Порождающие грамматики

$G = (VT, VN, P, S)$, где

- VT – множество терминальных символов;
- VN – множество нетерминальных символов;
- P – непустое конечное множество правил грамматики;
- S – начальный(стартовый) символ грамматики, $S \in VN$.

Язык L(G) – в общем случае подмножество VT^*.

Формальный язык

$$L(G) = \{\alpha \mid \alpha \in VT^* \ \& \ S \Rightarrow* \alpha\}$$

Грамматика определяет структуру предложения.

Процесс порождения предложения называется **вывод**

$S \rightarrow aQb \mid accb$

$Q \rightarrow cSc$

$S \Rightarrow aQb \Rightarrow acScb \Rightarrow$ acaccbcb ! accccb ?

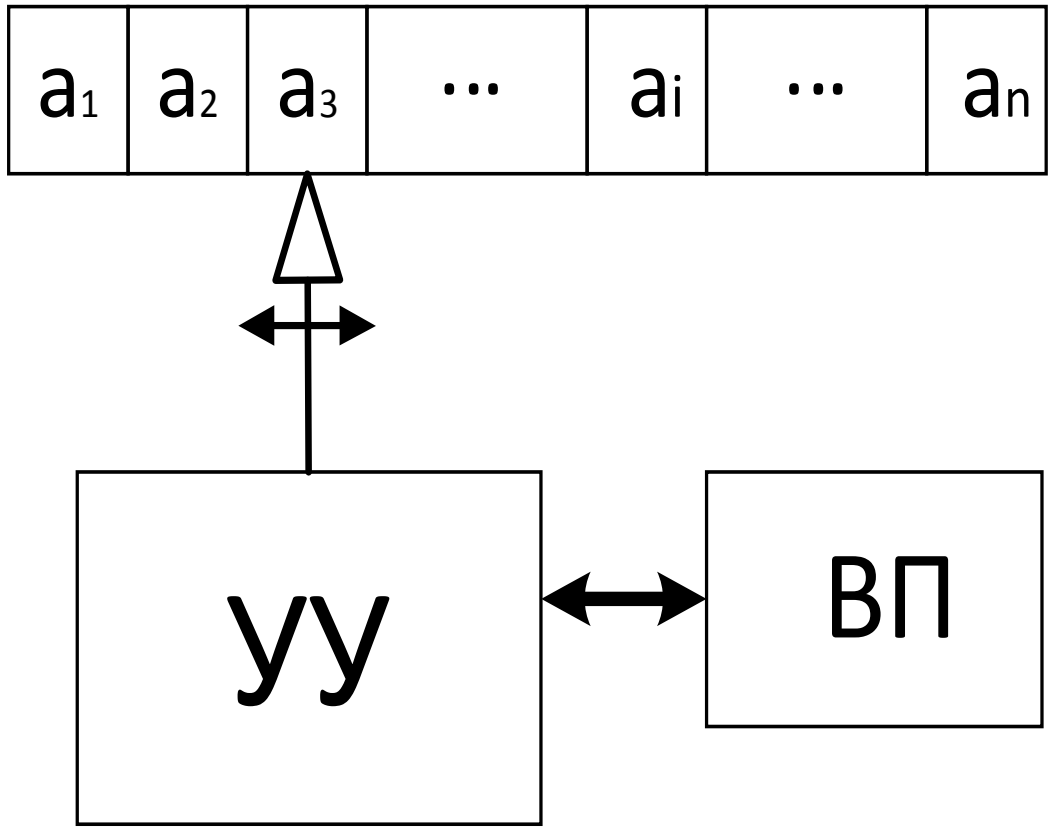
Цепочка принадлежит языку, порождаемому грамматикой, только в том случае, если существует ее вывод из начального символа этой грамматики.

Решение обратной задачи

- Принадлежит ли цепочка символов языку, порождаемому грамматикой?
- Распознаватель – алгоритм или программа, позволяющие определить принадлежность цепочки символов некоторому языку.

- Очевидно существует очень тесная связь между порождающей грамматикой и распознавателем.

Распознаватель



Конфигурация распознавателя

- содержимое входной цепочки символов и положение считывающей головки в ней;
- состояние УУ;
- содержимое внешней памяти.

Распознаватель (продолжение)

Типы конфигураций

- Начальная конфигурация
- Одна или больше конечных конфигураций
- Промежуточные конфигурации

Детерминизм

- для каждой допустимой конфигурации распознавателя, которая возникла на некотором шаге его работы, существует единственно возможная конфигурация, в которую распознаватель перейдет на следующем шаге работы

Классификация распознавателей

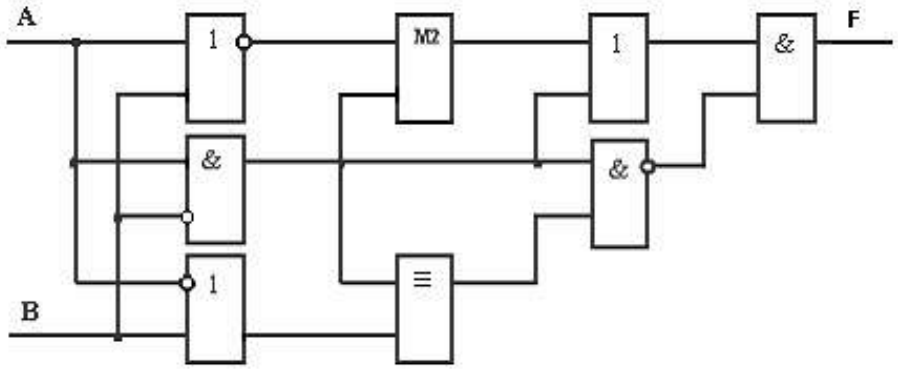
- По **видам считывающего устройства** распознаватели могут быть *двусторонние* и *односторонние*.
- По **видам устройства управления** распознаватели бывают детерминированные и недетерминированные.
- По **видам внешней памяти** распознаватели бывают следующих типов:
- распознаватели *без внешней памяти*;
- распознаватели *с ограниченной внешней памятью*;
- распознаватели *с неограниченной внешней памятью*.

Классификация распознавателей

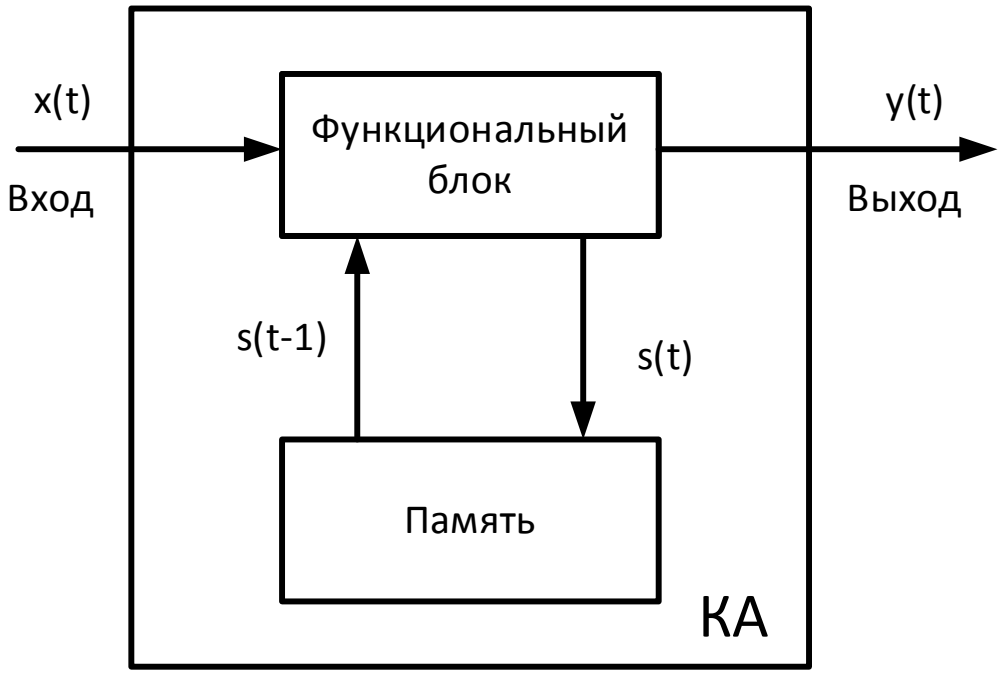
языки типа 0	Машина Тьюринга
языки типа 1	Линейно-ограниченные автоматы МТ с ограниченной лентой
языки типа 2	Автоматы с магазинной памятью
языки типа 3	Конечные автоматы

Комбинационная схема vs Автомат

Комбинационная схема



Автомат



Конечные автоматы.

Абстрактный дискретный исполнитель, преобразующий последовательность входных символов в последовательность выходных символов, так, что значение символа на выходе исполнителя зависит не только от значения входного символа, но и от предыдущей последовательности входных символов (предыстории) принято называть ***автоматом***.

Подробности:

Карпов Ю. Г., Теория автоматов: Учебник для вузов. - 1-е издание. – СПб: Издат. дом ПИТЕР, 2003 год. – 208с.

Конечный автомат.

$A = (X, Y, S, s_0, \delta, \lambda)$, где:

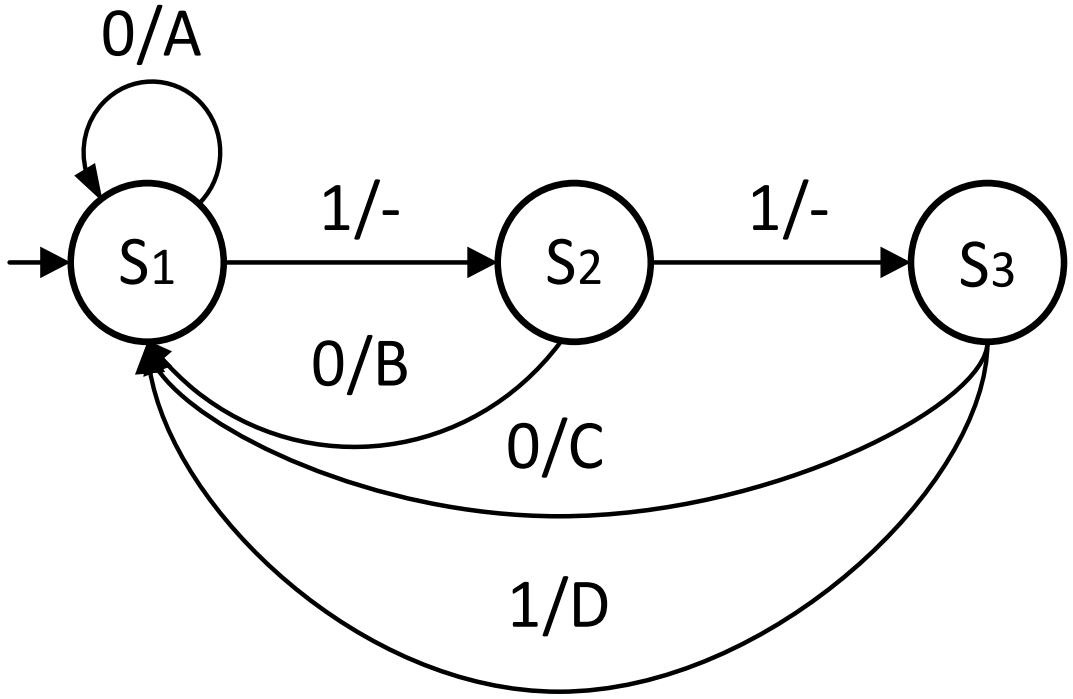
- X – конечное непустое множество входных сигналов (входной алфавит);
- Y – конечное непустое множество выходных сигналов (выходной алфавит);
- S – конечное непустое множество состояний;
- s_0 – начальное состояние автомата, $s_0 \in S$;
- $\delta: S \times X \rightarrow S$ – функция переходов;
- $\lambda: S \times X \rightarrow Y$ – функция выходов.

Конечный автомат-преобразователь

- Пусть символам A, B, C, D соответствуют неравномерные двоичные коды 0, 10, 110, 111.
- X = {0, 1}
- Y = {A, B, C, D, -}

δ	0	1
S1	S1	S2
S2	S1	S3
S3	S1	S1

λ	0	1
S1	A	-
S2	B	-
S3	C	D



Конечный автомат-распознаватель.

$A = (X, S, s_0, \delta, F)$, где:

- X – конечное непустое множество входных сигналов (входной алфавит);
- S – конечное непустое множество состояний;
- s_0 – начальное состояние автомата, $s_0 \in S$;
- $\delta: S \times X \rightarrow S$ – функция переходов;
- F – непустое множество заключительных состояний.

Конечный автомат-распознаватель. 2

- Под конфигурацией ДКА понимается двойка вида (s, ω)
- Начальная конфигурация ДКА (s_0, ω)
- пусть $\delta(s_i, t) = s_j$, причем $s_i, s_j \in S$, $t \in X$, тогда для всех цепочек $\omega \in X^*$ справедливо следующее отношение:

(s_i, tω,) $\vdash$ (s_j, ω)

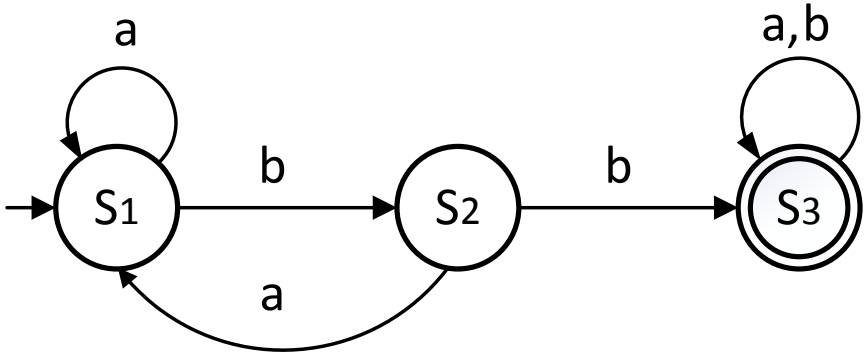
Конечной конфигурацией ДКА является двойка (s, ε) где $s \in F$

ДКА допускает цепочку входящих символов $\alpha(s_0, \alpha) \vdash * (s, \varepsilon)$, для некоторых $s \in F$

Автомат-распознаватель (продолжение)

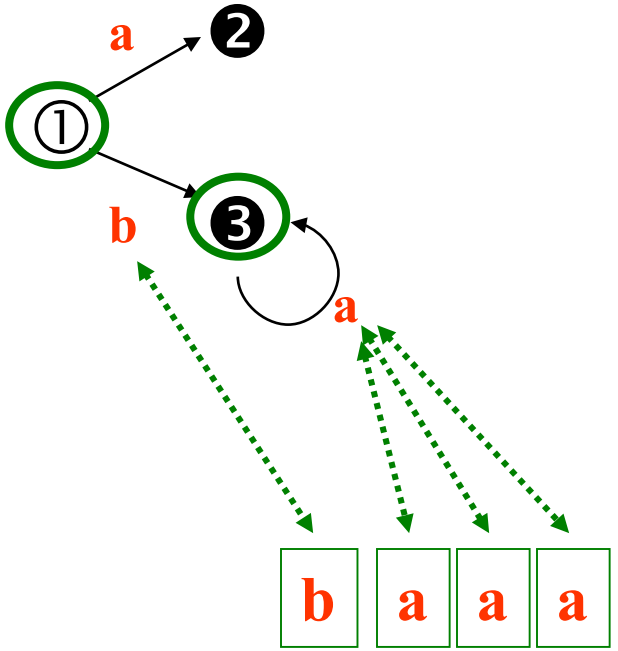
- язык L1 = {α | α = {a, b}$^+$ и α включает подцепочку bb}
- ДКА

Начальная вершина	Принимающая (допускающая) вершина



ДКА пример

Для ДКА из данного состояния s_i по a_i:
Существует единственное s_{i+1}
⇒ допуск/не допуск за линейное время



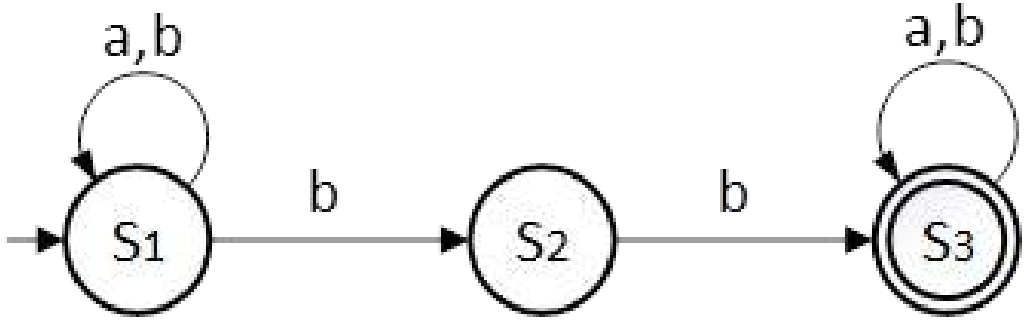
Допущена!

“a”	**допуск**
“aa”	отвергнута
“b”	**допуск**
“bb”	отвергнута
“baaa”	**допуск**
“”	отвергнута

Lge(**A**) =
{**a,b,ba,baa,baaa,baaaa,…**}

Недетерминированный конечный автомат

- $A = (X, S, s_0, \delta, F)$, где:
- X – входной алфавит;
- S – конечное непустое множество состояний;
- s_0 – начальное состояние автомата, $s_0 \in S$;
- $\delta: S \times X \to 2^S$ – функция переходов; (было $\delta: S \times X \to S$)
- F – множество принимающих состояний, $F \subseteq S$.

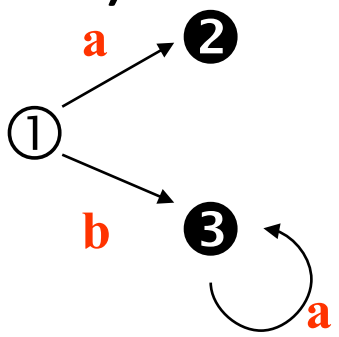


ДКА vs. НКА

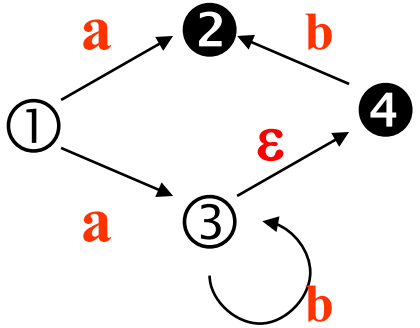
- **Детерминированный** конечный автомат (ДКА):

  Для каждого состояния s и символа **a**: существует единственная исходящая дуга, помеченная **a** **(проще моделировать)**

- **Недетерминированный** конечный автомат (НКА) для некоторых состояний может быть несколько исходящих дуг помеченным одним и тем же символом, кроме того допускаются **ε**-дуги (**проще синтезировать**)

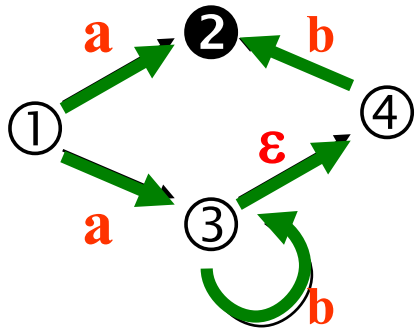


ДКА



НКА

НКА пример



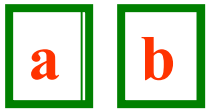
“a” **принято**

“aa” отклонено

“ab” **принято**

L(**A**) = {**a,ab,abb,abbb,…**}

Принята!



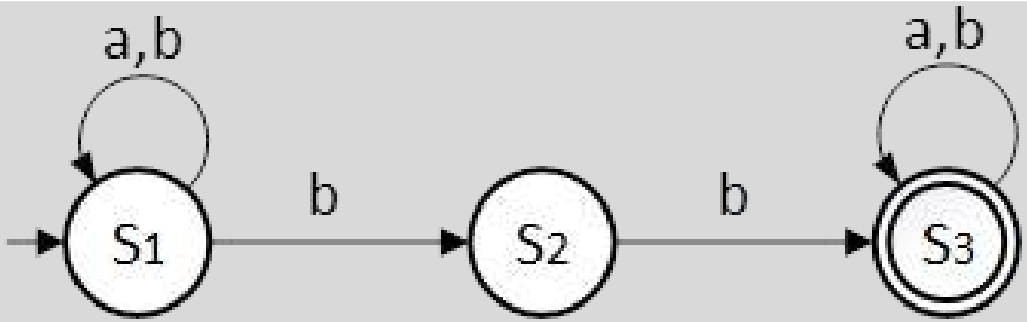
Регулярное множество

- Ø (пустое множество) – регулярное множество для T;
- {ε} – регулярное множество для T (ε– пустая цепочка);
- {a} – регулярное множество T, причем a∈T;
- если P и Q – регулярные множества, то регулярными являются и множества
  - P∪Q (объединение),
  - PQ (конкатенация),
  - P^* (итерация);
- ничто другое не является регулярным множеством для T.

Регулярное выражение

- $\emptyset$ – регулярное выражение, обозначающее множество $\emptyset$;
- ε – регулярное выражение, обозначающее множество $\{\varepsilon\}$;
- a – регулярное выражение, обозначающее множество {a};
- если p и q – регулярные выражения, соответствующие регулярным множествам P и Q соответственно, то
  - $p|q$ – регулярное выражение, обозначающее регулярное множество $P \cup Q$,
  - pq – регулярное выражение, обозначающее регулярное множество $PQ = \{xy \mid x \in P, y \in Q\}$,
  - p^* – регулярное выражение, обозначающее регулярное множество P^*.

Регулярные выражения

	Регулярное выражение	**Регулярное множество (язык)**
1	abc	{abc}
2	a|b|c	{a, b ,c}
3	a*	{ε, a, aa, aaa, aaaa, aaaaa, …}
4	$p^+ = pp^*$ p? = ε|p	
5	(a|b)*bb(a|b)*	

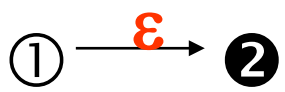
Теорема Клини (*Stephen Cole Kleene*)

- ДКА с входным алфавитом X допускает язык L тогда и только тогда, когда L является регулярным множеством для алфавита X.
- регулярные грамматики, конечные автоматы и регулярные выражения являются равномощными средствами определения регулярных языков, и значит, что существуют любые их взаимные преобразования на уровне алгоритмов
- Хопкрофт, Д. Введение в теорию автоматов, языков и вычислений. –М.: Изд.дом Вильямс, 2008. – Стр. 109

Регулярное выражение → НКА

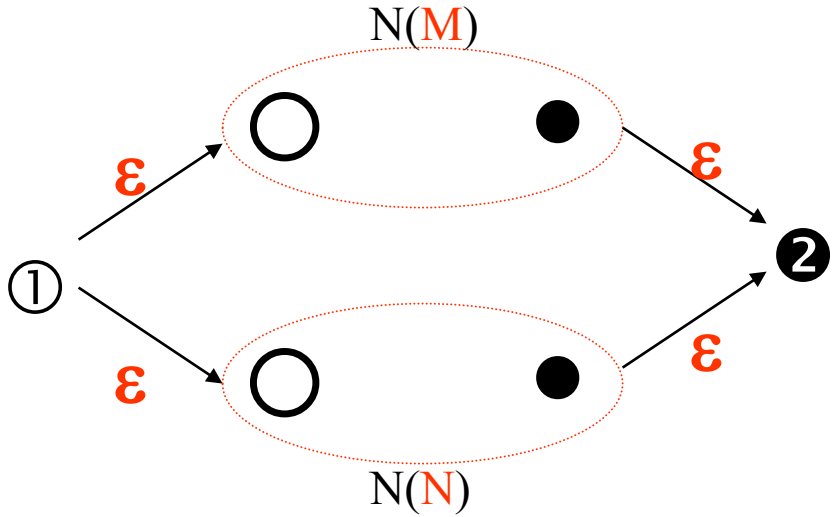
- Определено в [AhoLamSethiUllman] Алгоритм Мак-Нотона – Ямады – Томпсона (McNaughton – Yamada – Thompson)
  - Немного отличается от[Appel] Andrew W. Appel Modern Compiler Implementation in C
- **a**, где a $\in \Sigma$
  - N(**a**) =

- **ε**
  - N(**ε**) =

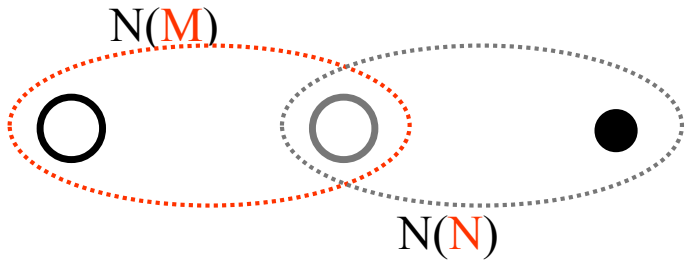


Регулярное выражение → НКА

- **M|N**
  - N(**M|N**) =

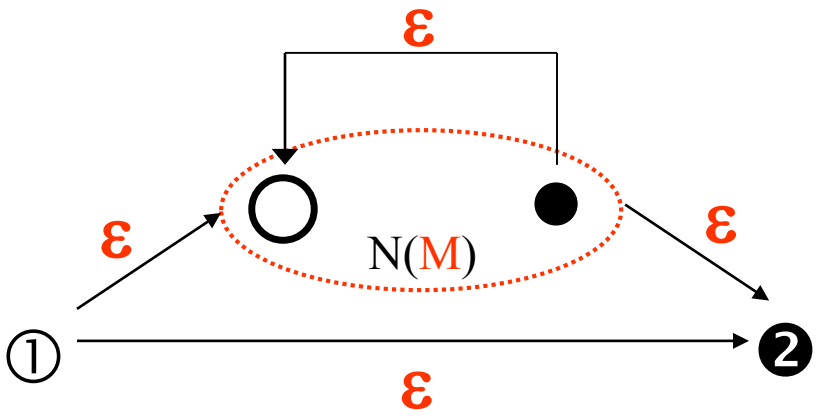


- **M.N**
  - N(**M.N**) =



Регулярное выражение → НКА

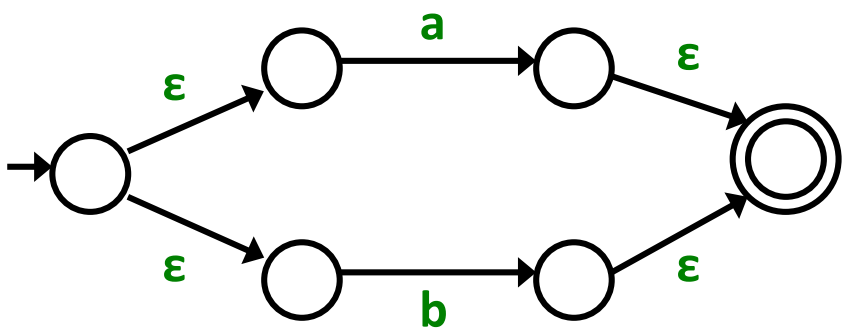
- **M***
  - N(**M***) =



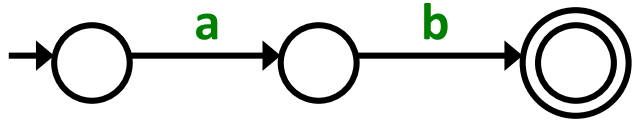
- Можем начинать конструировать НКА по регулярному выражению.
- Есть проблема с реализацией.

Пример: РВ → НКА

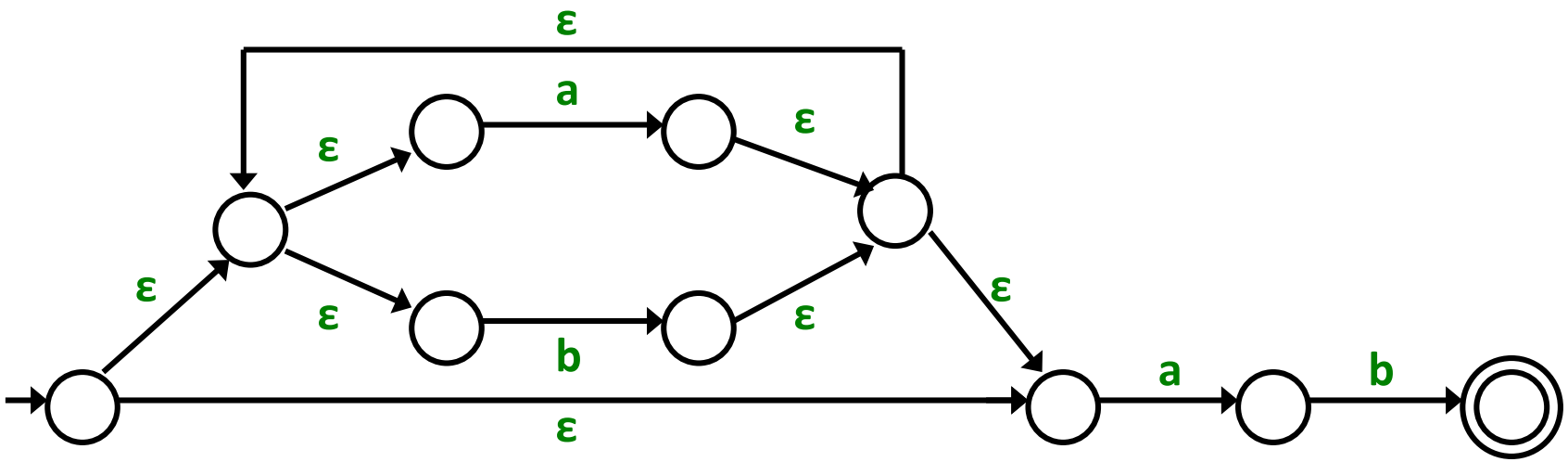
- (a | b)*ab
- a | b →



- ab →



- Объединяем:



Построение ДКА по НКА

Пусть N – НКА, причем $N = \langle X_N, S_N, s_0, \delta_N, F_N \rangle$.

Введем следующие обозначения: s – состояние автомата N, $s \in S_N$; T – множество состояний автомата N, $T \subset S_N$.

ε-closure(s)	Множество состояний НКА, достижимых из s по ε-переходам (эпсилон-замыкание).
ε-closure(T)	Множество состояний НКА, достижимых из $s \in T$ по ε-переходам. ε-closure(T) = $\bigcup_{s \in T}$ ε-closure(s)
move(T, a)	Множество состояний НКА в которые существуют переходы из состояний $s_i \in T$ по входному символу a.

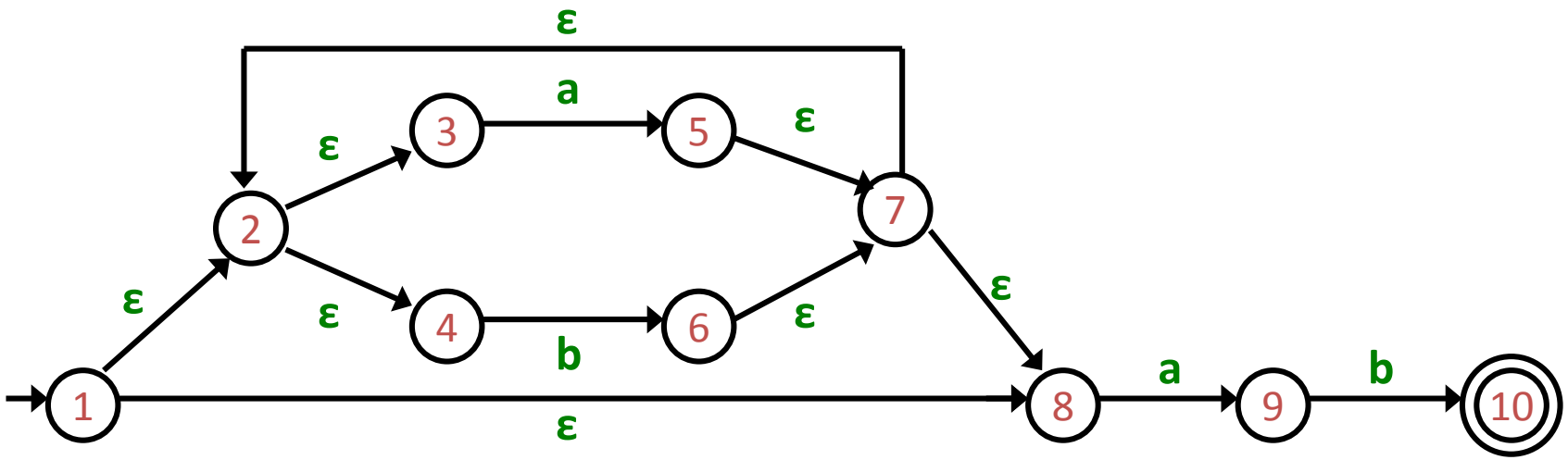
Построение ДКА по НКА

- ε-замыкание рекурсивно определяется следующим образом:
- $s_i \in$ ε-closure(s_i);
- Шаг рекурсии. Пусть s_j является элементом ε-closure(s_i). Если $s_k \in \delta(s_j, \varepsilon)$, то $s_k \in$ ε-closure(s_i).
- Замыкание: $s_j \in$ ε-closure(s_i) только если оно достижимо из состояния s_i конечного числа применения шага рекурсии.

Неформальное определение алгоритма

- Получаем начальное состояние s ДКА как ε-замыкание начального состояния НКА (S_{0DFA} = ε-closure(S_{0NFA}));
- Создаем пустое множество M. Для каждого символа t, по которому есть переход из s, строим множество Move(s, t) и добавляем к M.
- Строим множество K = ε-closure(M)
- Если K отсутствует в таблице переходов ДКА, помещаем это множество в таблицу в качестве нового состояния.
- Повторяем шаги 2-4 до тех пор, пока появляются новые состояния.

Пример: НКА → ДКА



Начальное состояние = ε-closure(1)

	State	a	b
①	1,2,3,4,8	2,3,4,5,7,8,9	2,3,4,6,7,8
②	2,3,4,5,7,8,9	2,3,4,5,7,8,9	2,3,4,6,7,8,10
③	2,3,4,6,7,8	2,3,4,5,7,8,9	2,3,4,6,7,8
❹	2,3,4,6,7,8,10	2,3,4,5,7,8,9	2,3,4,6,7,8