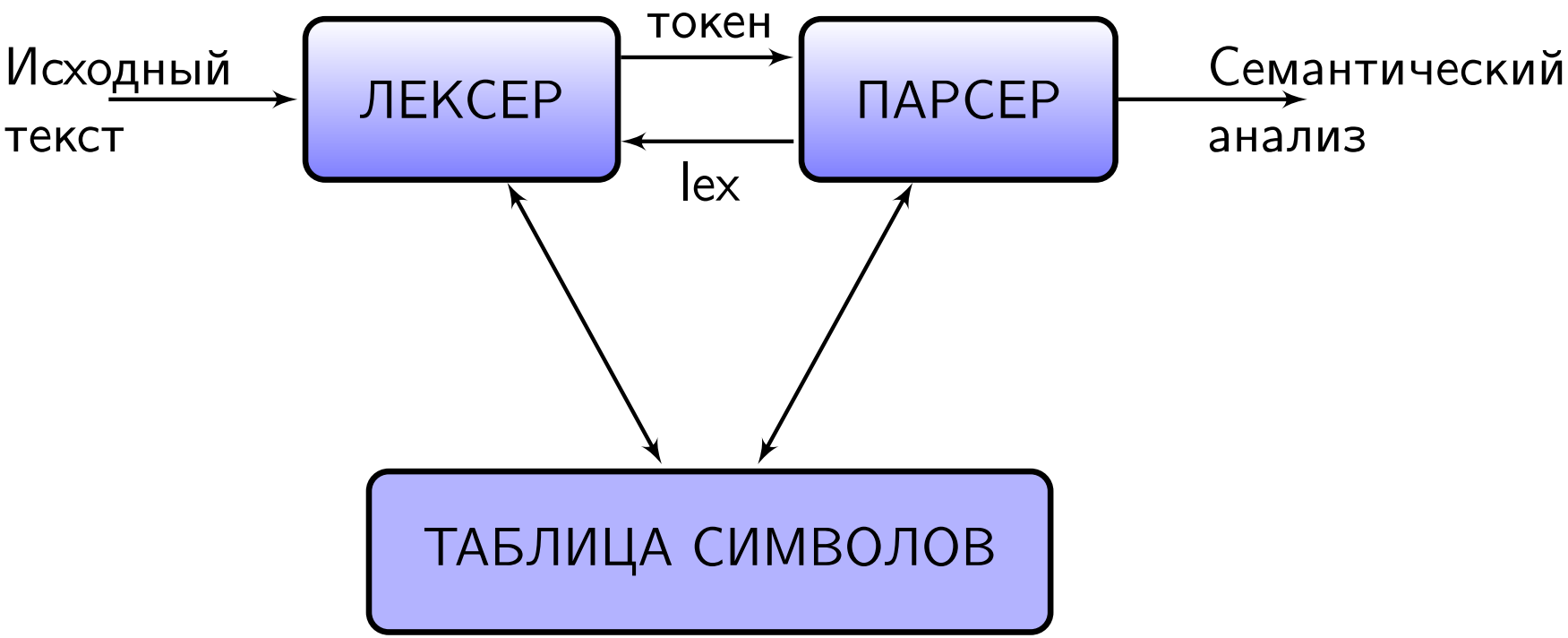
Теория и практика компиляции программ: лексический анализ

Юхин Кирилл

iLab, Intel Corp

МФТИ, 26 сентября 2018 года

Положение лексического анализа



Токен, лексама и шаблон

Токен – пара: имя и, опционально, прикреплённое значение. Имя – это элемент некоторого конечного множества.

Лексема – последовательность знаков, соответствующая определённому токену.

Шаблон – описание всех лексем, соответствующих токенам с одним и тем же именем.

Примеры

Имя токена	Шаблон	Лексемы
BOOLEAN	“#t” или “#f”	#t, #f
LPAREN	“(” или “[”	(, [
SYMBOL	набор букв и цифр, содержащий букву	alala, 3k34dk, if, let
NUMBER	непустой набор цифр и точки	32, 0.0, .121, 3.14
STRING	всё, кроме " окружённое "	"мечты" "error"
SPACE	непустая последовательность пробелов	

Атрибуты токенов

(format #t “~a” 18.3)

- (LPAREN)
- (SYMBOL, индекс “format” в таблице символов)
- (SPACE)
- (BOOLEAN, true)
- (SPACE)
- (STRING, индекс “~a” в таблице строк)
- (SPACE)
- (NUMBER, 18.3)
- (RPAREN)

Лексические ошибки

- Некоторые ошибки лексер не способен обнаружить, например:
  - (let ((a 3))) a) $\rightarrow$ лишняя скобка
  - (/ 1 “2”) $\rightarrow$ неправильный тип
- Однако некоторые, может:
  - "abcd $\langle$ EOF $\rangle$ $\rightarrow$ незаконченная строка
  - 13a $\rightarrow$ недопустимая конструкция в Си
- Такие ошибки как правило сигнализируются когда ни один шаблон не может отыскать лексему в тексте

Восстановление после ошибок

Паника Игнорирование всех последующих знаков. До конца файла, либо до некоторого знака (например пробела)

Догадка

- Удаление одного знака из входного потока,
- Вставка надостающего знака во входной поток
- Замена одно знака другим
- Перестановка двух соседних знаков

и повторная попытка найти лексему.

Общий механизм анализа

- Лексер считывает и запоминает по одному знаку.
- Когда информации становится достаточно, он передаёт парсеру очередной токен.
- Иногда лексеру требуется знать последующие символы, чтобы определить текущий токен. Например:
  - В Fortran: DO 5 I = 1.25 $\leftrightarrow$ DO 5 I = 1,25
  - В Си: -, <, =

  Для этого применяется двухбуферная схема.

Определения

Алфавит – конечное множество знаков. Обозначим Σ.

Язык – набор строк знаков из алфавита Σ.

Грамматика – набор правил, выделяющий из всех последовательностей алфавитных знаков, строки принадлежащие языку.

Функция значания L(e) = M – язык M соответствующий грамматике e.

Грамматика задаёт только один язык. Но язык может быть определён несколькими грамматиками. L(e) - сюръекция.

Регулярные грамматики

- Задают языки, включающие:
  - Пустую строку ε: {“”} или $\varnothing$
  - Одиночные символы из алфавита Σ : {“a”}, {“b”}
  - Объединение: $A + B = \{a|a \in A\} \cup \{b|b \in B\}$
  - Конкатенация: $AB = \{ab|a \in A \wedge b \in B\}$
  - Повторение: $A^* = \bigcup_{i \geq 0} A^i, \{A^0 = \varepsilon, A^i = A \ldots A\ i\text{раз}\}$
- Контекстно-свободные
  - C++ шаблон: A<B<C>> a;
  - C++ сдвиг: C>> a;
- Задаются регулярными выражениями
- Задаются конечными автоматами

Регулярные выражения

- База:
  - Пустая строка ε: $L(\varepsilon) = \{\varnothing\}$
  - Один символ $a \in \Sigma$: $L(\#\backslash a) = \{"a"\}$
- Композиция:
  - Объединение: $L(A + B) = L(A) \cup L(B)$
  - Конкатенация: $L(AB) = \{ab | a \in L(a) \wedge b \in B\}$
  - Повторение: $L(A^*) = \bigcup\limits_{i \geq 0} A^i$

- Расширения:
  - $A^+ = AA^*$
  - $A? = A + \varepsilon$
  - $A^k = \underbrace{AA \cdots A}_{k \text{ раз}}$

POSIX нотация

- $\text{“”} \equiv \varepsilon$
- $\text{“}abc\text{”} \equiv \{\text{“}a\text{”}\}|\{\text{“}b\text{”}\}|\{\text{“}c\text{”}\}$
- $(A) \equiv A$
- $A|B \equiv A + B$
- AB
- $. \equiv \Sigma \backslash \{\text{“перевод строки”}\}$
- $[abc] \equiv \text{“}a\text{“}|\text{“}b\text{“}|\text{“}c\text{“}$
- $[j - p] \equiv \{$все буквы коды которых лежат между j и p включительно $\}$
- $[\hat{}...] \equiv \Sigma$ кроме $[...]$
- $A\{n\} \equiv A^n$
- $A\{n, m\} \equiv A^n + A^{n+1} + \cdots + A^m$
- $A\{, m\} \equiv A\{0, m\}; A\{n, \} \equiv A^n A^*$

Примеры

- Цифра: '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9' $\equiv [0-9]$
- Целое неотрицательное число(Z_0^+):
- Целое число(Z):
- Десятичное дробное число:
- Идентификатор в Си:

Примеры

- Цифра: '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9' $\equiv [0-9]$
- Целое неотрицательное число(Z_0^+): $[0-9]^+$
- Целое число(Z):
- Десятичное дробное число:
- Идентификатор в Си:

Примеры

- Цифра: '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9' $\equiv [0-9]$
- Целое неотрицательное число(Z_0^+): $[0-9]^+$
- Целое число(Z): $'-'?[0-9]^+$
- Десятичное дробное число:
- Идентификатор в Си:

Примеры

- Цифра: $'0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9' \equiv [0-9]$
- Целое неотрицательное число(Z_0^+): $[0-9]^+$
- Целое число(Z): $'-'?[0-9]^+$
- Десятичное дробное число: $'-'?[0-9]^*'.'[0-9]^+$
- Идентификатор в Си:

Примеры

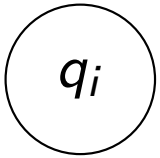
- Цифра: $'0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9' \equiv [0-9]$
- Целое неотрицательное число(Z_0^+): $[0-9]^+$
- Целое число(Z): $'-'?[0-9]^+$
- Десятичное дробное число: $'-'?[0-9]^*'.'[0-9]^+$
- Идентификатор в Си: $[a-zA-Z\_][0-9a-zA-Z\_]^*$

А имплементация?

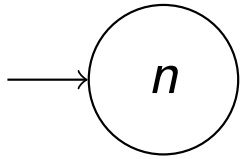
Для строки s и выражения e, $s \in L(e)$?

Конечный автомат

- Входной алфавит Σ
- Конечный набор состояний S



- Начальное состояние $n \in S$



- Набор конечных состояний $F \subseteq S$

- Набор переходов из состояния $s_1 \in S$ в состояние $s_2 \in S$ по входу $a \in \Sigma$: $s_1 \xrightarrow{a} s_2$

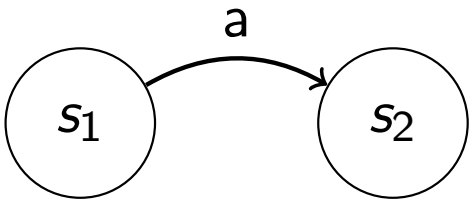
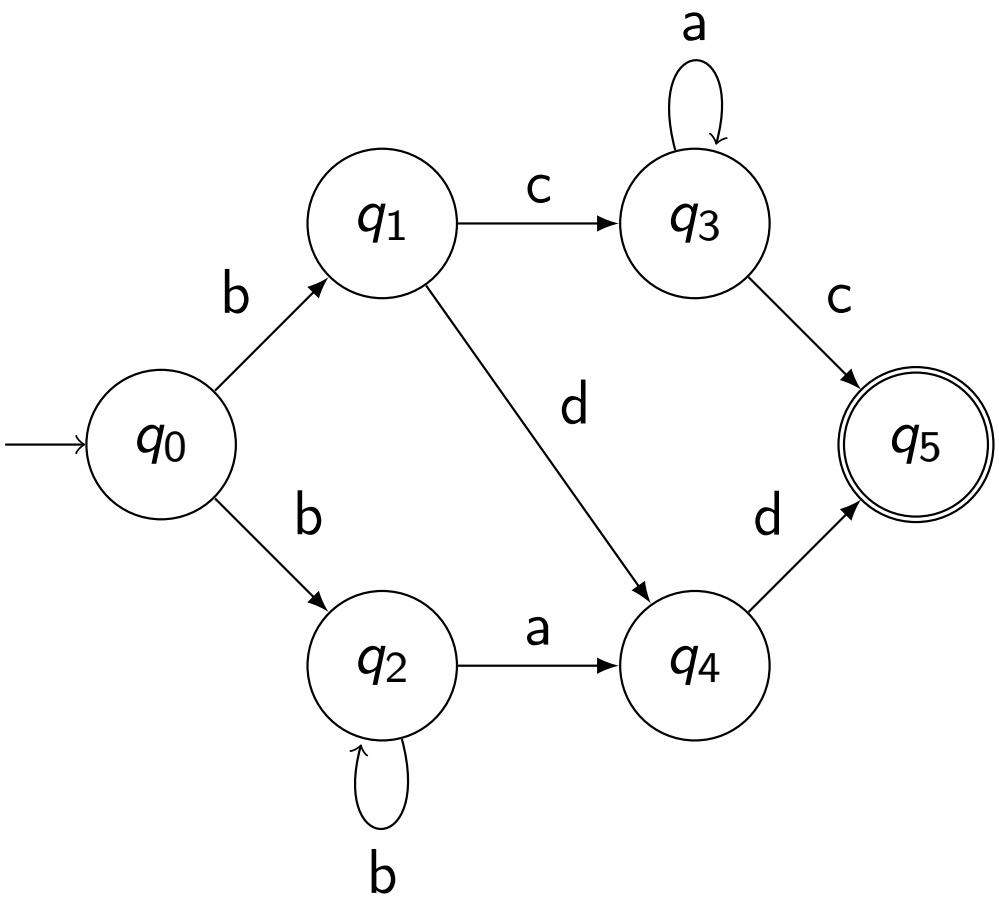
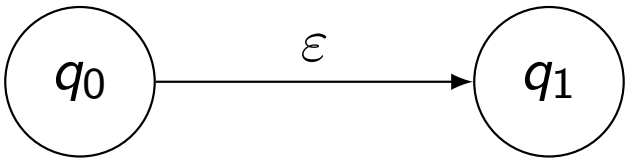


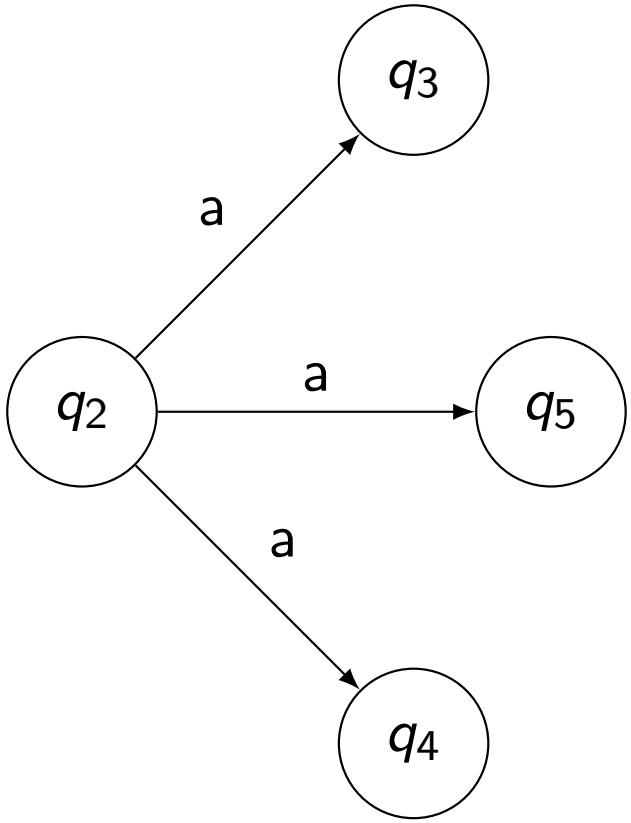
Таблица и диаграмма переходов



	a	b	c	d
q_0	$\varnothing$	q_1, q_2	$\varnothing$	$\varnothing$
q_1	$\varnothing$	$\varnothing$	q_3	q_4
q_2	q_4	q_2	$\varnothing$	$\varnothing$
q_3	q_3	$\varnothing$	q_5	$\varnothing$
q_4	$\varnothing$	$\varnothing$	$\varnothing$	q_5
q_5	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

множественные и ε- переходы





ДКА и НКА

Не детерменированный конечный автомат (НКА)

Конечный автомат, который может одновременно быть в нескольких состояниях.

Может содержать:

- ε-переходы
- множественные переходы

НКА как правило лаконичнее.

Детерменированный конечный автомат (ДКА)

Конечный автомат, находящийся в каждый момент времени только в одном состоянии.

Не содержит:

- ε-переходы
- множественные переходы

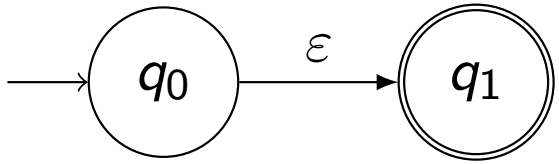
НКА может быть преобразован в ДКА принимающий тотже язык.

Общая схема генерации сканера

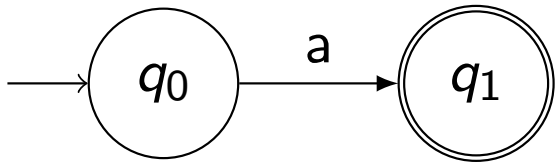
Построение НКА для регулярного выражения

Автоматы для базовых елементов:

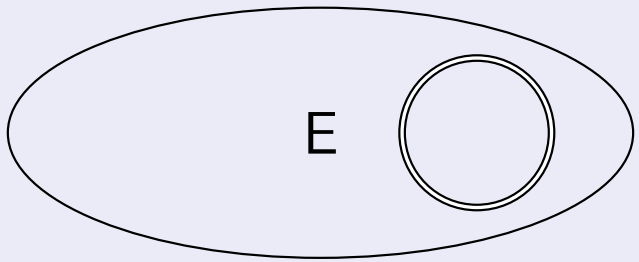
- ε (пустая строка):



- a$\in \Sigma$:



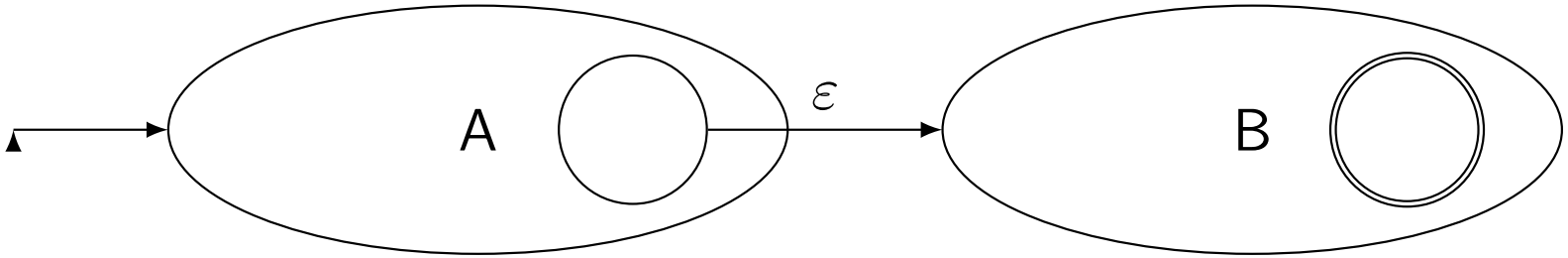
Общее обозначение НКА для выражения E



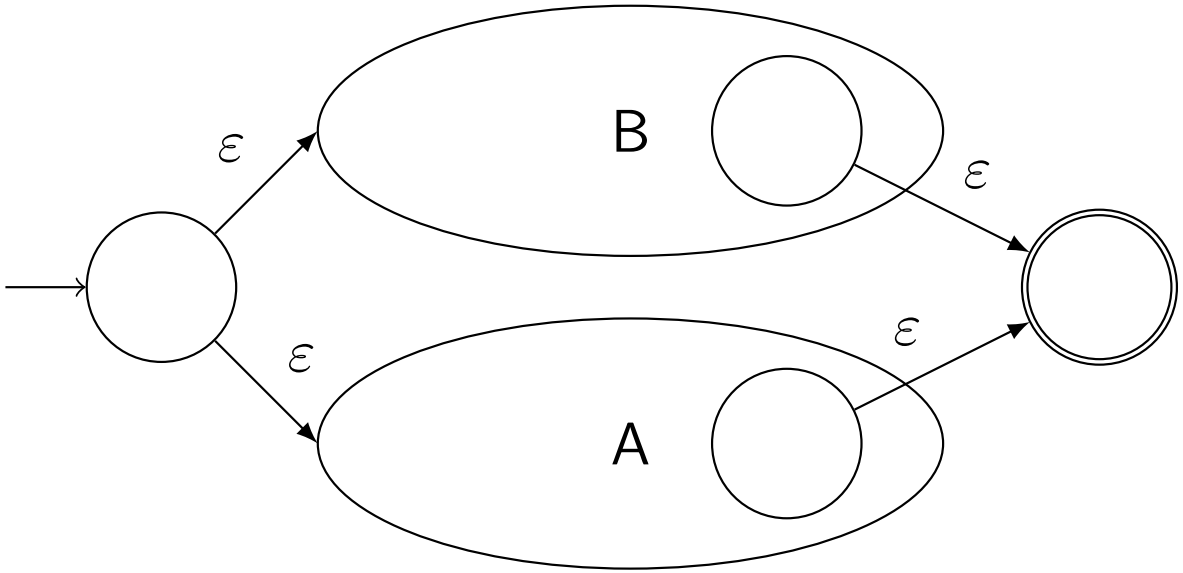
Построение НКА для регулярного выражения

Комбинации:

- AB:

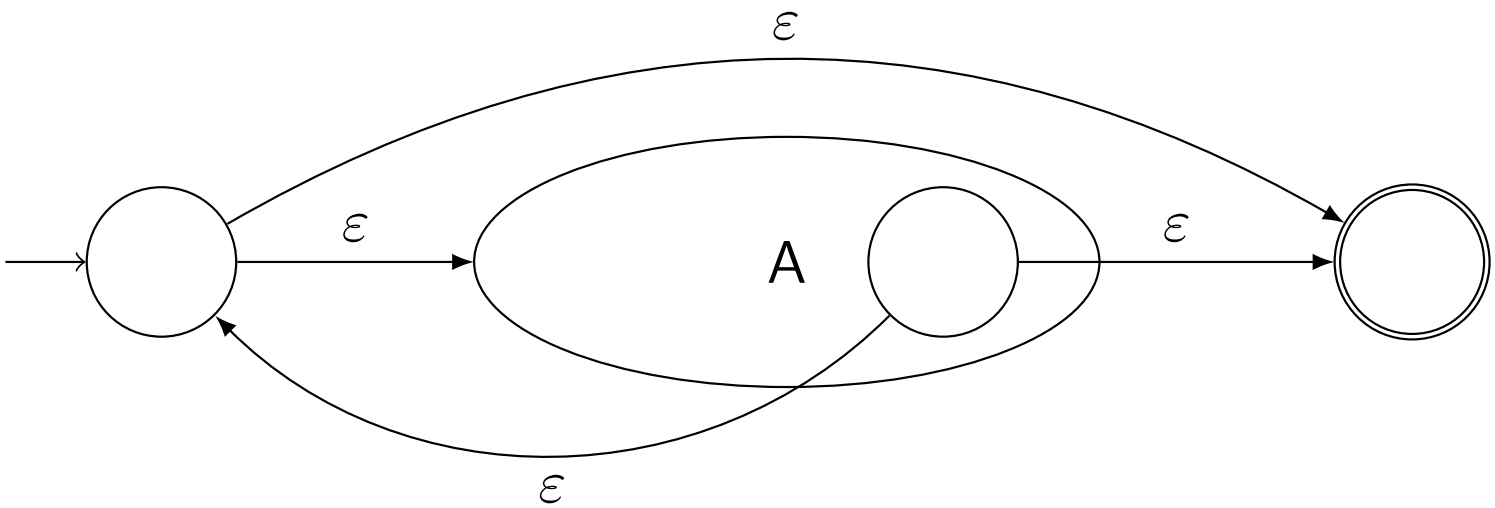


- $A + B$:



Построение НКА для регулярного выражения

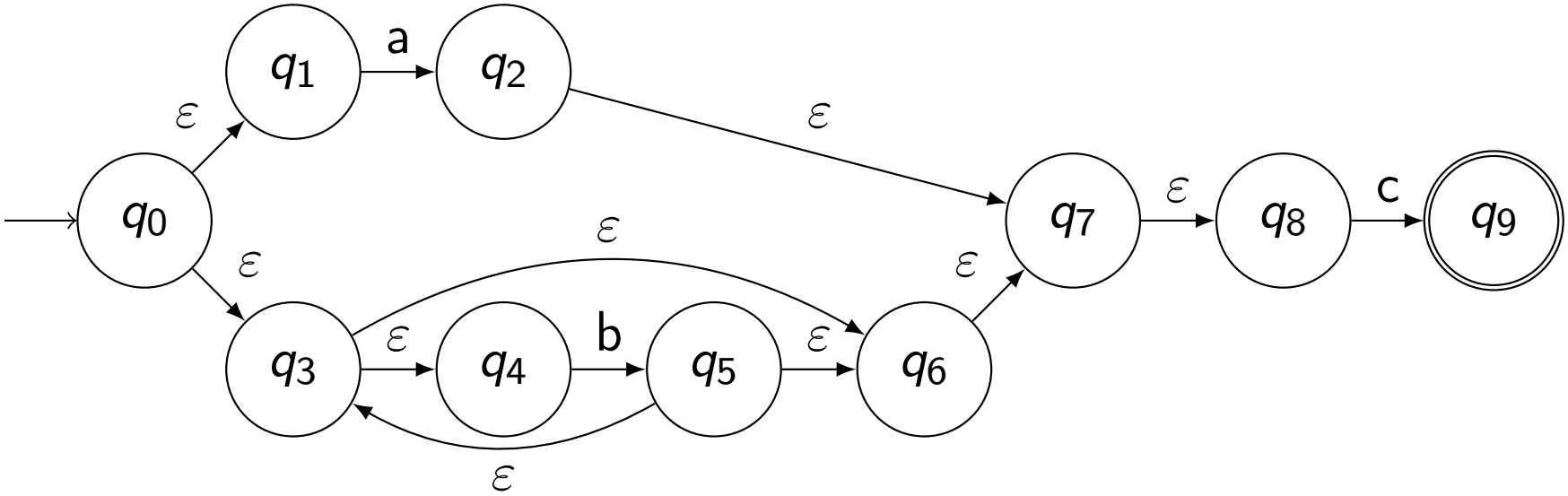
- A^*:



Пример

Выражение: $(a + b^*)c$

(Для удобства, вершины помечены)



Преобразование НКА в ДКА

ε-замыкание

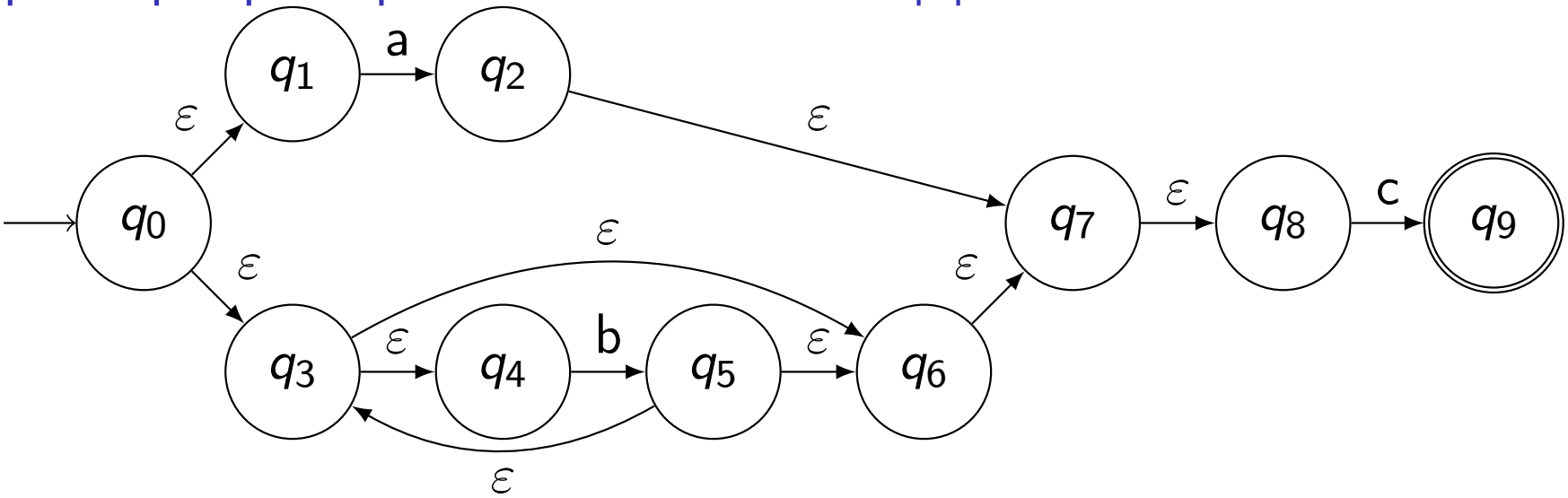
ε-замыкание — Множество состояний НКА, в которые можно перейти из данного по цепочке ε-переходов.

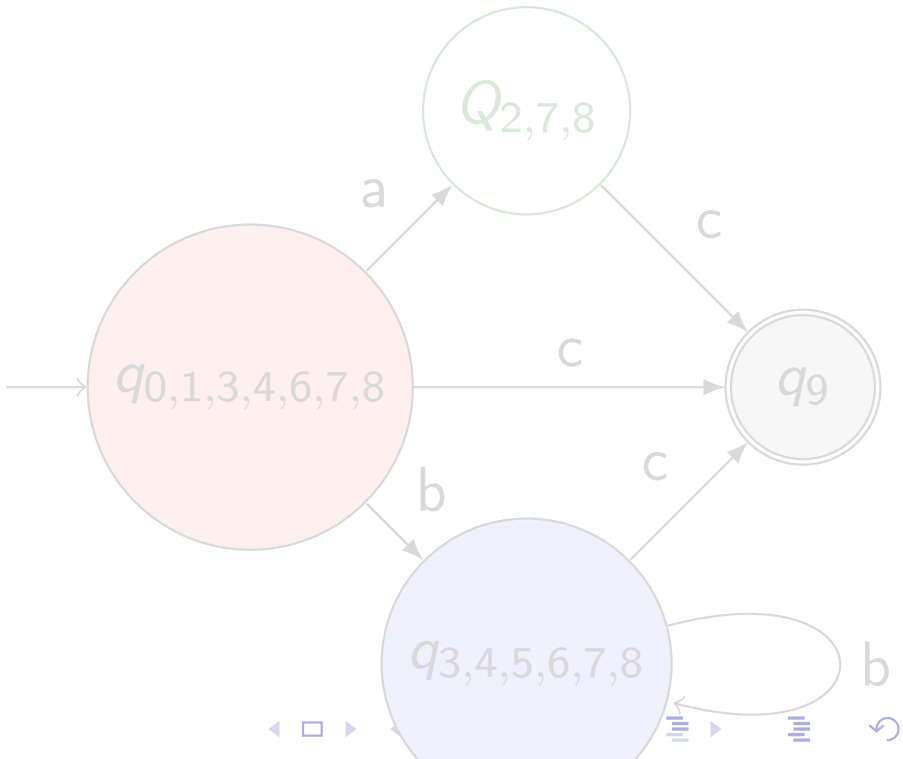
$$\varepsilon closure(q_i) = \{q_i\} \cup \{\varepsilon closure(q_j) | \exists q_i \xrightarrow{\varepsilon} q_j\}$$

Пусть S – множество состояний НКА, s – его начальное состояние, F – все конечные.

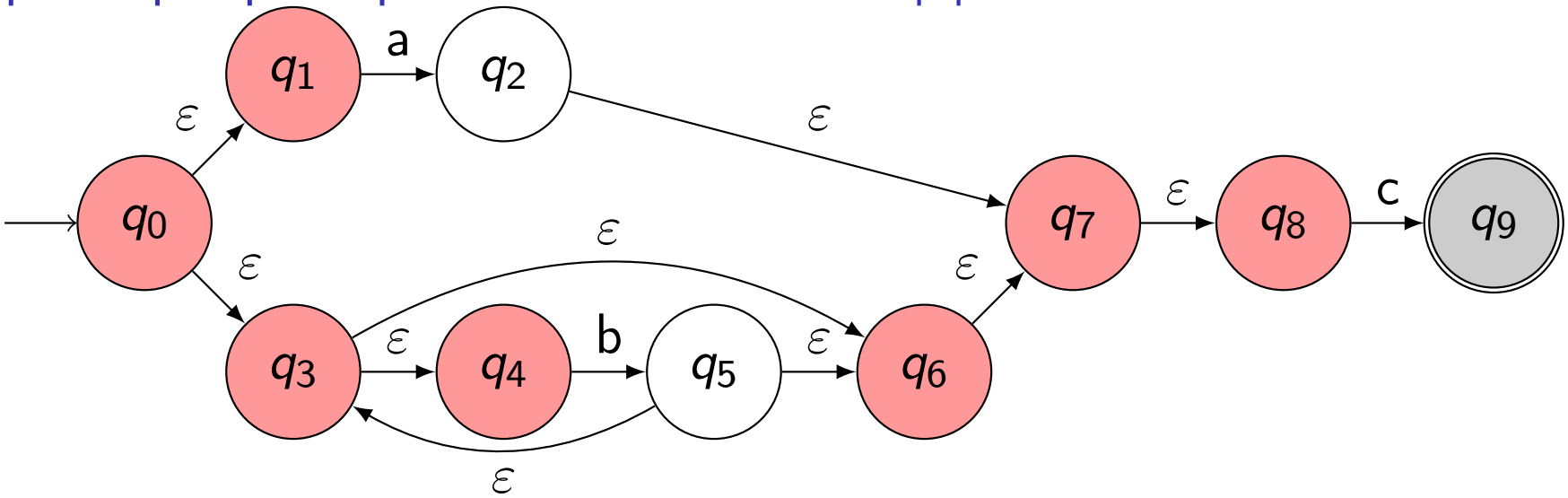
- Каждое состояние ДКА (X) — это подмножество (S) состояний НКА. $X \subseteq S$
- Начальное состояние: $\varepsilon closure(s)$
- Конечные состояния: $\{X | X \cap F \neq \varnothing\}$
- Переходы: $\exists X \xrightarrow{a} Y \Leftrightarrow Y = \bigcup\limits_{q_i} \varepsilon closure(q_i), q_i \in X$

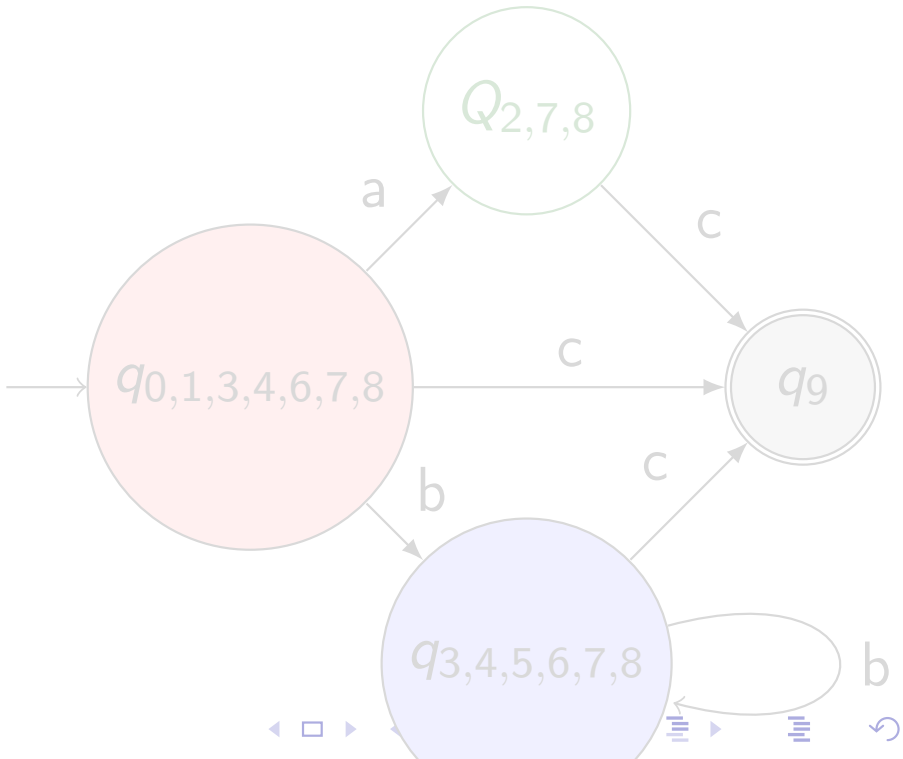
Пример преобразования НКА в ДКА



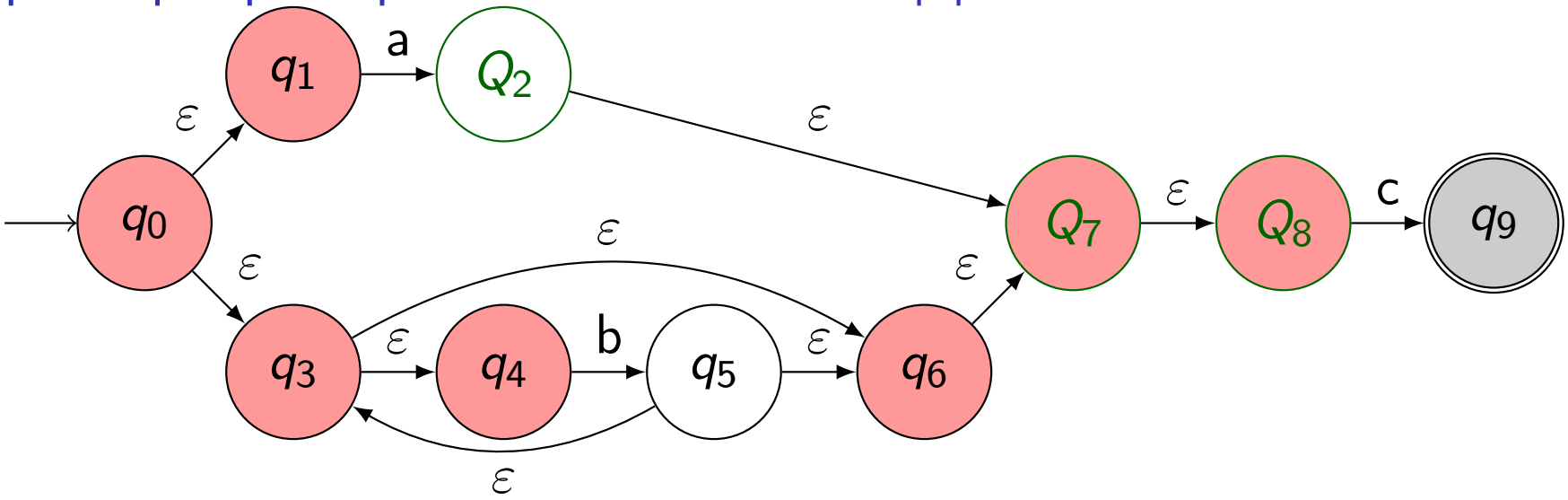


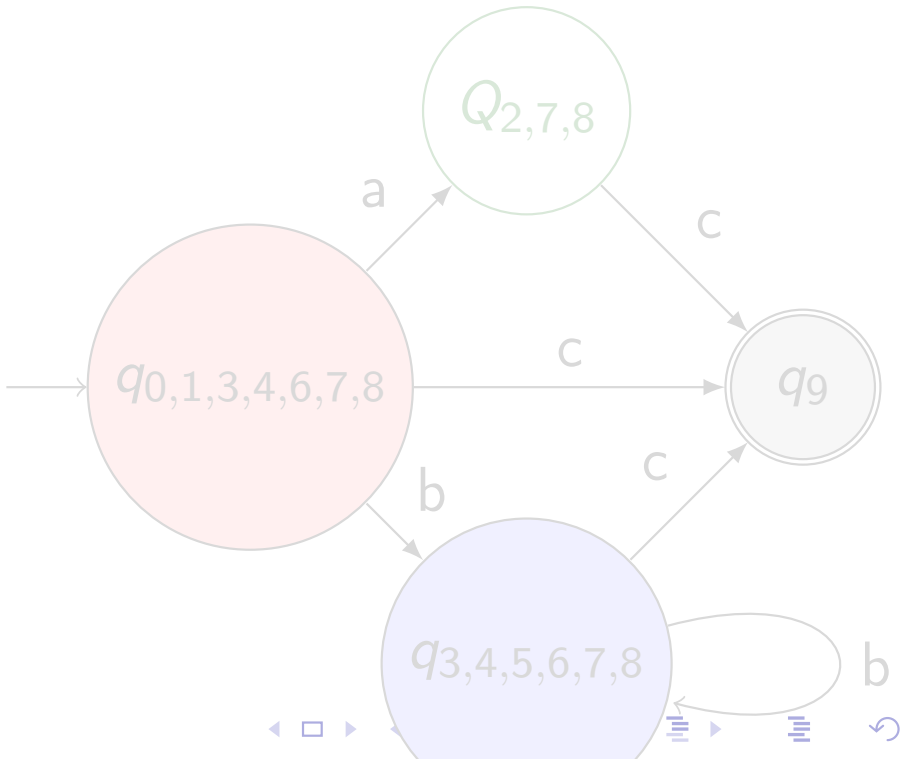
Пример преобразования НКА в ДКА



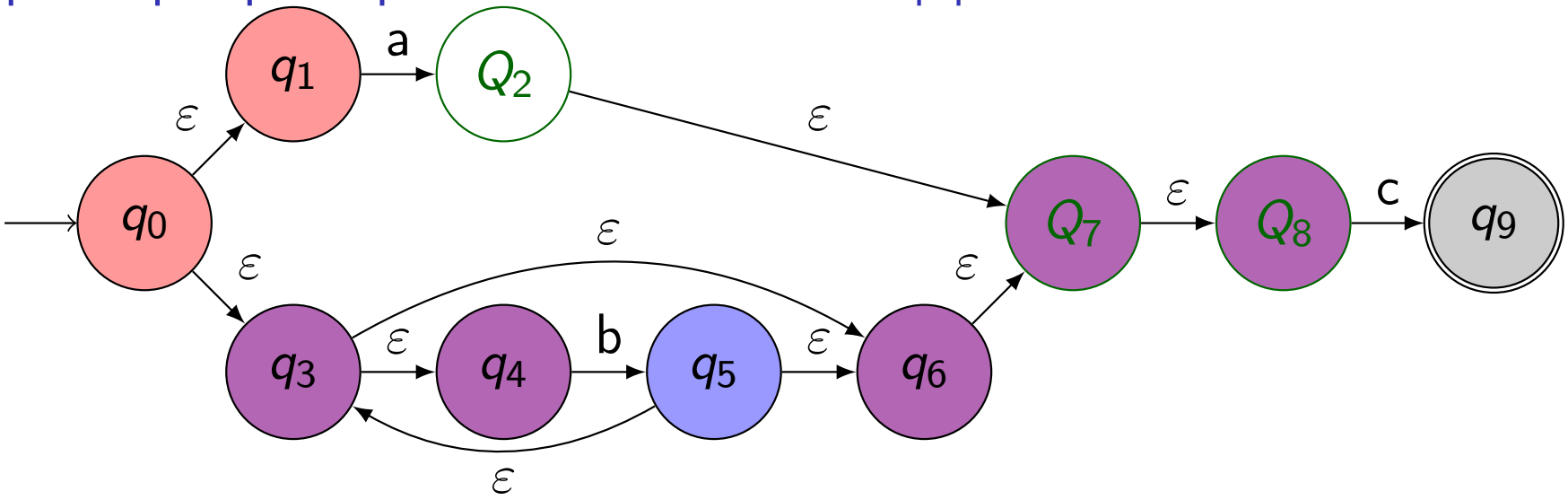


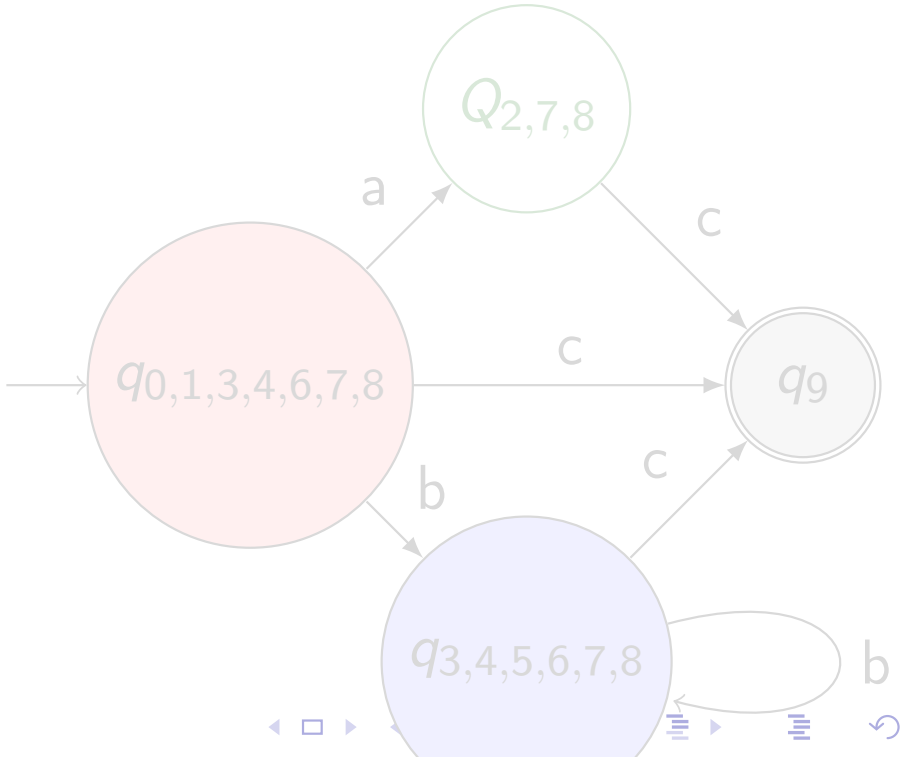
Пример преобразования НКА в ДКА



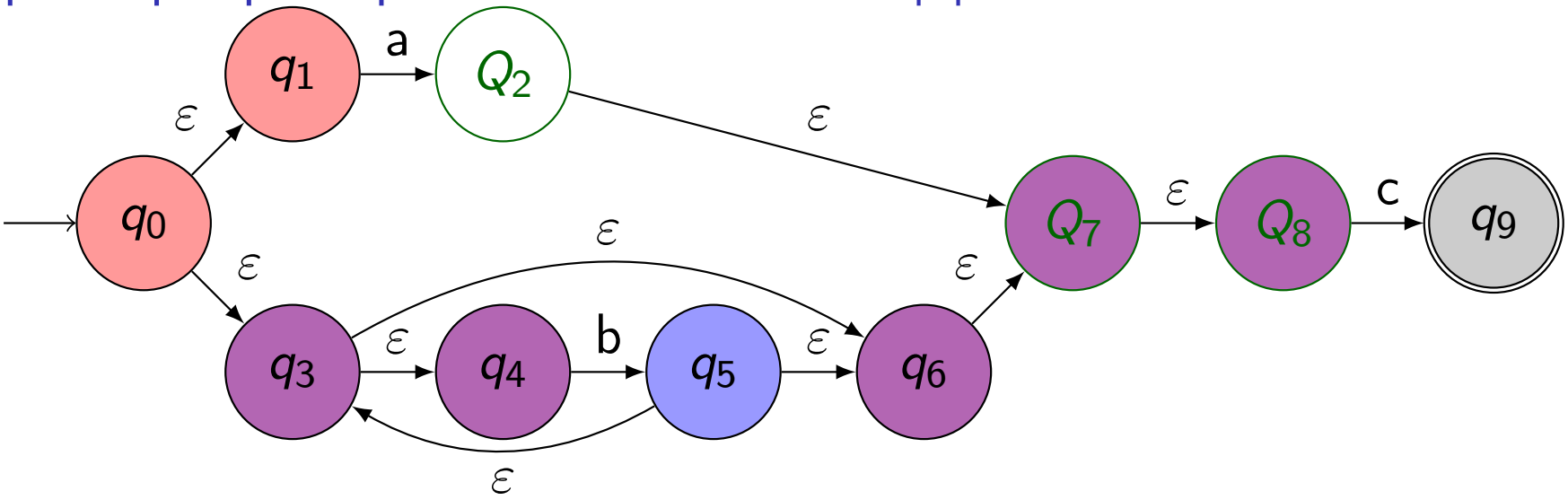


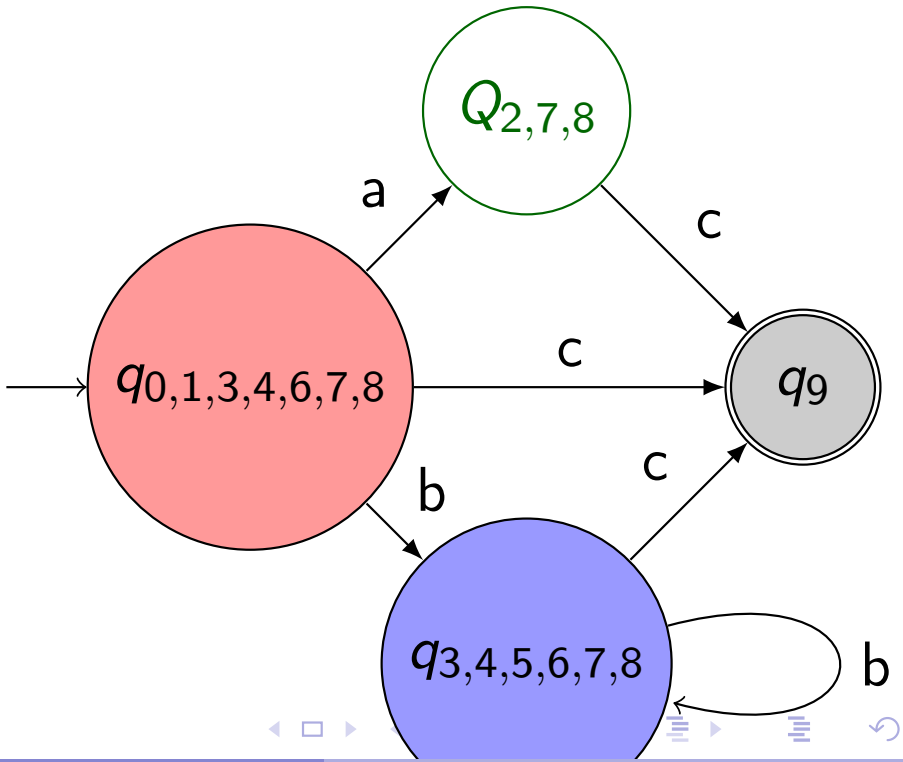
Пример преобразования НКА в ДКА





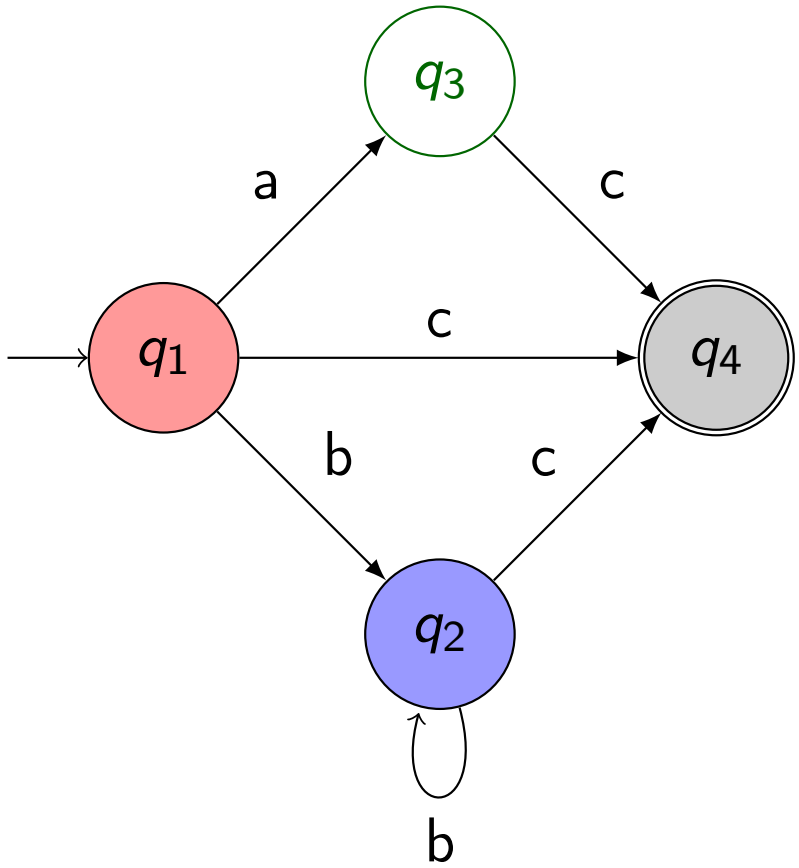
Пример преобразования НКА в ДКА





Табличная имплементация

A	a	b	c
q_1	q_3	q_2	q_4
q_2	$\varnothing$	q_2	q_4
q_3	$\varnothing$	$\varnothing$	q_4
q_4	$\varnothing$	$\varnothing$	$\varnothing$




```
(define (go state input)
  (let ((next (gethash '(state input))))
      (if next
          (go next (move (forward input)))
         #f)))
(define (accept input)
  (let ((rez (go n input)))
    (and rez (accepting? rez))))
```