так же, как и для 2DFA; преобразование в DFA немного усложняется.

Упражнение 1. *Доказать, что всякий 2NFA с n состояниями можно преобразовать в DFA с 2^{n^2} состояниями.*

Сложность преобразования 2NFA к 2DFA — едва ли не самая важная открытая задача теории автоматов. Вопрос в том, можно ли перевести всякий 2NFA с n состояниями в 2DFA с числом состояний, полиномиальном от n. Предполагается, что нельзя; однако единственный известный способ преобразования 2NFA к 2DFA — это преобразование 2NFA к DFA, а единственная известная нижняя оценка — всего лишь $\Omega(n^2)$.

Список литературы

[2012] F. Bassino, J. David, C. Nicaud, "Average case analysis of Moore's state minimization algorithm", *Algorithmica*, 63:1–2 (2012), 509–531.

[1956] E. F. Moore, "Gedanken experiments on sequential machines", in: C. E. Shannon, J. McCarthy (Eds.), *Automata Studies*, Princeton University Press, 1956, pp. 129–153.

[1971] J. E. Hopcroft, "An $n \log n$ algorithm for minimizing states in a finite automaton", *Theory of Machines and Computations* (Haifa, Israel, 16–19 August 1971), 189–196.

[2005] C. A. Kapoutsis, "Removing bidirectionality from nondeterministic finite automata", *Mathematical Foundations of Computer Science* (MFCS 2005, Gdansk, Poland, 29 August–2 September 2005), LNCS 3618, 544–555.

[1959] M. O. Rabin, D. Scott, "Finite automata and their decision problems", *IBM Journal of Research and Development*, 3:2 (1959), 114–125.

[1959] J. C. Shepherdson, "The reduction of two-way automata to one-way automata", *IBM Journal of Research and Development*, 3 (1959), 198–200.