2. Детерминированные Конечные Автоматы (DFA)

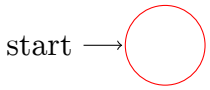
2.1. Детерминированные Конечные Автоматы

***Определение* 2.1.** Детерминированный конечный автомат $A = (\Sigma, Q, q_o, T, \delta)$, где

1. Σ – конечный алфавит;
2. Q – конечное множество состояний;
3. $q_o \in Q$ – начальное состояние;
4. $T \subseteq Q$ – множество терминальных состояний;
5. $\delta : Q \times \Sigma \to Q$ – функция переходов.

Обозначение.

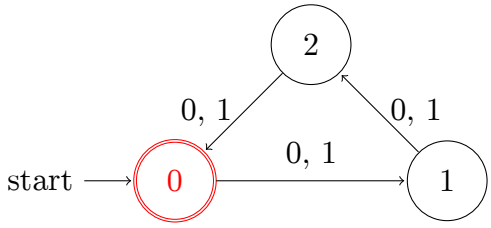
Далее в конспекте так будет обозначаться стартовые состояния:



А вот так терминальные:

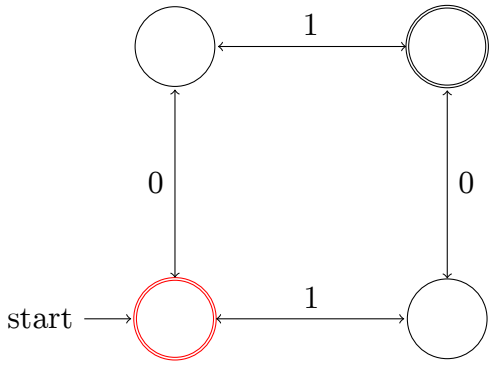
Примеры.

1. Зафиксируем алфавит $\{0, 1\}$. $L = \{w \mid |w| \vdots 3\}$



Переход по любому символу происходит по указанной стрелке. Значение в вершине совпадает с остатком по модулю 3 слова. Терминальное состояние совпадает с начальным.

2. $\Sigma = \{0, 1\}$



Как несложно понять, распознаются только слова четной длины.

***Определение* 2.2.** Обобщенная функция переходов $\hat{\delta} : Q \times \Sigma^* \to Q$, причем $\hat{\delta}(q, \varepsilon) = q$ и $\hat{\delta}(q, xw) = \hat{\delta}(\hat{\delta}(q, x), w)$. И тогда язык принимаемый ДКА $L(A) = \{w \in \Sigma^* : \hat{\delta}(q_o, w) \in T\}$.

Замечание. Проверка принадлежности слова w языку $L(A)$ может быть осуществлена за $\mathcal{O}(|w|)$.

2.2. Минимизация ДКА

Задача минимизации: $A_1 \to A_2$ так, что $L(A_1) = L(A_2)$ и $|Q(A_2)| \to \min$.

***Определение* 2.3.** Состояния p и q – различимые, если $\exists w \in \Sigma^* \;\; \hat{\delta}(p, w) \in T \oplus \hat{\delta}(q, w) \in T$.

Лемма. "p неразличима с q" является отношением эквивалентности.

Доказательство. Рефлексивность, симметричность и транзитивность очевидны. □

Алгоритм (поиска всех пар различимых состояний)**.**

1. $p \in T, q \notin T \Rightarrow (p, q)$ – различима (и симметричный случай);
2. $\hat{\delta}(p, x) = u$ и $\hat{\delta}(q, x) = v$. Тогда (u, v) различима $\Rightarrow (p, q)$ – различима.

Рассмотрим граф, построенный на парах состояний. Для пар из пункта 1 мы знаем, что они различимы. Более того, понятно, что для каждой различимой пары (p, q) существует такая пара $(\hat{\delta}(p, w), \hat{\delta}(q, w))$, где w взято из определения различимой пары (*). Тогда построим граф на таких парах, где будет проведено ребро $(p, q) \to (u, v)$, если $\exists a \in \Sigma : \delta(u, a) = p$ и $\delta(v, a) = q$. Тогда для нахождения всех различимых пар, достаточно запустить DFS или BFS в таком графе от вершин из пункта 1, помечая все достигнутые пары, как различимые. Понятно, что если пара различимая, то по (*), мы найдем все такие пары. Время работы будет $\mathcal{O}(|Q|^2)$, так как в этом графе $|Q|^2$ вершин и $|Q|^2 \cdot |\Sigma|$ ребер.

Замечание. Можно не разворачивать ребра, а просто запустить ленивую динамику.

Алгоритм (Минимизации ДКА)**.**

Рассмотрим класс эквивалентности. Будем строить автомат A_2. Новые состояния соответствуют классам эквивалентности. Начальным состоянием будет класс начального состояния. Каждый класс либо состоит только из терминальных, либо только из нетерминальных, на основании чего определяем терминальность в новом автомате. Несложно заметить, что не существует двух переходов по одной и той же букве из двух состояний из одного класса эквивалентности в два разных класса (тогда эти две вершины были бы различимы). Оставляем только состояния, достижимые из начального. Получили автомат, принимающий те же слова. Автомат меньшего размера получить нельзя, так как все классы эквивалентности нужны.

Замечание. Можно легко обобщить понятие различимости на вершины двух разных автоматов (пара состоит из одной вершины из одного автомата и еще одной из другой). Тогда аналогично найдем все различимые пары вершины в этих двух автоматах. Проверка на эквивалентность этих двух автоматов заключается в проверке различимости стартовых состояний.

2.3. Правые контексты

***Определение* 2.4.** Правым контекстом называется функция $C_L^R : \Sigma^* \to 2^{\Sigma^*}$, $C_L^R(w) = \{u \in \Sigma^* : wu \in L\}$

Аналогично можно определить левые контексты.

Замечание. $w \in L \Leftrightarrow \varepsilon \in C_L^R(w)$

Замечание. $\forall w \in \Sigma^* \;\; C_L^R(w)$ – различные множества правых контекстов. Количество $= |Q_{minA}|$.

***Определение* 2.5.** Тупиковым (дьявольским) состоянием ДКА называется состояние, все переходы из которого ведут в него же самого.

Замечание. Тупиковые состояния на картинке обычно не рисуют.

3. Недетерминированные конечные автоматы с ε-переходами

3.1. Недетерминированные конечные автоматы с ε-переходами

***Определение* 3.1.** Недетерминированный конечный автомат (НКА) – это пятёрка $A = (\Sigma, Q, q_0, T, \delta)$, где

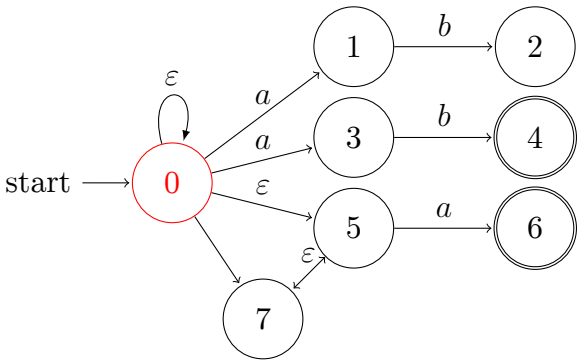
1. Σ – конечный алфавит
2. Q – конечное множество
3. $q_0 \in Q$ – начальное состояние
4. $T \subseteq Q$ – множество терминальных состояний
5. $\delta : Q \times (\{\varepsilon\} \cup \Sigma) \to 2^Q$ – функция переходов (по состоянию и символу получаем множество состояний, в которое мы можем перейти)

НКА принимает строку w и выдаёт "Да", если есть хотя бы один путь по строке w из начальной вершины в терминальную.

***Определение* 3.2.** ε-переход – это переход, который можно спонтанно сделать в любой момент чтения слова.

Число переходов, возможно, экспоненциально больше, чем у детерминированного, но тем не менее конечно.

Пример. Рассмотрим следующий автомат:

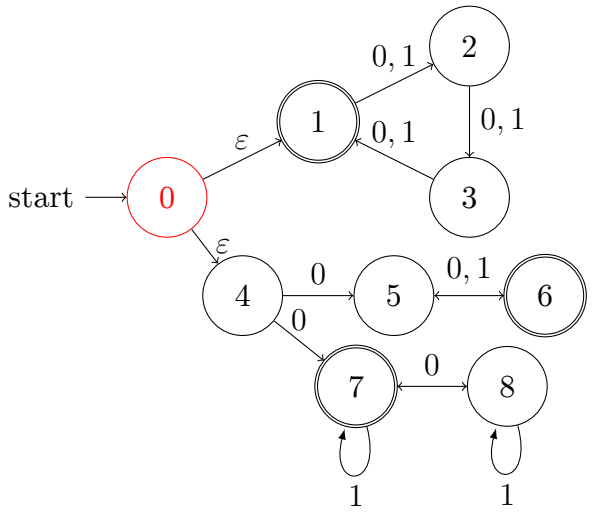


В этот автомат слово "a" можно принять перейдя из 0 в 3, а можно перейти по ε-переходу из 0 в 5, потом походить миллион раз туда-обратно по ε-переходу между 5 и 7, а потом из 5 перейти в 6, и таким образом, это слово распознаётся автоматом.

***Определение* 3.3.** Язык, распознаваемый НКА – это все такие слова, по которым существует хотя бы один путь из стартовой вершины в терминальную.

$L(A) = \{w \in \Sigma^* : \exists$ путь из q_0 в $q \in T$, на ребрах которого написано слово $w\}$

Пример.



Этот автомат разпознаёт следующий язык: $L(A) = \{$слова длины, кратной трём $\cup$ слова чётной длины, начинающиеся с нуля $\cup$ слова с нечётным числом нулей, начинающиеся с нуля$\}$.

То есть мы сначала выбираем, каким образом мы будем принимать наше слово, а потом пытаемся его принять, то есть "хвост" у нас детерминированный.

Надо подумать, как запрограммировать правильный выбор ветки автомата, которая принимает наше слово.

Первое, что приходит в голову – это каждый раз перебирать все возможные варианты, куда пойти, тогда на каждом ходу у нас $\leqslant |Q|$ рёбер, значит нам придётся рассмотреть $O(|Q|^{|w|})$ вариантов (это очень много, не говоря уже о том, что возможны циклы по ε).

Можем вместо того, чтобы перебирать все пути смотреть на множество достижимых по данной строке вершин, и при добавлении новой буквы обновлять это множество, а потом обновлять ещё раз с учётом всех ε-переходами.

***Определение* 3.4.** ε-замыкание состояния q – это множество состояний, достижимых из q только по ε-переходам.

Это множество можно предподсчитать с помощью dfs для каждой вершины. И теперь, когда мы собираемся считать очередной символ, мы добавим вместо вершины сразу её ε-замыкание (это решает проблему с ε-переходами).

Чтобы ещё сильнее оптимизировать, можно использовать вместо булева массива bitset, а также для каждого состояния и каждого символа предподсчитать побитовое "ИЛИ" eps-замыканий всех вершин, в которые из данной вершины есть переход по данному символу.

Таким образом, на предподсчёт у нас уйдёт $O(|Q|^2)$ времени, а именно $|Q|^2$ на построение замыканий и ещё $|\Sigma||Q|^2$ на описанный выше предподсчёт, а про размер алфавита мы сказали, что он константный.

И с помощью этого предподсчёта мы добились того, что у нас переход по очередной букве работает за $O(|Q|^2)$, но такая оценка почти никогда не достигается, поэтому на деле алгоритм работает примерно за $|w|$ (но это не точно).

Заметим, что если программа хранит всегда конечное число информации и читает слово слева направо, то она, по сути, является детерминированным конечным автоматом.

Если же мы посмотрим на все возможные замыкания состояний в ε-НКА, то среди них окажется всего $2^{|Q|}$ различных, то есть константа. То есть программа, которая осуществляет переходы по ε-НКА является детерминированным конечным автоматом.

3.2. Детерминизация ε-НКА

Мы уже догадались, что ε-НКА эквивалентен некоторому ДКА. Давайте формально объясним, как сделать из конкретного ε-НКА A_1 эквивалентный ему ДКА A_2, то есть такой, что $L(A_2) = L(A_1)$.

Алгоритм. Детерминизации ε-НКА

Построим все необходимые части ДКА:

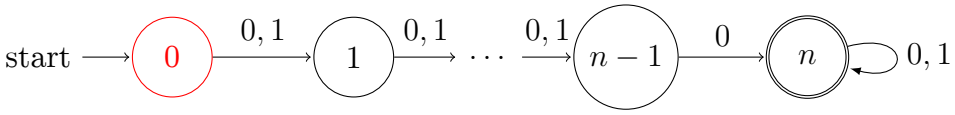
1. $\Sigma_2 = \Sigma_1$, то есть алфавит берём такой же
2. $Q_2 = 2^{Q_1}$, то есть смотрим на множество всех достижимых по данной строке состояний
3. $q_{02} = \varepsilon\text{-closure}(q_{01})$, то есть вместо стартового состояния смотрим на его ε-замыкание.
4. $T_2 = \{S \subset Q : S \cap T \neq \varnothing\}$, то есть все подмножества множества состояний, содержащие хотя бы одно терминальное состояние
5. $\delta_2(S, a) = \bigcup\limits_{p \in S, p \to q} \varepsilon\text{-closure}(q)$, то есть объединение ε-замыканий по всем вершинам, в которые мы можем перейти по этому символу из какого-то состояния текущего множества достижимых

Замечание. Таким образом, множество рапознаваемых ε-НКА и ДКА языков совпадает, потому что по ε-НКА мы научились получать ДКА, а ДКА сам является НКА. Здесь всё вполне логично, так как, по факту, определения отличаются только тем, что у ε-НКА функция перехода возвращает не одно состояние, а множество состояний, и ещё наличие ε-переходов.

Оптимизации: состояние входит всегда со своим замыканием, поэтому получится не полная экспонента, то есть набор множеств достижимых вершин, который может получиться скорее всего будет сильно меньше, чем просто все подмножества.

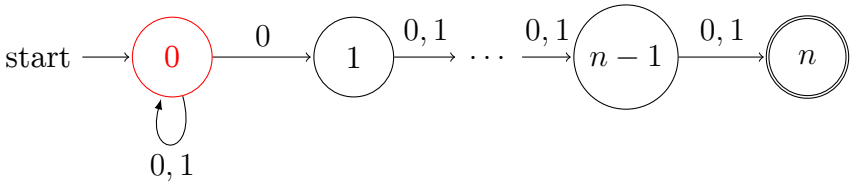
Также dfs для подсчета замыкания можно запускать только из достижимы вершин, и только когда мы в них первый раз заходим. Получается такая ленивая ДП.

Пример. $L_1 = \{w : w_n = 0\}$



И ДКА и НКА для этого языка имеют размер n.

Пример. $L = \{w : w_{|w|-n+1} = 0\} = L_1^R$. ε-НКА:



В ДКА для того же языка $|Q| \geq 2^n$ (можно доказать с помощью числа правых контекстов, но мы не будем). Почувствуйте разницу с $n+1$ состоянием в НКА. Это плохой пример, на котором алгоритм детерминизации экспоненциально взрывной.

Мы умеем делать в НКА каждый переход в худшем случае за $|Q|^2$. Тем временем в ДКА переход – это просто посмотреть число в прямоугольном массиве. Но мы знаем, тем не менее, что если мы захотим построить ДКА, то у нас может получиться слишком много состояний, и мы просто в память не влезет эта табличка.

Идея: попробуем детерминизировать НКА, и если за мало операций получилось детерминизировать, то будем работать с ДКА, а если не повезло, то будем работать с НКА.

3.3. Произведение конечных автоматов

***Определение* 3.5.** Пусть есть два автомата: A и B, тогда произведением автоматов называется автомат $A \times B$, у которого:

1. $\Sigma = \Sigma$, то есть алфавит такой же
2. $Q = Q_A \times Q_B$, множество пар состояний
3. $q_0 = (q_{0A}, q_{0B})$, пара из двух начальных
4. $\delta((p, q), a) = (\delta_A(p, a), \delta_B(q, a))$, то есть переходим по символу в обоих автоматах

В зависимости от того, как мы выберем терминальные состояния, мы можем получить автоматы, распознающие следующие языков:

1. $L_1 \cap L_2$, для этого возьмём $T = T_A \times T_B$
2. $L_1 \cup L_2$, для этого возьмём $T = T_A \times Q_B \cup Q_A \times T_B$

Замечание. Когда мы в ДКА инвертировали множество терминальных состояний, мы получали, автомат, который распознаёт дополнение исходного языка. В ε-НКА такой фокус не пройдёт. Если мы так сделаем, мы получим непонятно что.

4. Регулярные выражения

4.1. Академические регулярные выражения

***Определение* 4.1** (Академические регулярные выражения)**.**

Регулярное выражение R	$L(R)$
$\varnothing$	$\varnothing$
ε	$\{\varepsilon\}$
$a \quad (\forall a \in \Sigma)$	$\{a\}$
$R_1 \mid R_2$	$L(R_1) \cup L(R_2)$
$R_1 R_2$	$\{w \in \Sigma^* \mid w = xy, x \in L(R_1), y \in L(R_2)\}$
R^*	$\{w \in \Sigma^* \mid w = x_1 x_2 \dots x_n, n \in \mathbb{Z}_{0+}, \forall x_i \in L(R)\}$
(R)	$L(R)$

Последние четыре операции перечислены в порядке возрастания приоритета. Предпоследняя операция называется "замыкание Клини". Получили рекурсивное определение, т.е. выражение, полученное в результате применения нескольких правил выше и будет называться академическим регулярным выражением.

Примеры.

1. $(0|1)^*$ – язык, состоящий из всех слов над алфавитом $\{0, 1\}$
2. $(0|1)^*001^*$ – язык из слов, последняя группа нулей которых имеет длину хотя бы два
3. $(00|1)^*$ – язык, состоящий их слов, все группы нулей которых имеют четную длину

Замечание.

1. $RR* = R_+$
2. $RR(\varepsilon|R)(\varepsilon|R) = R\{2-4\}$
3. Пример из промышленных (расширенных) регулярных выражений: $(00+)\backslash 1+$ – слова составной длины из нулей

Теорема 4.1 (Клини)**.** Множество языков, задаваемых академическими регулярными выражениями совпадает с множество языков, задаваемыми конечными автоматами.

Доказательство.

1. Докажем, что регулярные языки $\subseteq$ автоматные.
   $\forall R \to \varepsilon-$НКА $A: \; L(A) = L(R)$.
   Далее, аналог индукции:
   Итак, будем строить автомат, в котором будут выполнены следующие условия:

   (a) ровно одно терминальное состояние

   (b) в начальное состоянии нет переходов

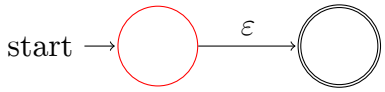
(c) из терминального состояния нет переходов

(d) начальное и терминальное состояние не совпадают

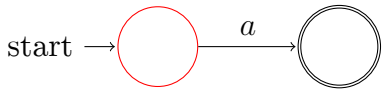
Итак, построение:

1) Автомат, задающий пустой язык:

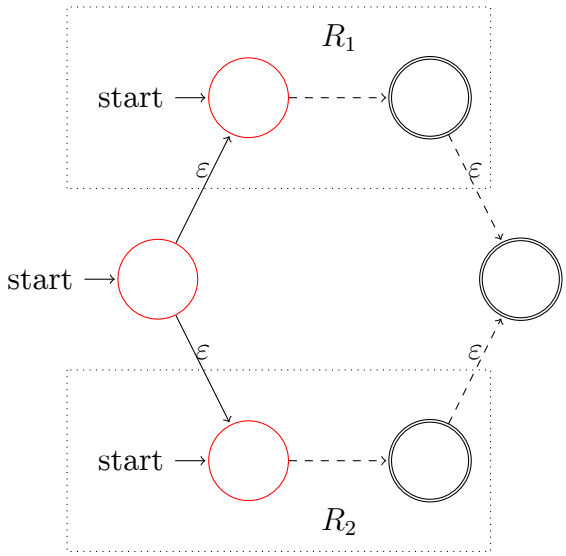
2) Автомат, задающий пустое слово:



3) Автомат, задающий одну букву a:

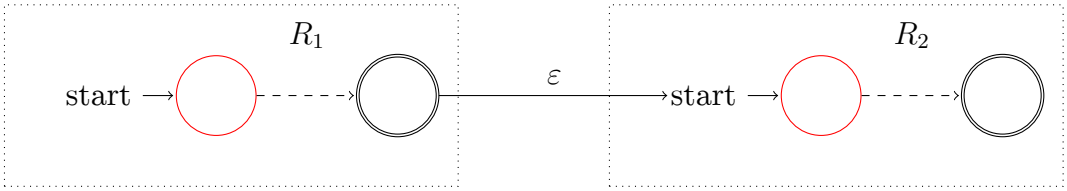


4) Автомат, задающий $R_1|R_2$ (для которых уже есть свои автоматы)



Т.е. берем старые автоматы для R_1 и R_2, после чего добавляем новое стартовое состояние, из которого добавляем переходы по ε в стартовые состояния старых автоматов, добавляем новое терминальное состояние, в которое добавляем переходы по ε из старых терминальных состояний (по построению таких ровно два). Заметим, что теперь мы должны сказать, что старые стартовые и терминальные состояния больше не являются таковыми, так как их должно быть по одному (в точности новые). Тогда все условия будут выполнены.

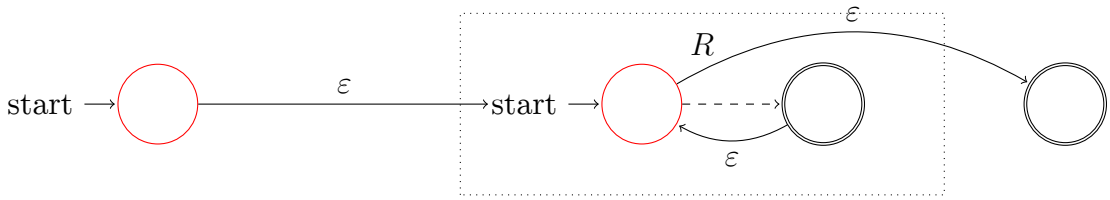
5) Автомат, принимающий $R_1 R_2$



Т.е. берем старые автоматы для R_1 и R_2, добавляем ε-переход из терминального состояния R_1 в стартовое R_2. После чего говорим, что единственное стартовое состояние – стартовое состояние R_1, а единственное терминальное – терминальное состояние R_2.

Замечание. На самом деле, можно просто схлопнуть терминальное состояние R_1 и стартовое R_2.

6) Автомат, принимающий R^*



Т.е. добавляем новое терминальное состояние, ε-переход из терминального состояния автомата, распознающего R в стартовое состояние, а так же из стартового состояния ε-переход в новое терминальное состояние. Для того, чтобы выполнить условие (b), добавим новое стартовое состояние и ε-переход из него в стартовое состояние автомата, распознающего R. Т.е. терминальное и стартовое состояние у нас новые.

Таким образом, мы разберем регулярное выражение (формально, построим дерево разбора, применим индукцию), после чего получим автомат с $\mathcal{O}(|R|)$ состояниями. Причем данный алгоритм работает также за линейное время.

2. Автоматные языки $\subseteq$ регулярные.
ДКА $A \to$ АРВ $R : L(R) = L(A)$
$R_{i,j,k}$ – АРВ, задающее язык всех слов, переводящих автомат A из состояния i в состояние j, использующая промежуточные состояния только меньшие k (в данном автомате состояния $\{0, 1 \dots, n-1\}$)

Слой $k = 0$

1) $i = j$ $R_{i,i,0} = \varepsilon | a_1 | a_2 | \dots | a_k$, где $a_1, a_2, \dots, a_m$ – буквы, по которым есть петля $i \to i$.
2) $i \neq j$ $R_{i,j,0} = a_1 | a_2 | \dots | a_k$, где $a_1, a_2, \dots, a_m$ – буквы, по которым есть переход $i \to j$ и $\varnothing$, если таких ребер нет.

Слой $k > 0$

$R_{i,j,k} = R_{i,j,k-1} | R_{i,k-1,k-1} R^*_{k-1,k-1,k-1} R_{k-1,j,k-1}$

Ответ

$R = R_{q_0,t_1,n}|R_{q_0,t_2,n}|\ldots|R_{q_0,t_{|T|},n}$ или $\varnothing$, если терминальных состояний нет.
Тогда $L(R) = L(A)$.
Оценим время работы: $\mathcal{O}^*(4^n)$. Размер ответа такой же.

□

5. Ещё про детерминированные конечные автоматы

5.1. Лемма о разрастании

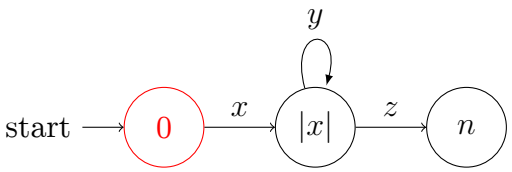
Лемма. Для любого регулярного языка L существует $n \in \mathbb{N}$, что любое слова $w \in L$ длины $\geq n$ представляется в виде xyz, где $x, y, z \in \Sigma^*$ и выполняются следующие условия:

1. $y \neq \varepsilon$
2. $|xy| \leqslant n$
3. $\forall i \in \mathbb{Z}_+ \cup \{0\}$ верно, что $xy^iz \in L$

Доказательство. Так как язык L регулярный, то для него есть ДКА A, такой, что $L(A) = L$. Тогда в качестве n нам подойдёт $|Q|$.

Если мы возьмём строку длины больше, чем $|Q|$, и будем последовательно переходить по её символам в автомате, то к тому моменту, как мы пройдём $|Q|$ символов, мы в каком-то состоянии побываем дважды.

Заметим, что если бы мы прошли по циклу ещё сколько угодно раз или же вовсе не прошли бы, слово не перестало бы лежать в языке. Значит, мы можем взять самый первый цикл, в который мы попали в качестве y, а его предпериод в качестве x, тогда $|xy| < |Q|$, а так как чтобы зациклиться мы должны были пройти по каким-то символам, то $y \neq \varepsilon$.



□

Замечание. Любой конечный язык регулярен, так как можно взять n такой, что любое слово имеет длину $< n$, например, $n := \max\limits_{w \in L} |w| + 1$

Пример. Язык Дика, aka $L =$ ПСП. Докажем, что он нерегулярный

Доказательство. Предположим, что он регулярный, тогда по лемме о накачке существует n с вышеописанными свойствами. Возьмём последовательность из n открывающих, а затем n закрывающих скобок. Для неё существуют соответствующие x, y, z из Леммы. Но так как $|xy| \leq n$, то y состоит только из открывающих скобок, причём по условию Леммы y не пустая. А значит при $i = 2$ в строке xy^iz получится больше открывающих скобок, чем закрывающих, то есть это будет не ПСП. Получили противоречие. □

Доказательство. (С помощью правых контекстов)

Если язык регулярный, то количество его различных правых контекстов равно количеству состояний в минимальном автомате, то есть конечно. Посмотрим, что можно сказать про правые контексты в языке Дика.

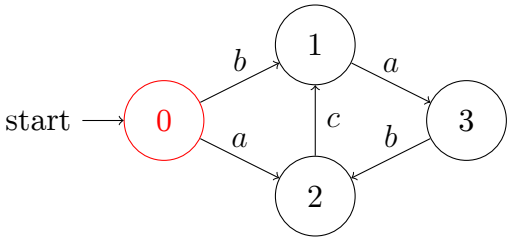
w_n – строка из n открывающих скобок. Правый контекст для w_n содержит строку из n закрывающих скобок, но при этом ни для какого $m \neq n$ правый контекст w_m эту строку не содержит по определению ПСП.

Таким образом, для всех w_n правые контексты различаются, значит их бесконечное число, а значит язык Дика нерегулярен. □

5.2. Динамическое программирование по ДКА

Пример. Дан регулярный язык, найти кратчайшее слово, принадлежащее ему.

Запускаем bfs, выходим, когда пришли в терминальное состояние + восстановление ответа.



Кратчайшее слово в языке, распознаваемом этим автоматом – ba.

Пример. Количество слов длины l в L.

$a_{q,i}$ – количество слов длины i, переводящих A из q_0 в q.

Чтобы пересчитать эту величину, нужно просуммировать значения ДП из предыдущего по длине слоя для всех состояний, из которых есть ребро в q.

У нас есть правило, по которому линейной комбинацией из столбца выводится следующий, поэтому можно посчитать n-ый столбец даже для очень больших n с помощью возведения матрицы в степень.

Ответ – это сумма элементов столбца, соответствующих терминальным вершинам.