Department of Computer & Information Sciences
Pakistan Institute of Engineering and Applied Sciences

Regular Expressions

Based on slides by Costas Busch [http://csc.lsu.edu/~busch]

Umar Faiz

http://www.pieas.edu.pk/umarfaiz/cis505

Regular Expressions

- A regular expression (RE) is a pattern that describes some set of strings.
- Regular expressions describe regular languages.
- A language generator model instead of language acceptor.

Regular Expressions

Example:

$$(a + b \cdot c)*$$

describes the language

$$\{a, bc\}* = \{\lambda, a, bc, aa, abc, bca, ...\}$$

Regular Expressions

- Definition: A regular expression over an alphabet Σ is recursively defined as follows:
  1. ø denotes language ø
  2. ε denotes language {ε}
  3. a denotes language {a}, for all $a \in \Sigma$.
  4. (P + Q) denotes L(P) U L(Q), where P, Q are r.e.'s.
  5. (PQ) denotes $L(P) \cdot L(Q)$, where P, Q are r.e.'s.
  6. P* denotes L(P)*, where P is r.e.

Regular Expressions

- Operations on Regular Expressions

RE	Regular Language Description
a+b	{a,b}
(a+b) (a+b)	{aa, ab, ba, bb}
a*	{λ, a, aa, aaa, ...}
a*b	{b,ab, aab, aaab, ...}
(a+b)*	{λ, a, b, aa, ab, ba, aaa, bbb...}

Regular Expressions

Primitive regular expressions: $\emptyset, \quad \lambda, \quad \alpha$

Given regular expressions r_1 and r_2

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 \cdot r_2 \\ r_1 * \\ (r_1) \end{array}\right\} \text{are regular expressions}$$

Regular Expressions

A regular expression: $(a+b\cdot c)^{*}\cdot(c+\varnothing)$

Not a regular expression: $(a+b+)$

Languages of Regular Expressions

- If r is a regular expression, we will let L(r) denote the language associated with r.
- For primitive regular expressions $\emptyset, \quad \lambda, \quad \alpha:$

$$L(\emptyset) = \emptyset$$

$$L(\lambda) = \{\lambda\}$$

$$L(a) = \{a\}$$

Languages of Regular Expressions

$L(r)$: Language of regular expression r

Example:

$$L((a+b\cdot c)^*)=\{\lambda,a,bc,aa,abc,bca,...\}$$

Languages of Regular Expressions

For regular expressions r_1 and r_2

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1)\, L(r_2)$$

$$L(r_1 *) = (L(r_1))*$$

$$L((r_1)) = L(r_1)$$

Languages of Regular Expressions

Example:

Regular expression: $(a+b)\cdot a*$

$$
\begin{aligned}
L((a+b)\cdot a*) &= L((a+b))\,L(a*) \\
&= L(a+b)\,L(a*) \\
&= (L(a)\cup L(b))\,(L(a))* \\
&= (\{a\}\cup\{b\})\,(\{a\})* \\
&= \{a,b\}\,\{\lambda,a,aa,aaa,\ldots\} \\
&= \{a,aa,aaa,\ldots,b,ba,baa,\ldots\}
\end{aligned}
$$

Languages of Regular Expressions

Example:

Regular expression $$r = (a+b)^*(a+bb)$$

$$L(r) = \{a, bb, aa, abb, ba, bbb, \ldots\}$$

Languages of Regular Expressions

Example:

Regular expression $\quad r = (aa)^*(bb)^*b$

$$L(r) = \{a^{2n}b^{2m}b : \quad n, m \geq 0\}$$

Languages of Regular Expressions

Example:

Regular expression $r = (0+1)^*\,00\,(0+1)^*$

$L(r)$ = { all strings with at least two consecutive 0's }

Equivalent Regular Expressions

Definition:

Regular expressions r_1 and r_2

are equivalent if

$$L(r_1) = L(r_2)$$

Equivalent Regular Expressions

Example:

L = { all strings without two consecutive 0 }

$$r_1 = (1 + 01)^*(0 + \lambda)$$

$$r_2 = (1^*011^*)^*(0 + \lambda) + 1^*(0 + \lambda)$$

$L(r_1) = L(r_2) = L$ $\Rightarrow$ r_1 and r_2 are equivalent regular expressions

Algebraic Laws for REs

Axiom	**Description**
$r + s = s + r$	+ is commutative
$(r + s) + t = r + (s + t)$	+ is associative
$(rs)t = r(st)$	concatenation is associative
$r(s+t) = rs + rt$ $(s+t)r = sr + tr$	concatenation distributes over +
$\lambda r = r$ $r\lambda = r$	λ is the identity element for concatenation
$r^* = (r + \lambda)^*$	relation between * and λ
$r^{**} = r^*$	* is idempotent, i.e., taking the closure of a regular expression under closure does not change the language

Algebraic Laws for REs

- Laws Involving Closures
  - $(L^*)^* = L^*$
    - i.e., taking the closure of a regular expression under closure does not change the language
  - $\phi^* = \varepsilon$
  - $\varepsilon^* = \varepsilon$
  - $L^+ = LL^* = L^*L$
  - $L^* = L^+ + \varepsilon$

Algebraic Laws for REs

- Operator Precedence
  1. Kleene star (*)
  2. Concatenation (.)
  3. Union (+)

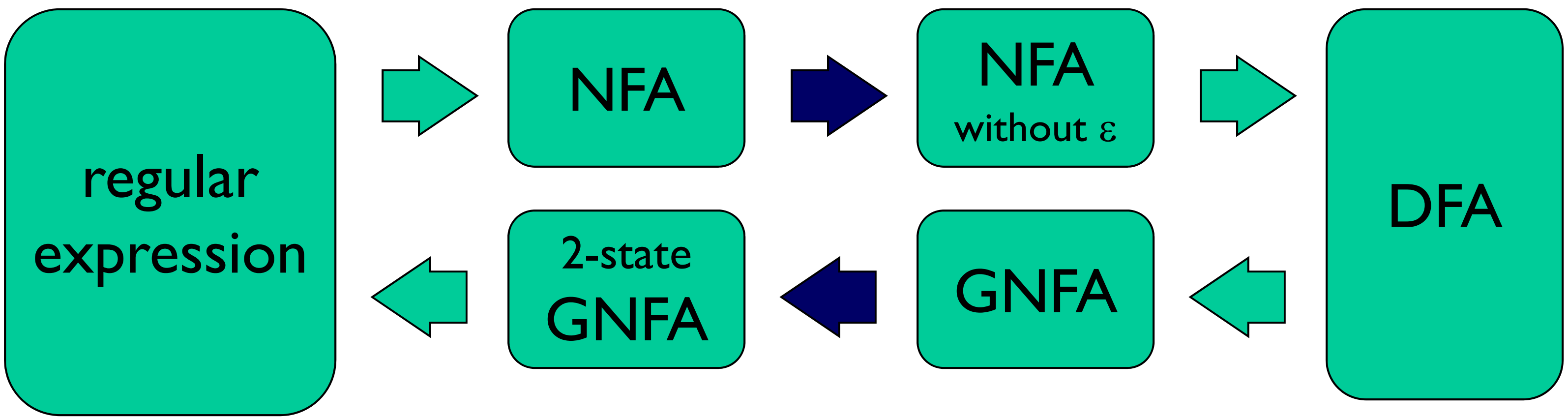
Equivalence of RE and Finite Automata

Finite Automata and Regular Expressions are equivalent.

1. There is an algorithm for converting any RE into an NFA.
2. There is an algorithm for converting any NFA to a DFA.
3. There is an algorithm for converting any DFA to a RE.

These facts tell us that REs, NFAs and DFAs have equivalent expressive power. All three describe the class of regular languages.

Roadmap



Converting Regular Expressions to NFAs

RE to ε-NFAs

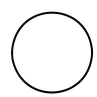
- We can convert a Regular Expression to a finite automaton.

- We can do this easiest by converting a RE to epsilon-NFA. Recursively build the FSA, mimicking the structure of the regular expression. Each FSA built has one start state, and one final state.

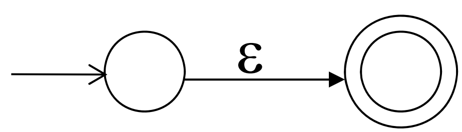
Converting RE to ε-NFAs

The regular expressions over finite Σ are the strings over the alphabet Σ such that:

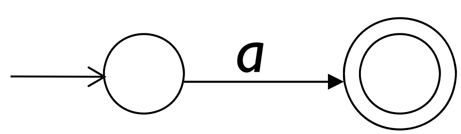
- { } (empty set) is a regular expression for empty set



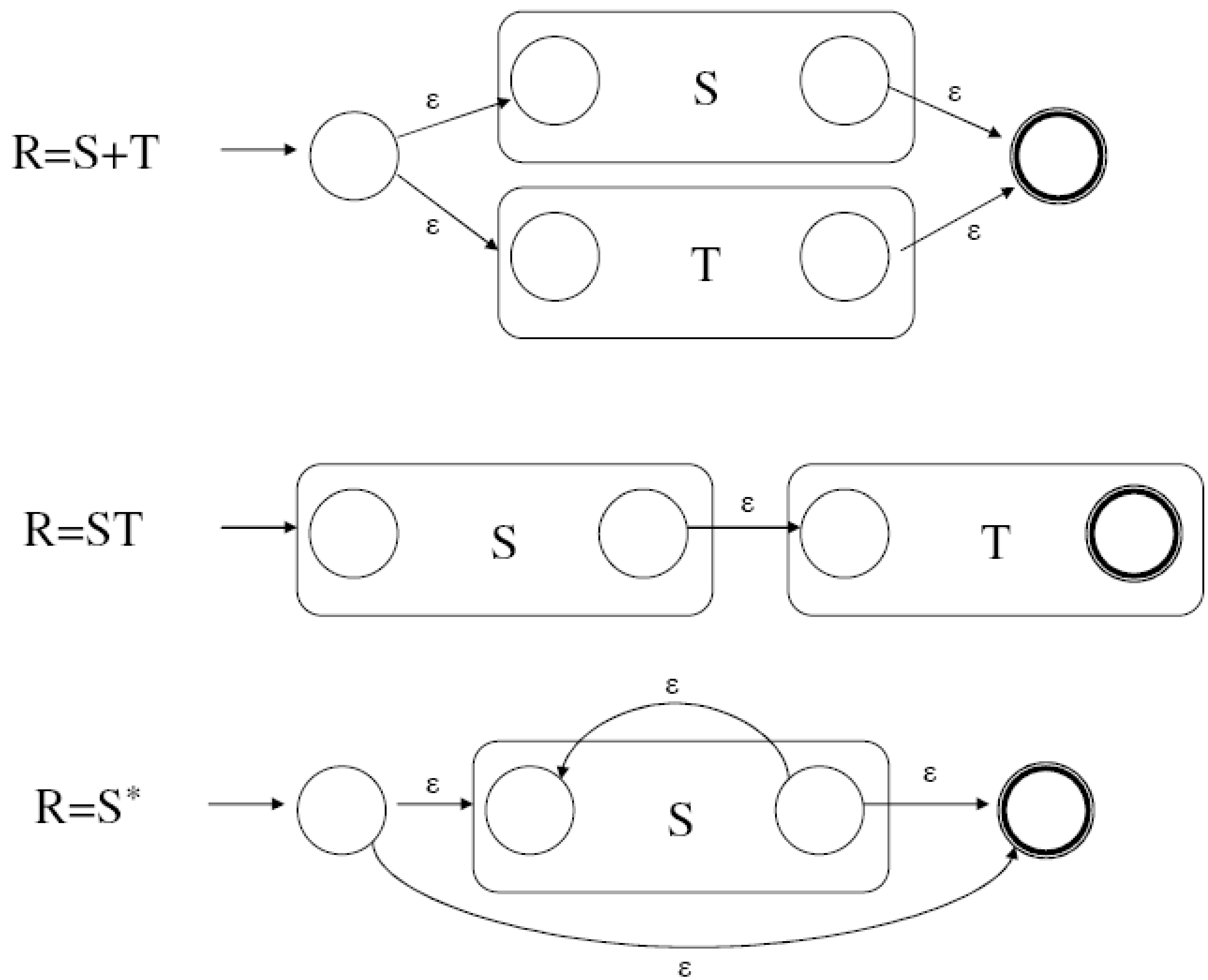
- Empty string ε is a regular expression denoting { ε }



- *a* is a regular expression denoting {*a* } for any *a* in Σ

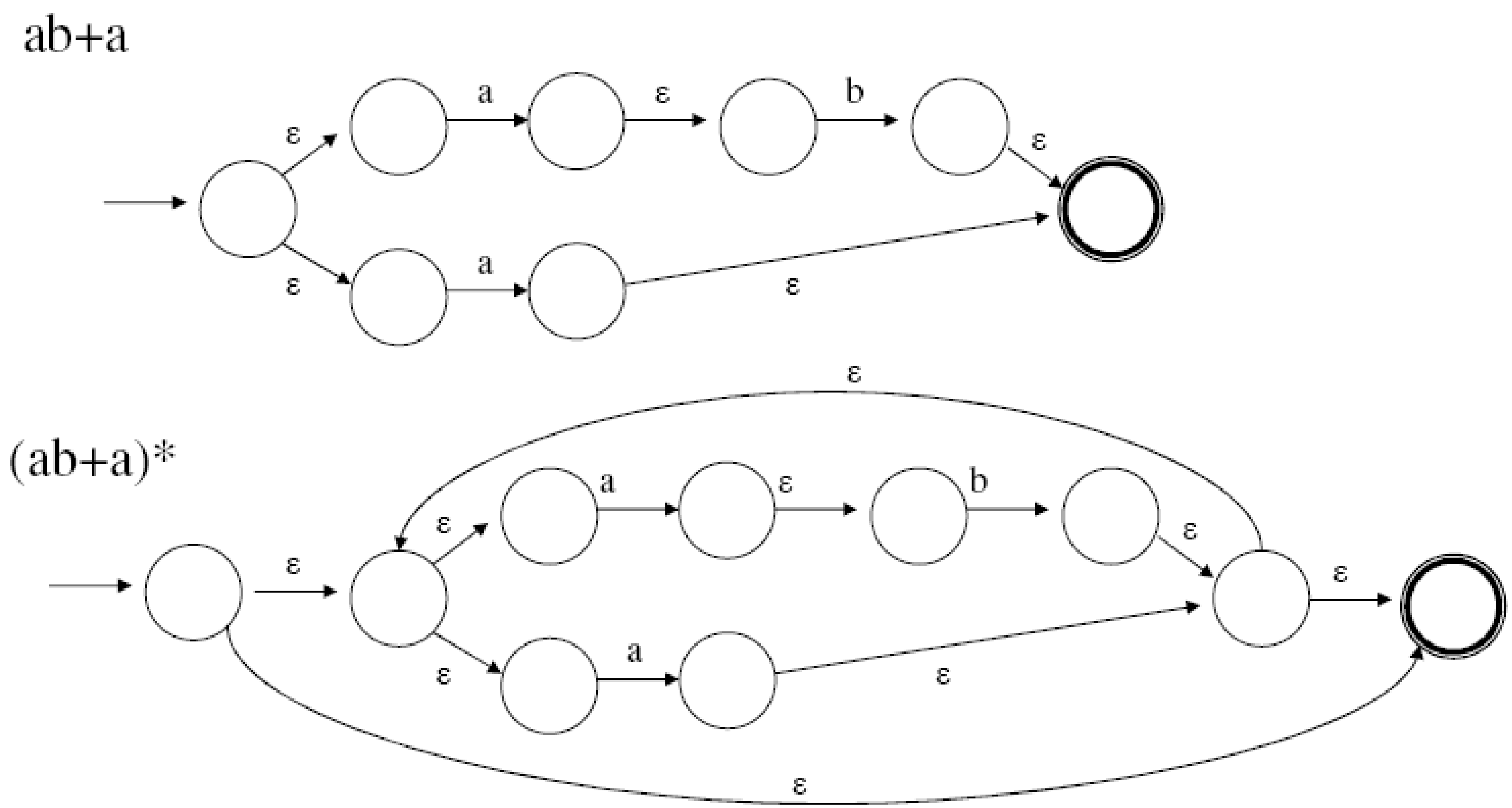


Converting RE to ε-NFAs



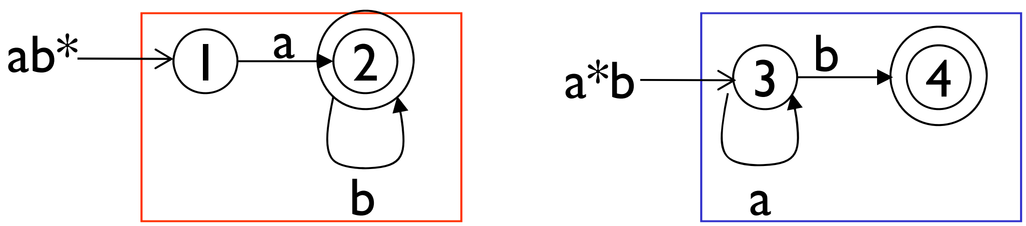
Converting RE to ε-NFAs

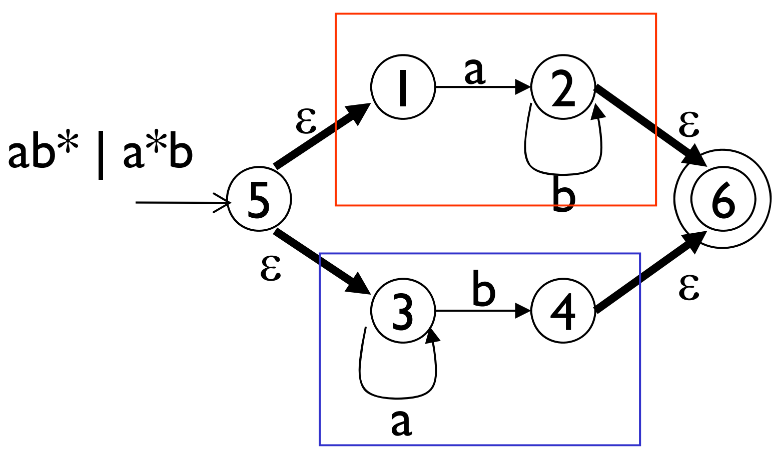
- Example 1: Convert ab+a and (ab+a)* into an NFA



Converting RE to ε-NFAs

Example 2: Convert (ab* | a*b)* into an NFA

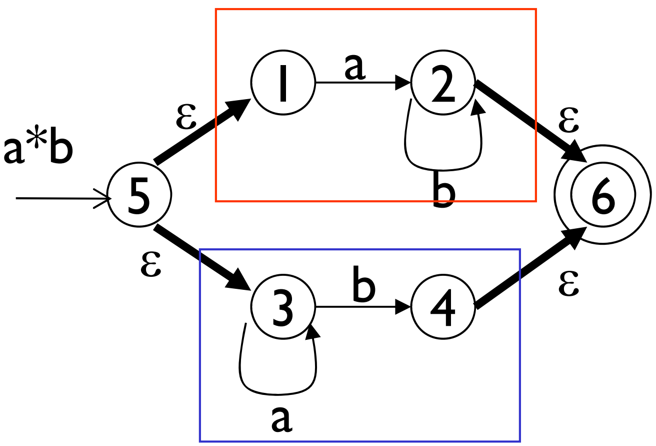




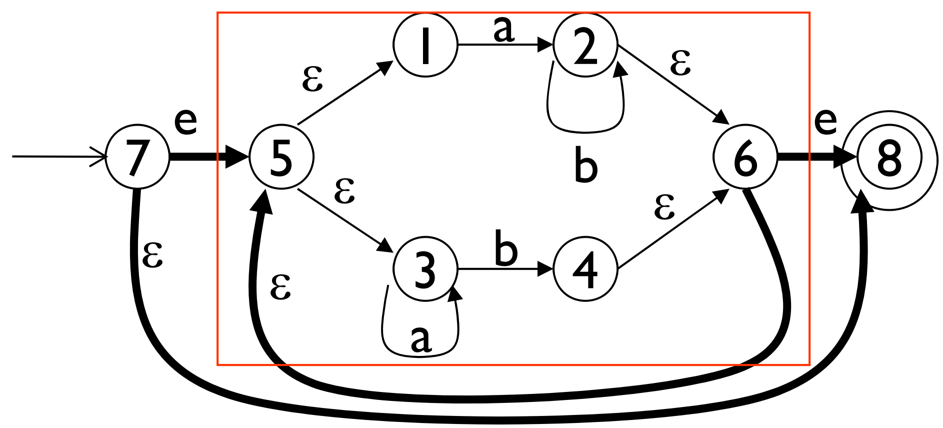
Converting RE to ε-NFAs

Example 2: Convert (ab* | a*b)* into an NFA

ab* | a*b



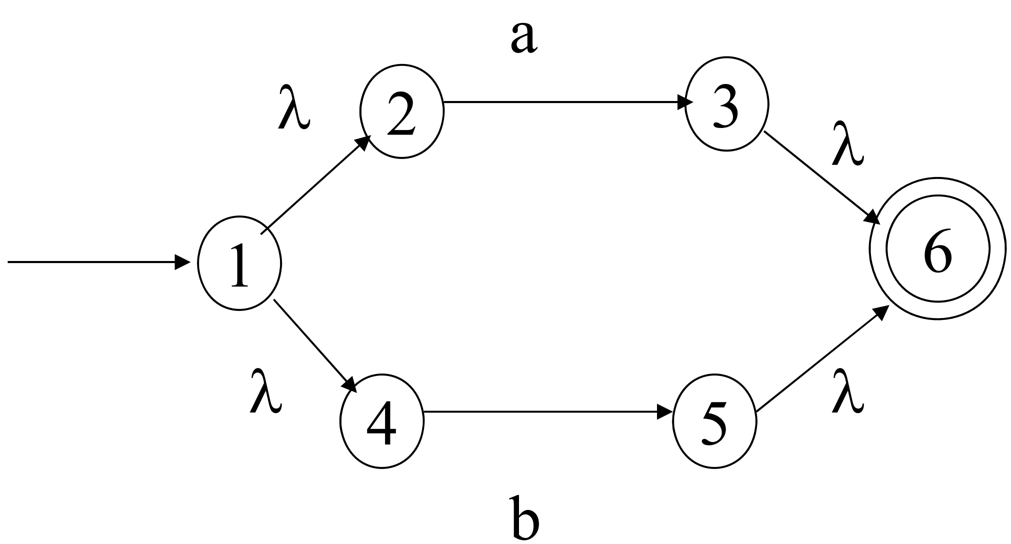
(ab* | a*b)*



Converting RE to ε-NFAs

Example 3: Convert (a+b)*abb into an NFA

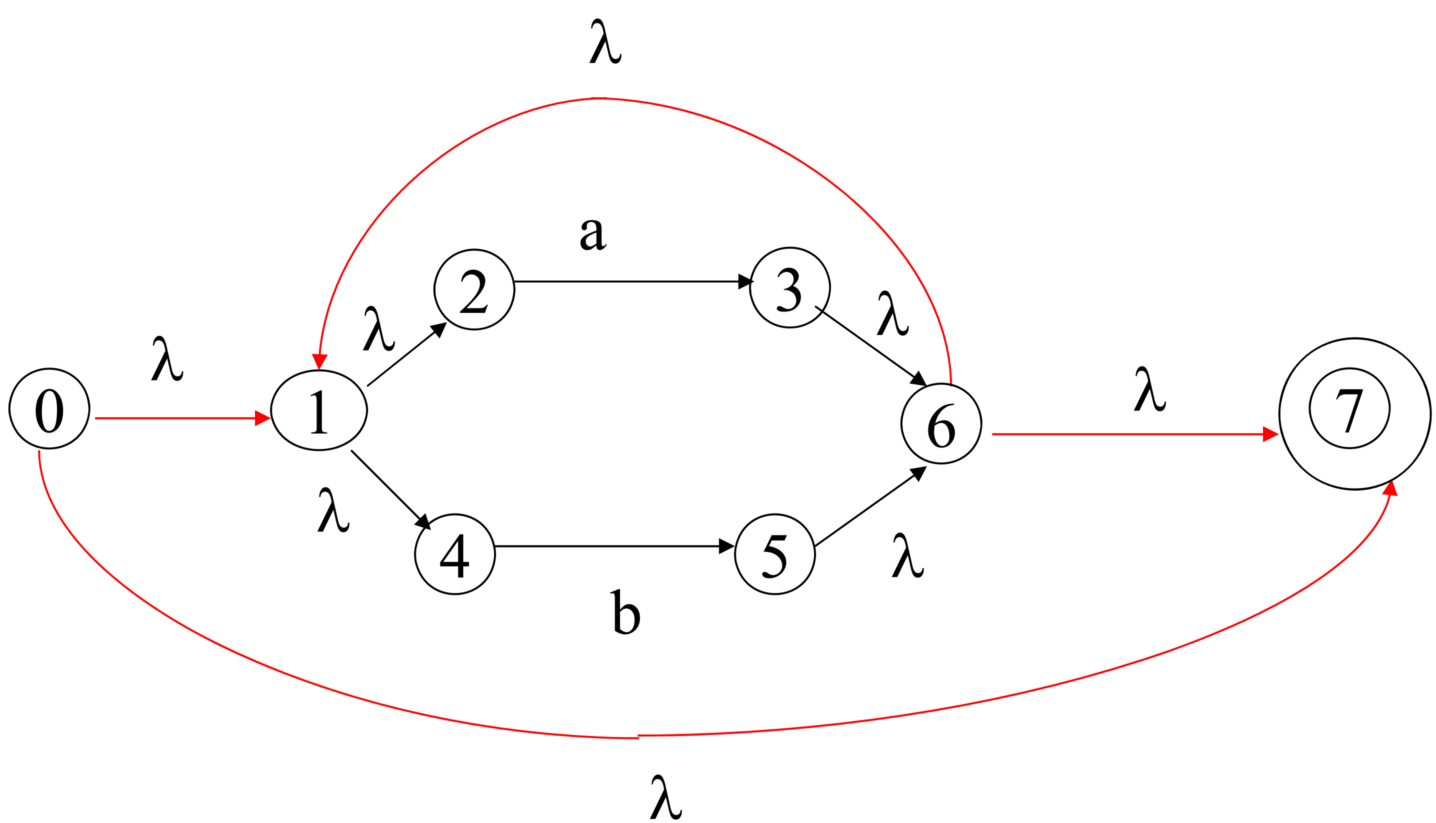
(a+b)



Converting RE to ε-NFAs

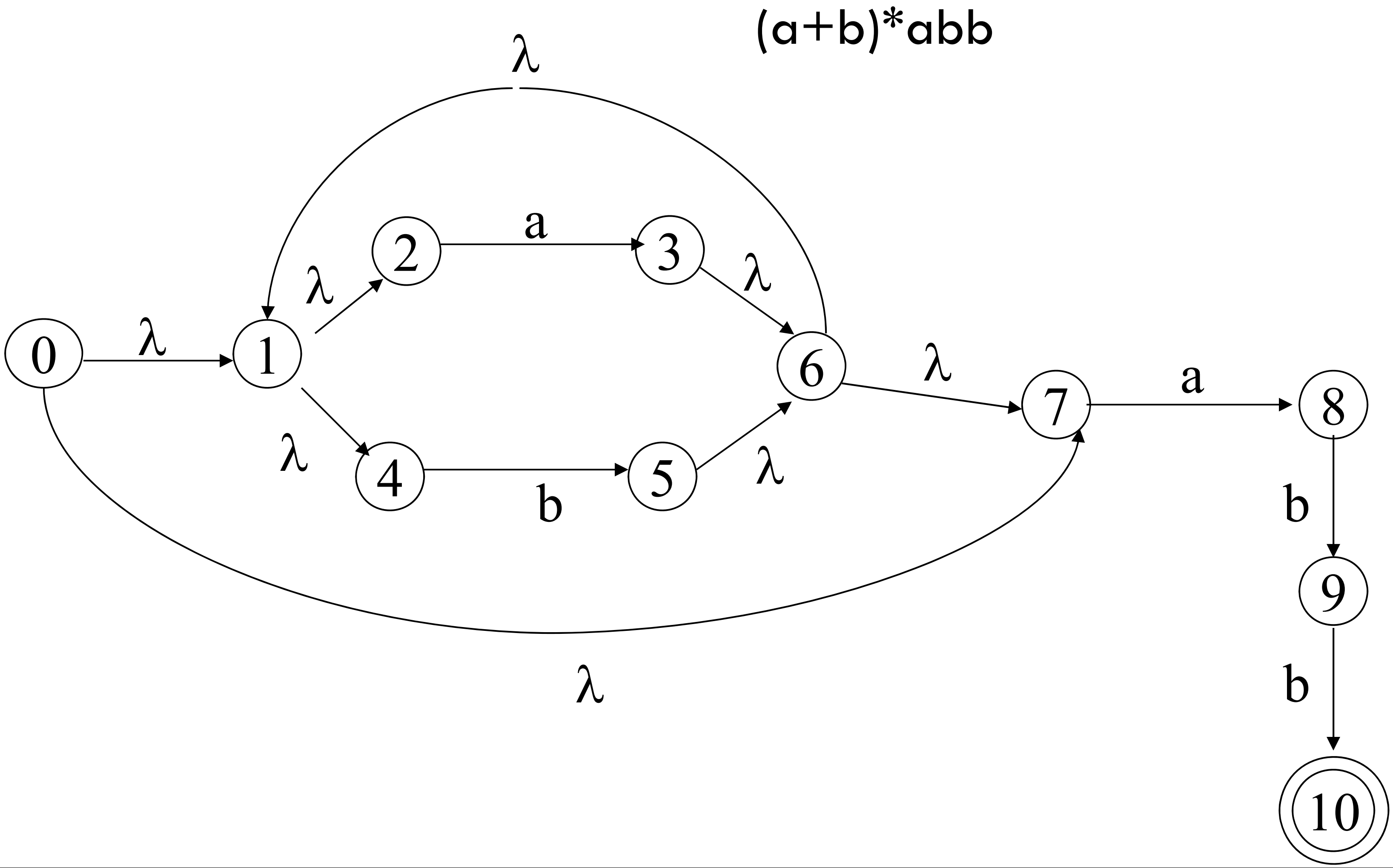
Example 3: Convert (a+b)*abb into an NFA

(a+b)*

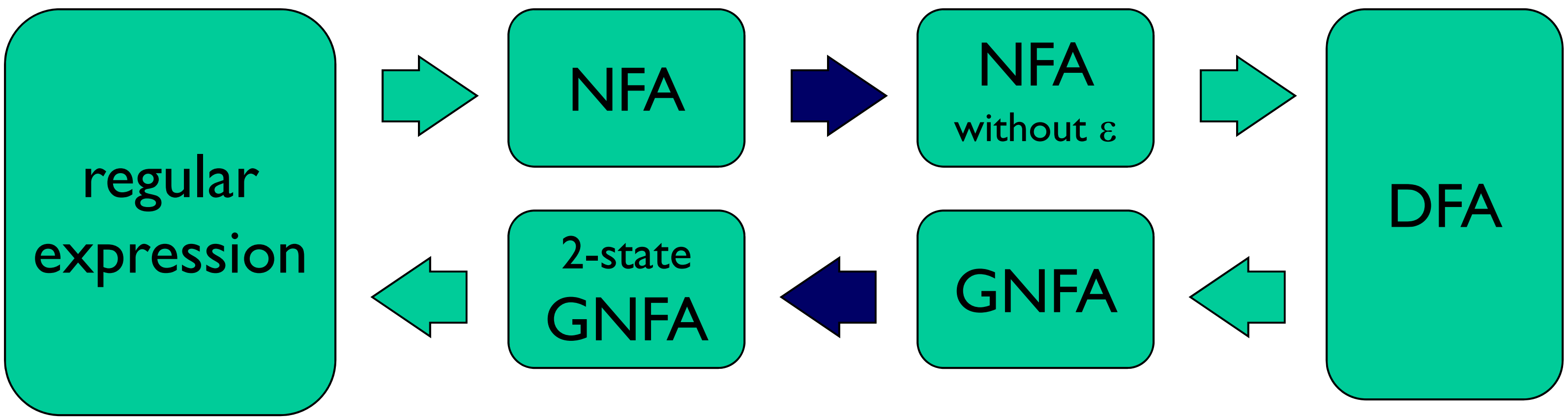


Converting RE to ε-NFAs

Example 3: Convert (a+b)*abb into an NFA

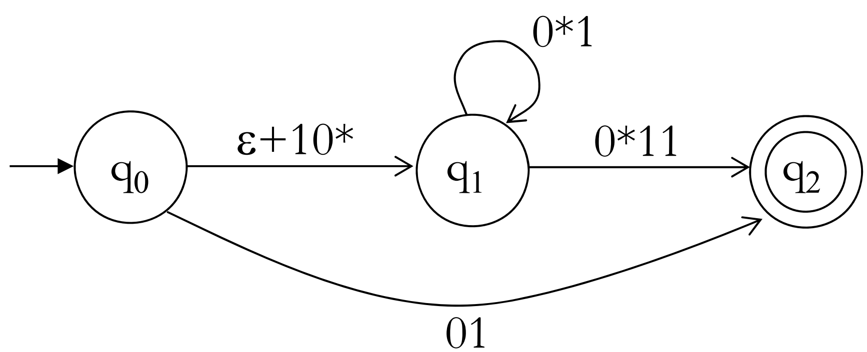


Roadmap



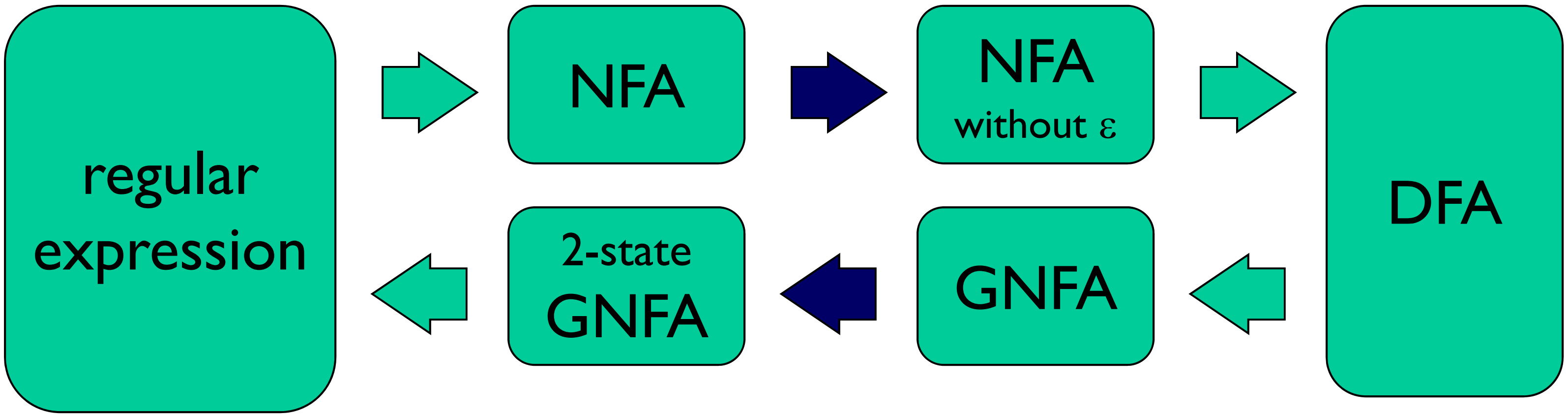
Generalized NFAs

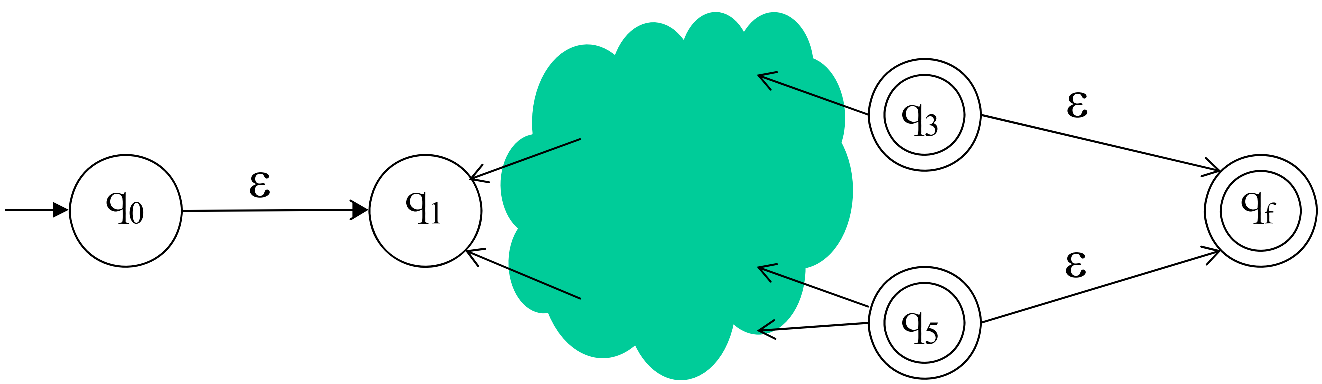
- A generalized NFA is an NFA whose transitions are labeled by regular expressions, like



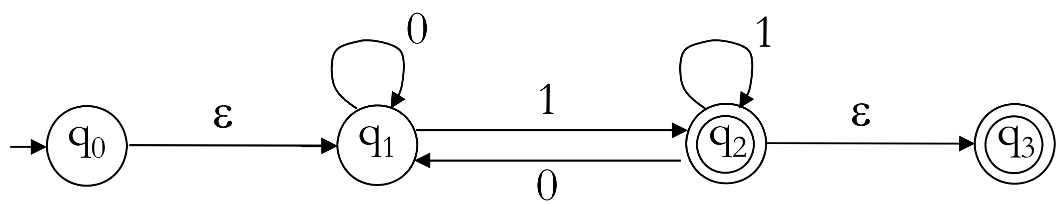
- Moreover
  - It has exactly one accept state, different from its start state
  - No arrows come into the start state
  - No arrows go out of the accept state

Converting a DFA to a GNFA



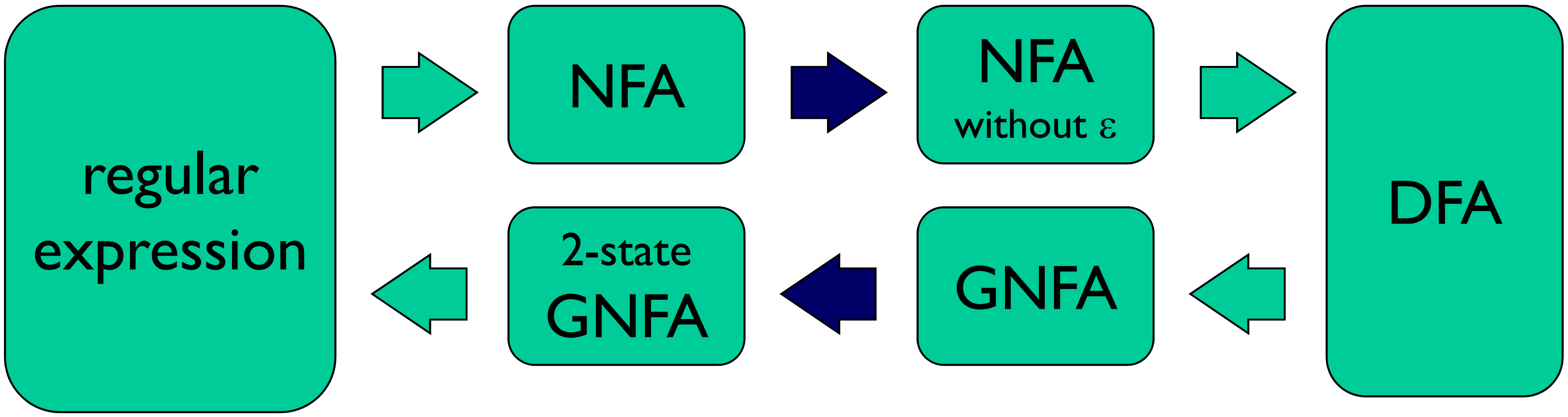


Conversion Example



- ✓ It has exactly one accept state, different from its start state
- ✓ No arrows come into the start state
- ✓ No arrows go out of the accept state

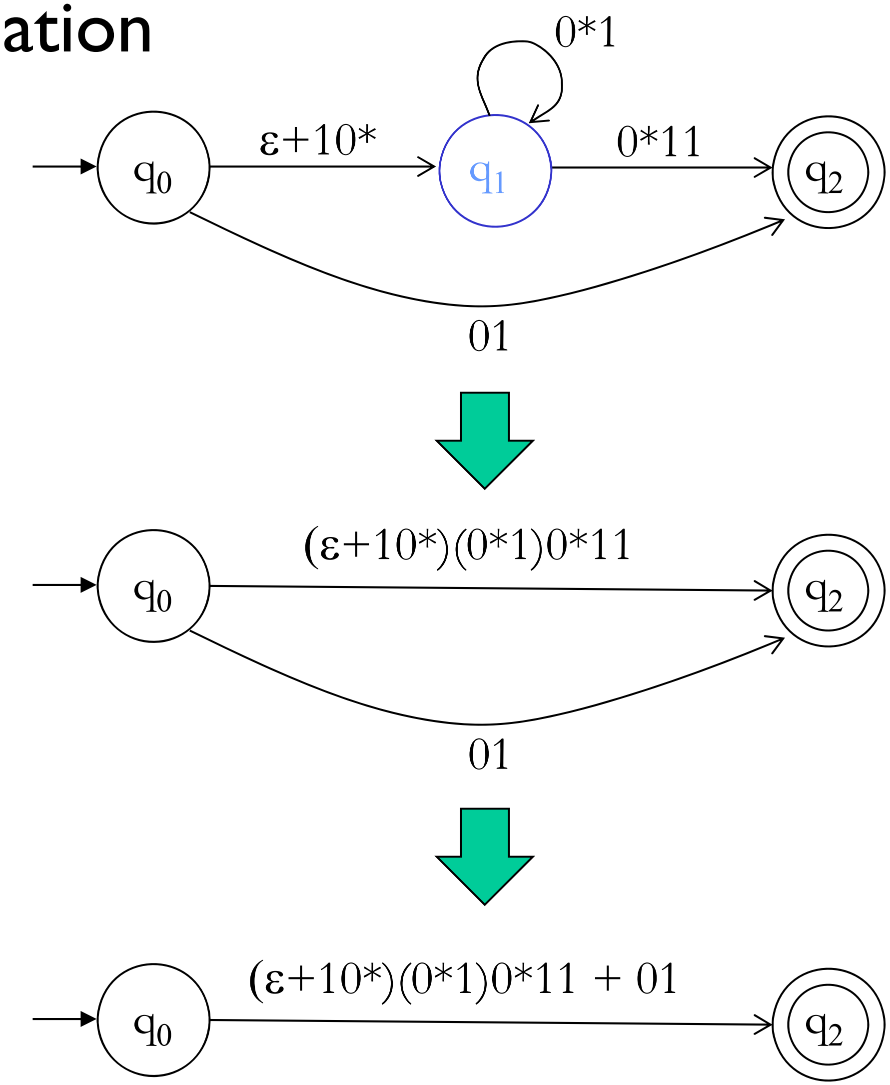
Converting a GNFA to a 2-state GNFA



From any GNFA, we can eliminate every state but the start and accept states

Converting a GNFA to a 2-state GNFA

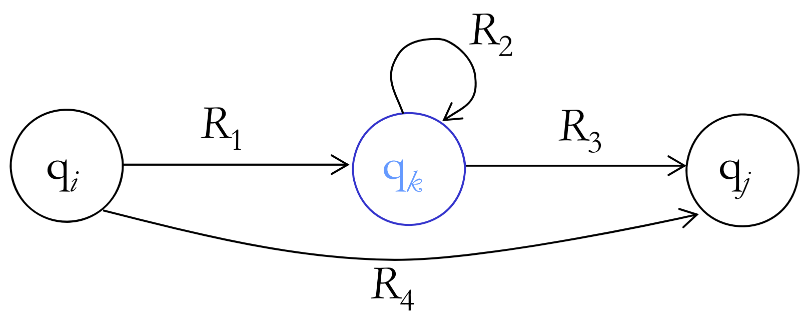
- State Elimination



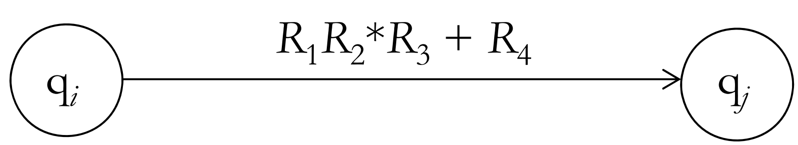
Converting a GNFA to a 2-state GNFA

- State Elimination – General Method
  - To eliminate state q_k, for every pair of states (q_i, q_j)

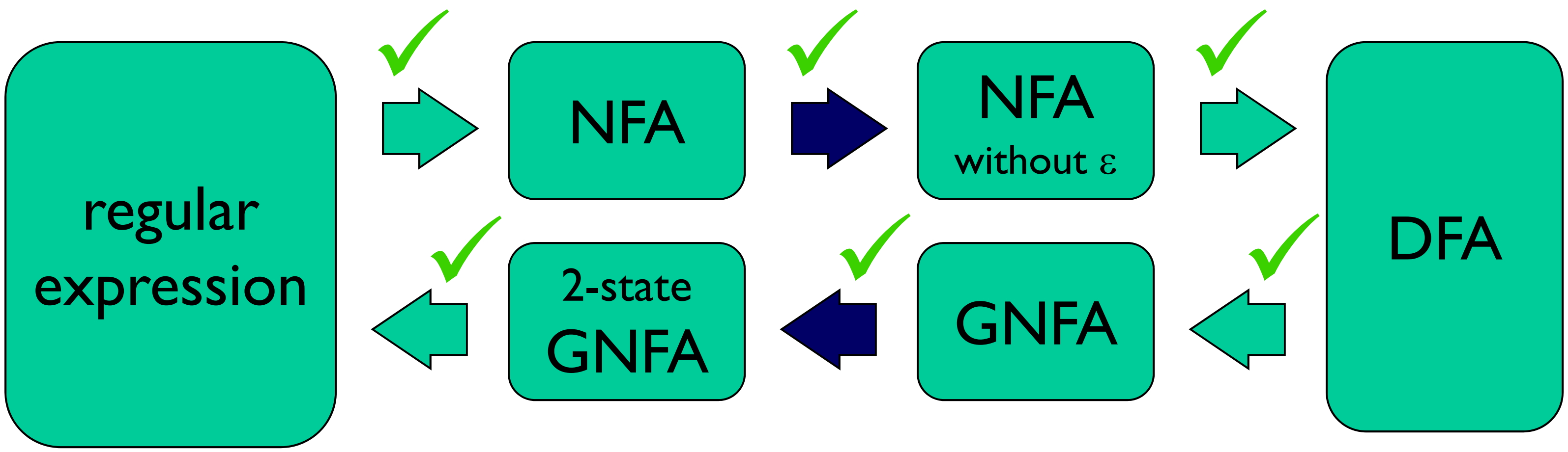
Replace

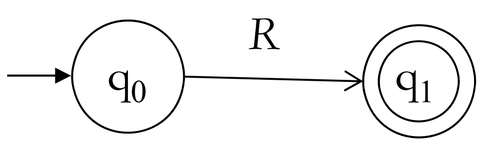


by



Roadmap





A 2-state GNFA is the same as a regular expression R

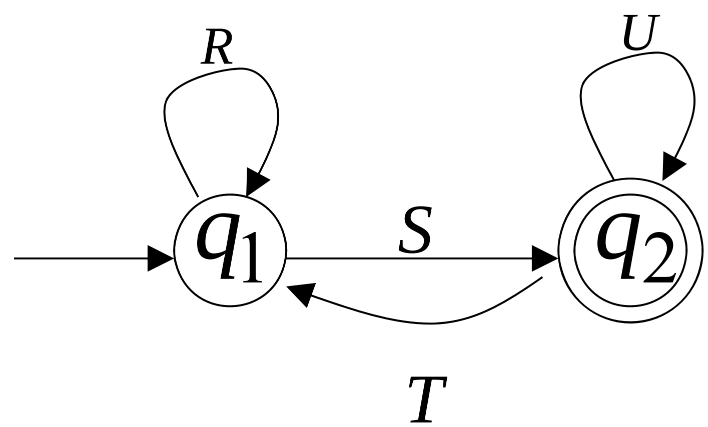
Converting DFAs to Regular Expressions

Converting DFAs to REs

- State Elimination Method
  - 1. Starting with intermediate states and then moving to accepting states, apply the state elimination process to produce an equivalent automaton with regular expression labels on the edges.
    - The result will be one or two state automaton with a Start state and an Accept state.

DFA to RE: State Elimination

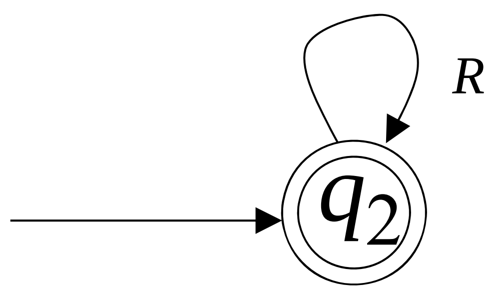
- State Elimination Method
  - 2. If the two sates are different, we get an automaton like the one shown below:



  - The regular expression for this automaton is (R+SU*T)*SU*

DFA to RE: State Elimination

- State Elimination Method
  - 3. If the start state is also an accepting state, then we must also perform a state elimination from the original automaton that gets rid of every state but the Start state. This leads us to:



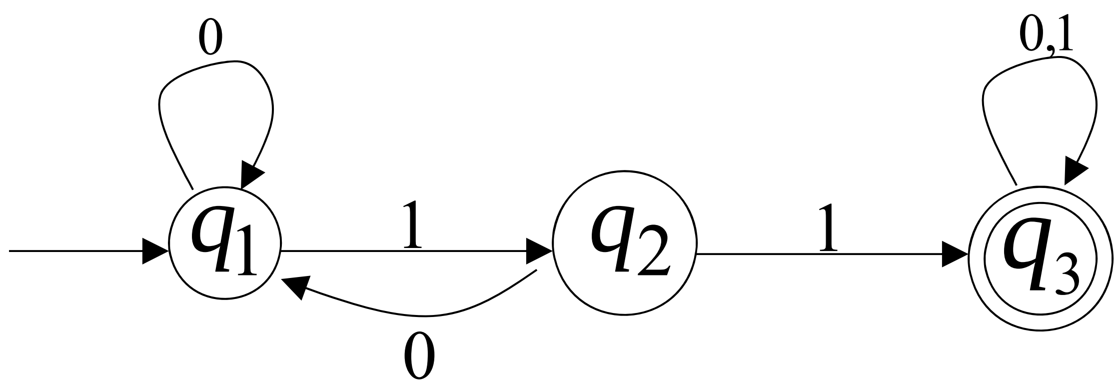
  - We can describe this automaton as a regular expression as R*

DFA to RE: State Elimination

- State Elimination Method
  - 4. If there are n Accept states, the repeat steps 1-3 for each Accept state to get n different regular expressions R_1, R_2, …….R_n.
  - For each repeat we turn any other Accept state to non-Accept state.
  - The final regular expression for the automaton is then the union of each of the n regular expressions.

DFA to RE: Example 1

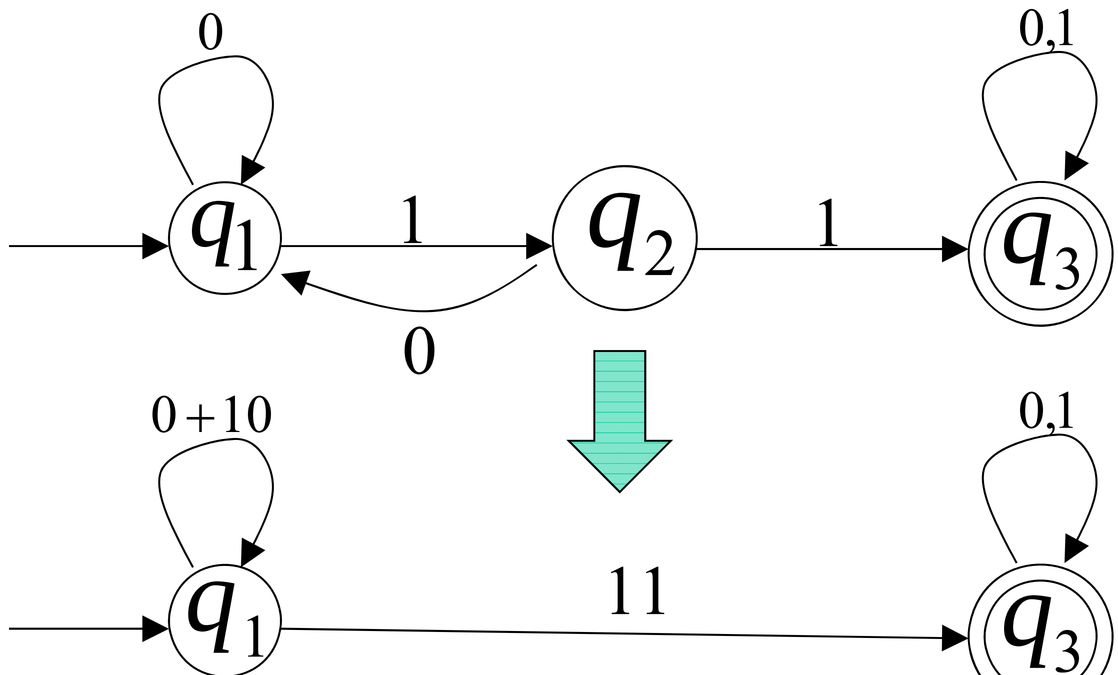
- State Elimination
  - Convert the following to a Regular Expression



DFA to RE: Example 1

State Elimination

- Eliminate State q_2

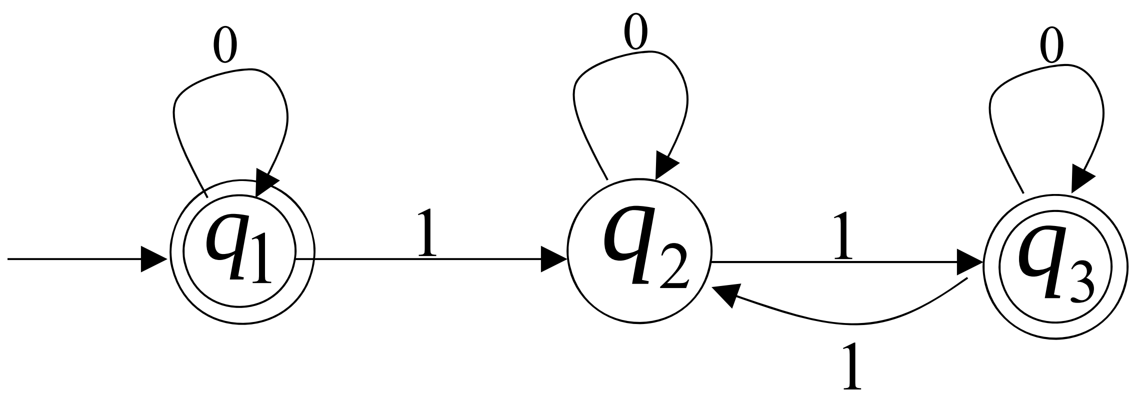


- The regular expression is (0+10)*11(0+1)*

DFA to RE: Example 2

State Elimination

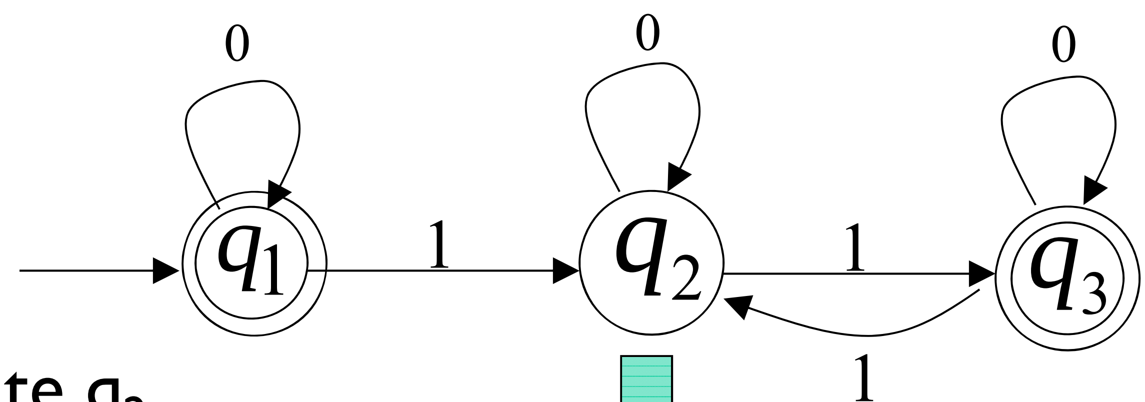
- Automaton that accepts even number of 1's



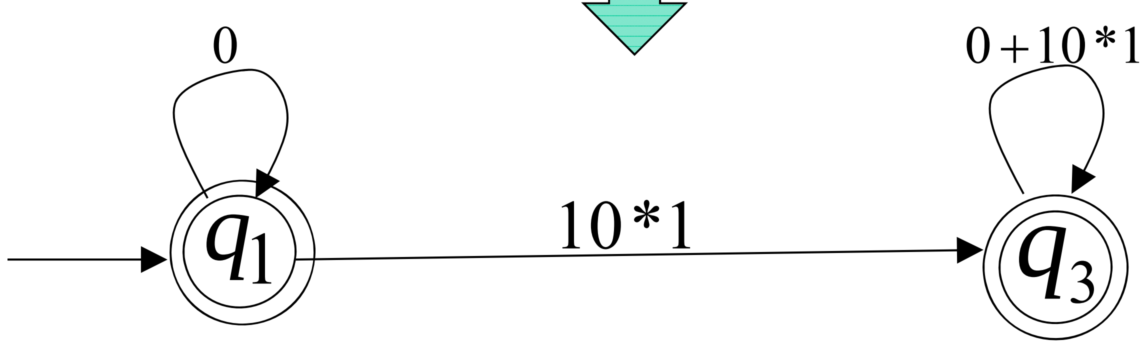
DFA to RE: Example 2

State Elimination

- Automaton that accepts even number of 1's



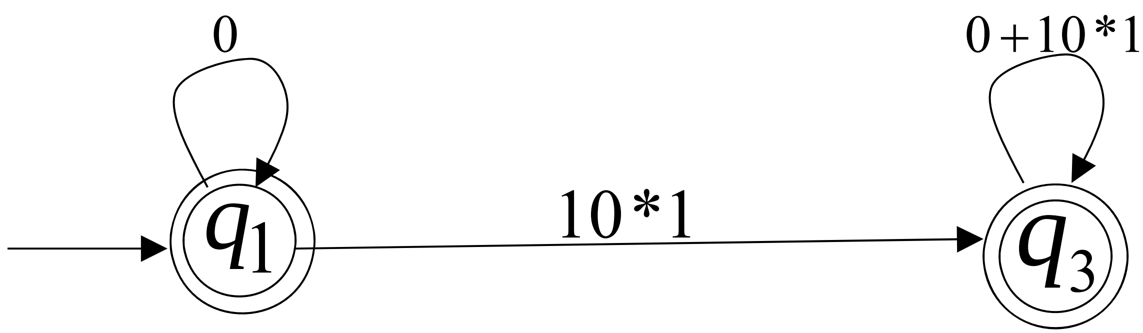
- Eliminate q_2



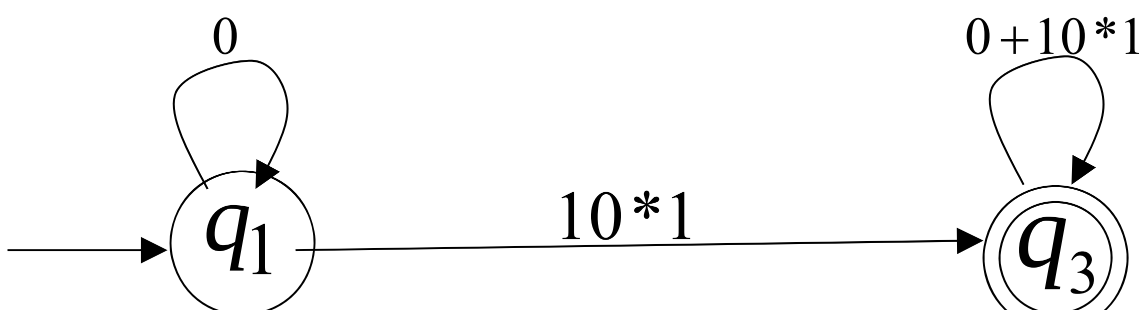
DFA to RE: Example 2

State Elimination

- Automaton that accepts even number of 1's



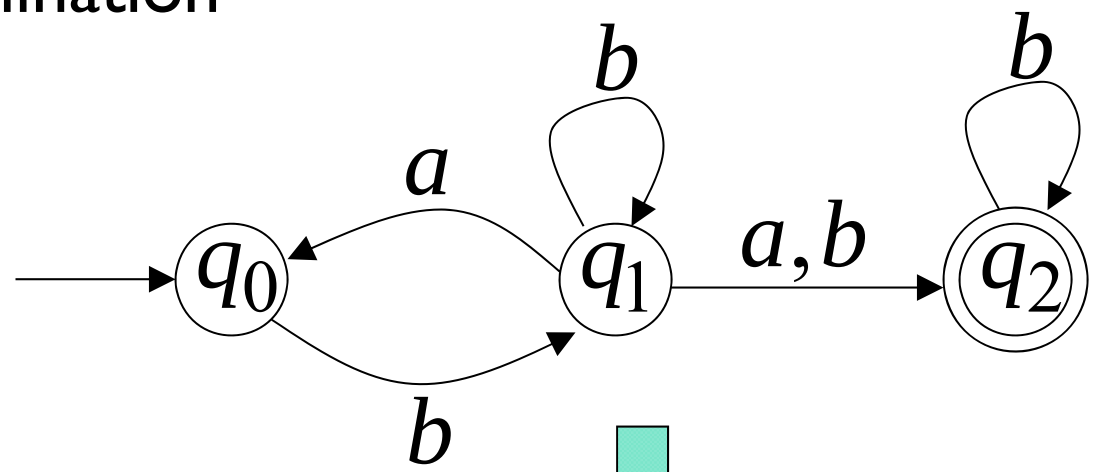
- Turn off state q_1

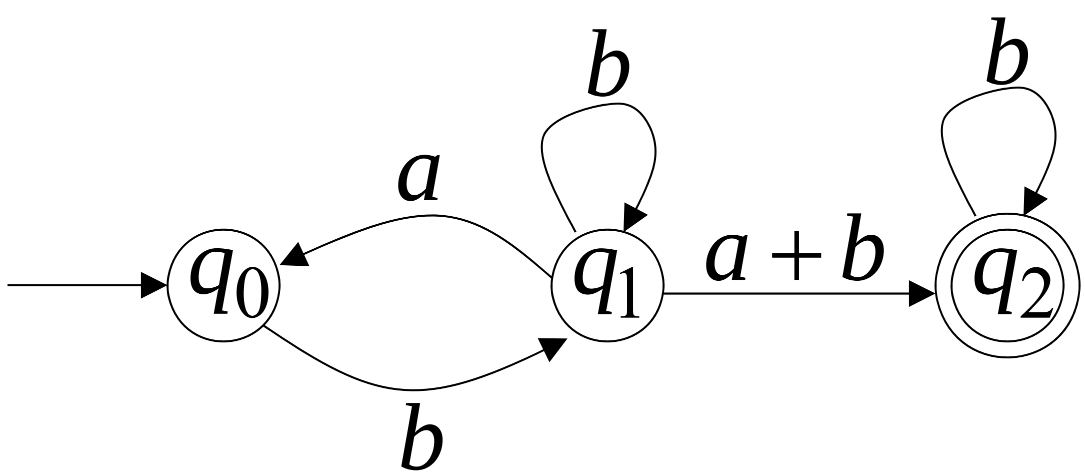


- The regular expression is 0*+0*10*1(0+10*1)*

DFA to RE: Example 3

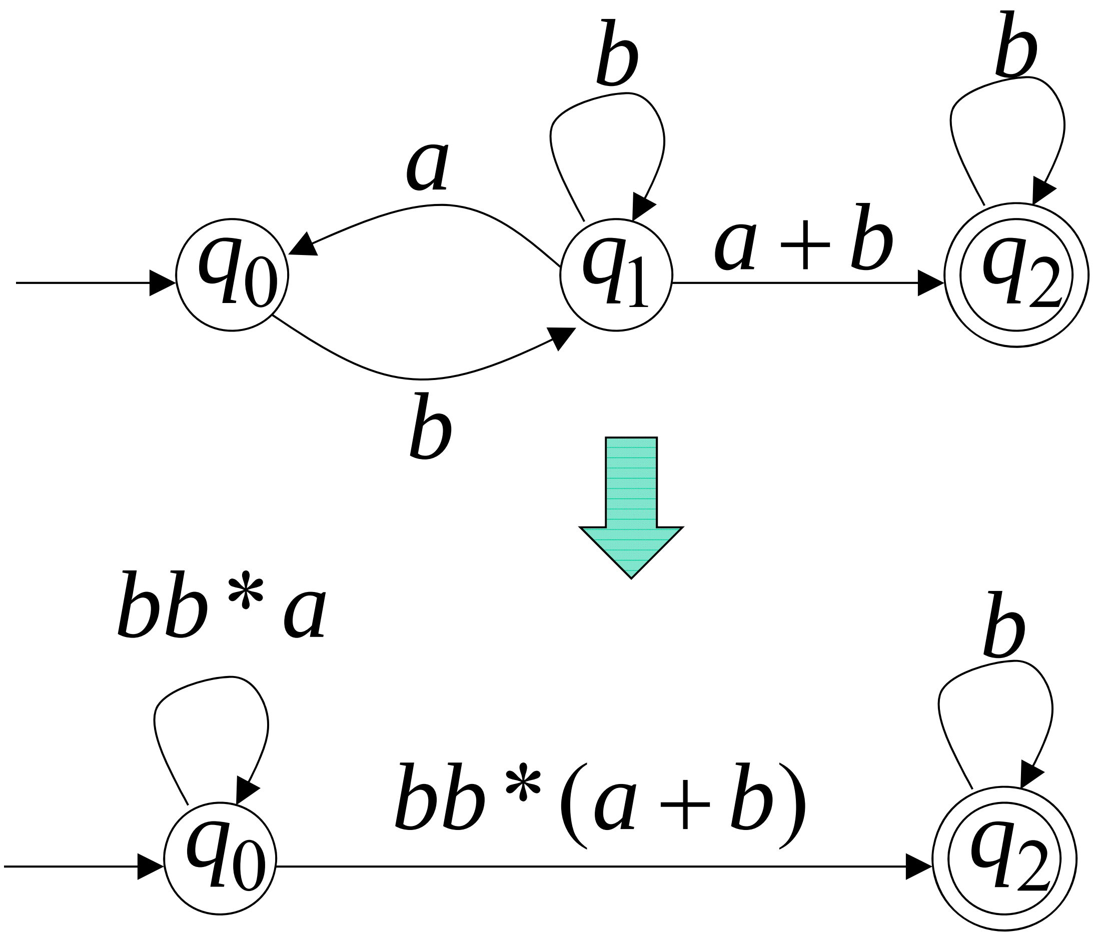
State Elimination





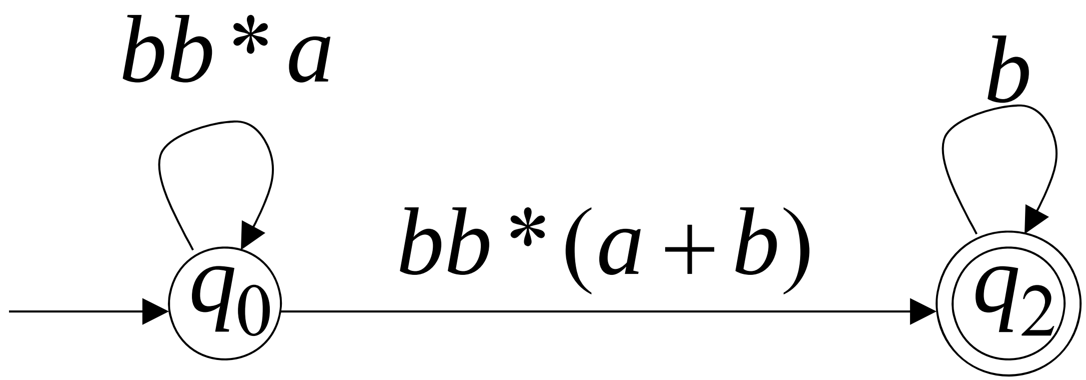
DFA to RE: Example 3

State Elimination



DFA to RE: Example 3

State Elimination



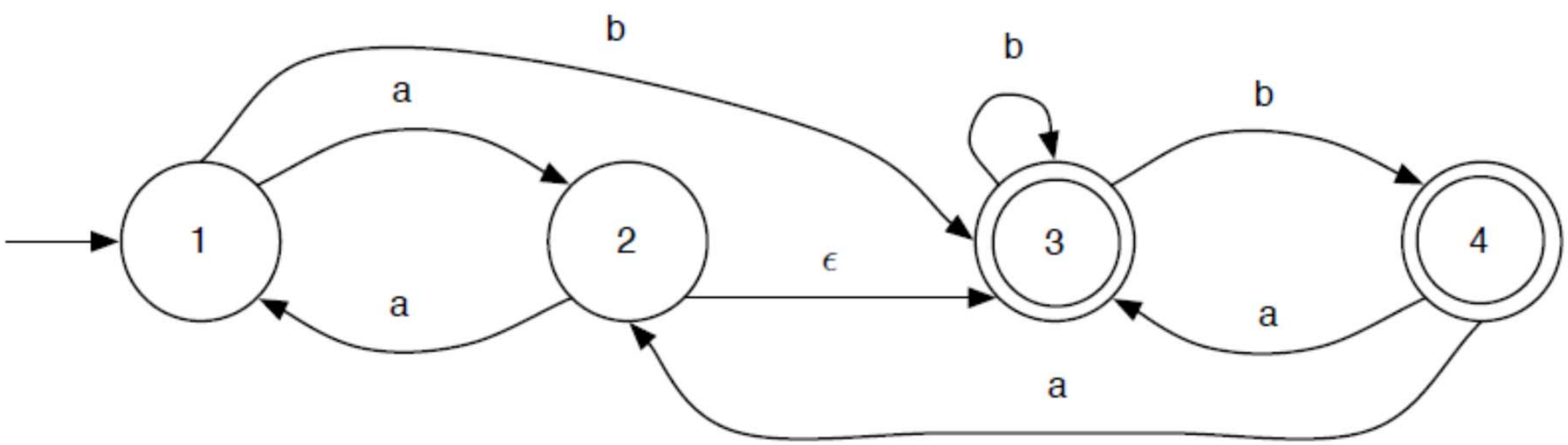
$$r = (bb^*a)^*bb^*(a+b)b^*$$

$$L(r) = L(M) = L$$

DFA to RE: Example 4

State Elimination

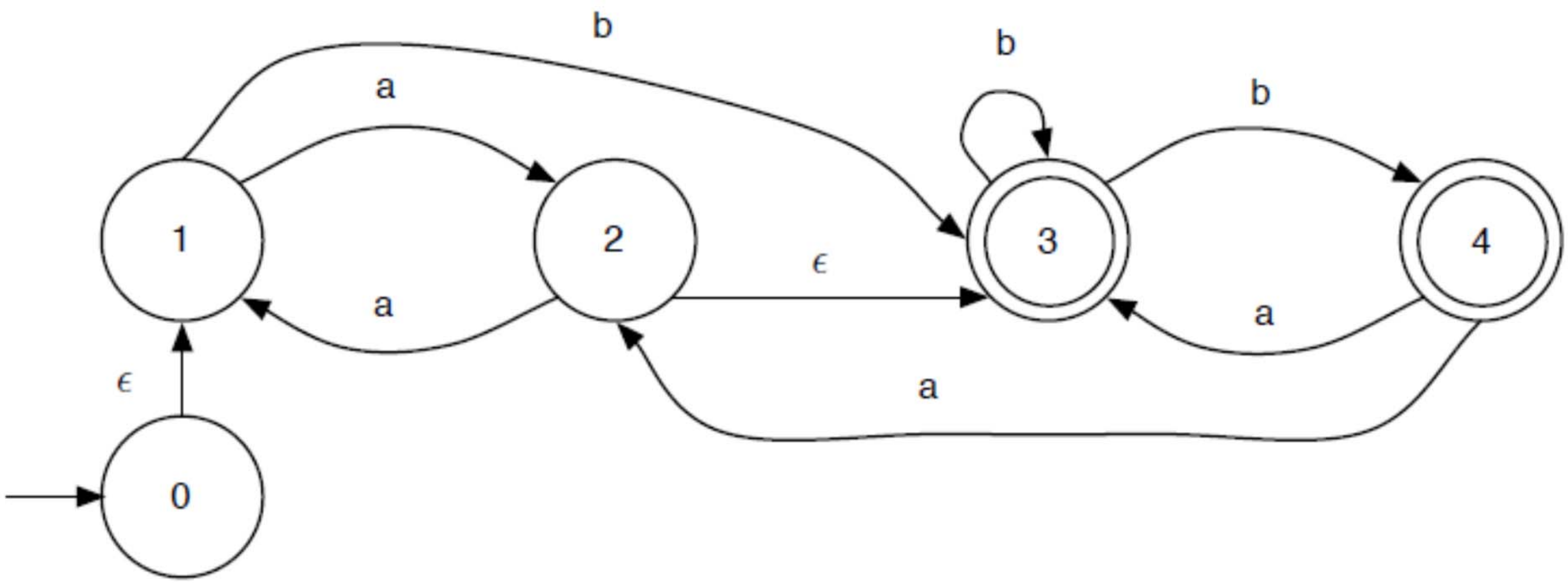
– Convert the following to a Regular Expression



DFA to RE: Example 4

State Elimination – Step1

- If the start state is an accepting state or has transitions in, add a new non-accepting start state and add an ε transition between the new start state and the former start state.



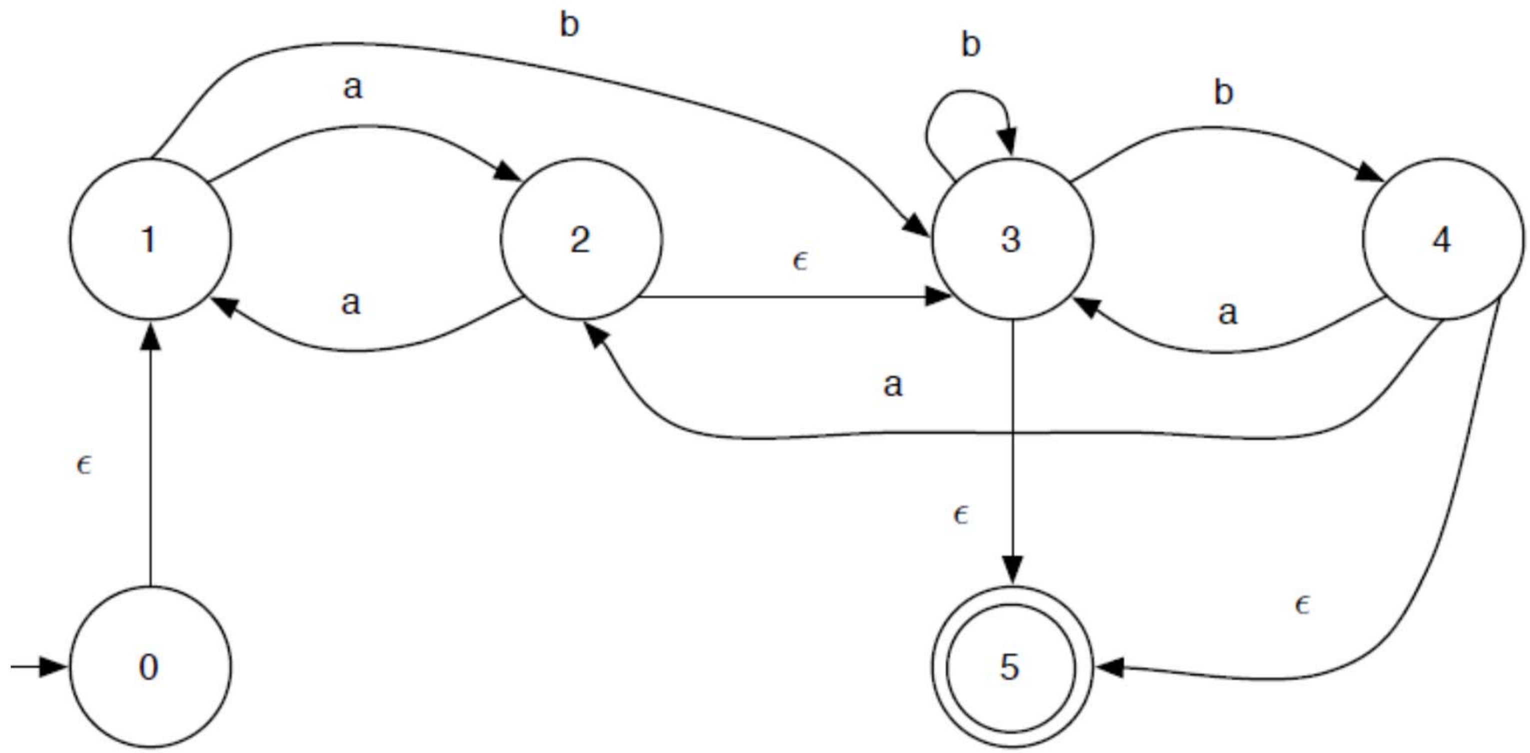
DFA to RE: Example 4

State Elimination – Step2

- If there is more than one accepting state or if the single accepting state has transitions out, add a new accepting state, make all other states non-accepting, and add an ε transition from each former accepting state to the new accepting state.

DFA to RE: Example 4

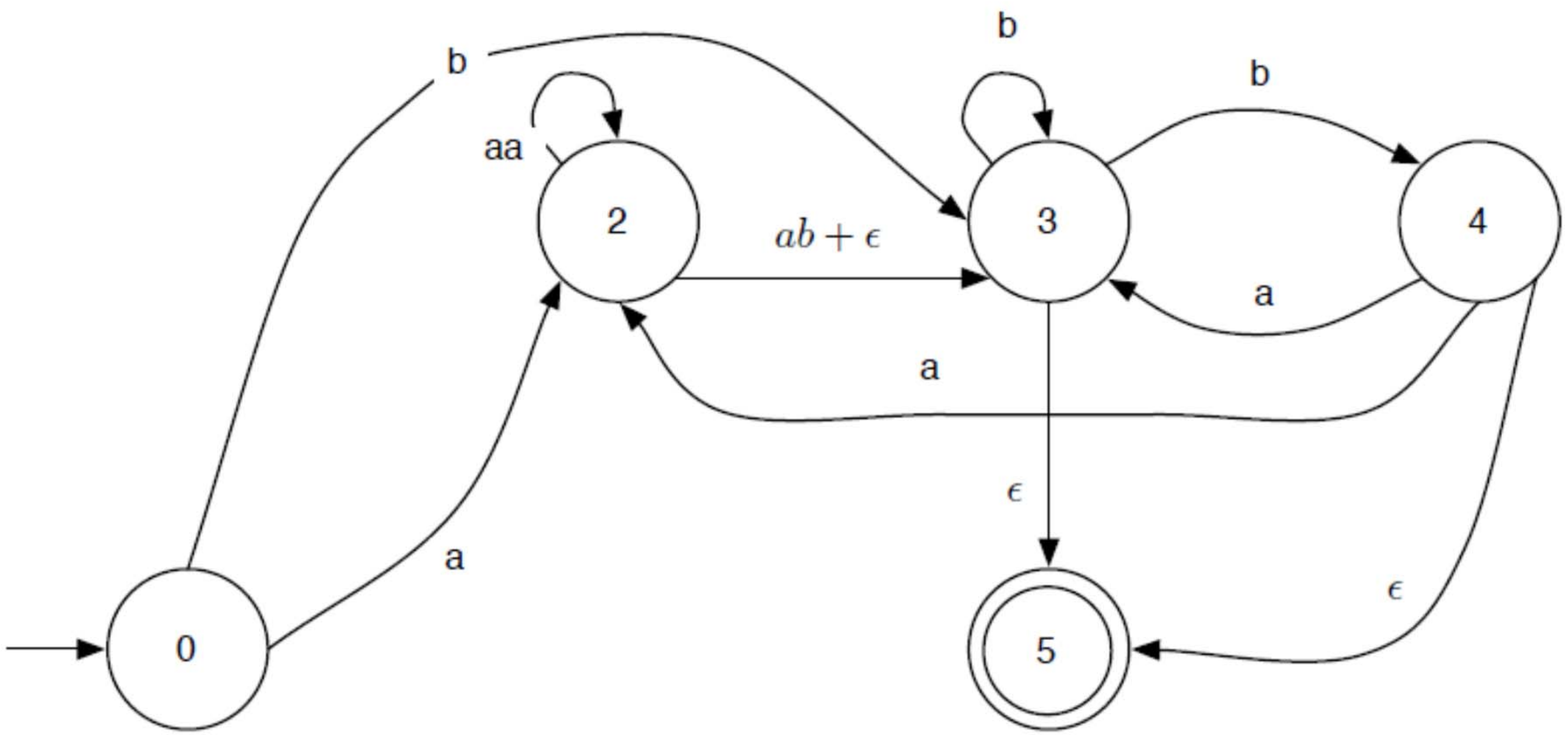
State Elimination – Step2



DFA to RE: Example 4

State Elimination – Step 3

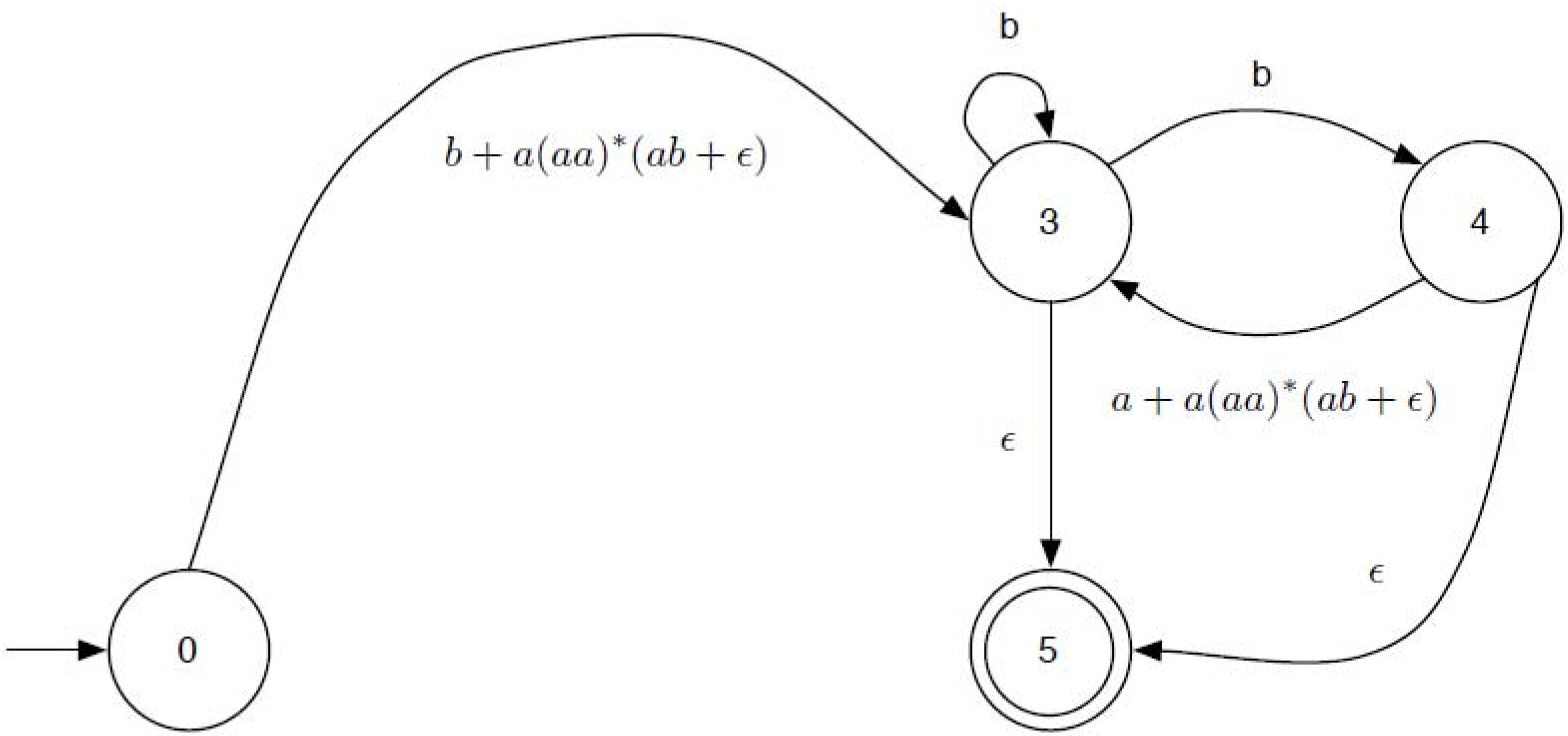
- For each non-start non-accepting state in turn, eliminate the state and update transitions. Eliminating state1, we get



DFA to RE: Example 4

State Elimination – Step 3

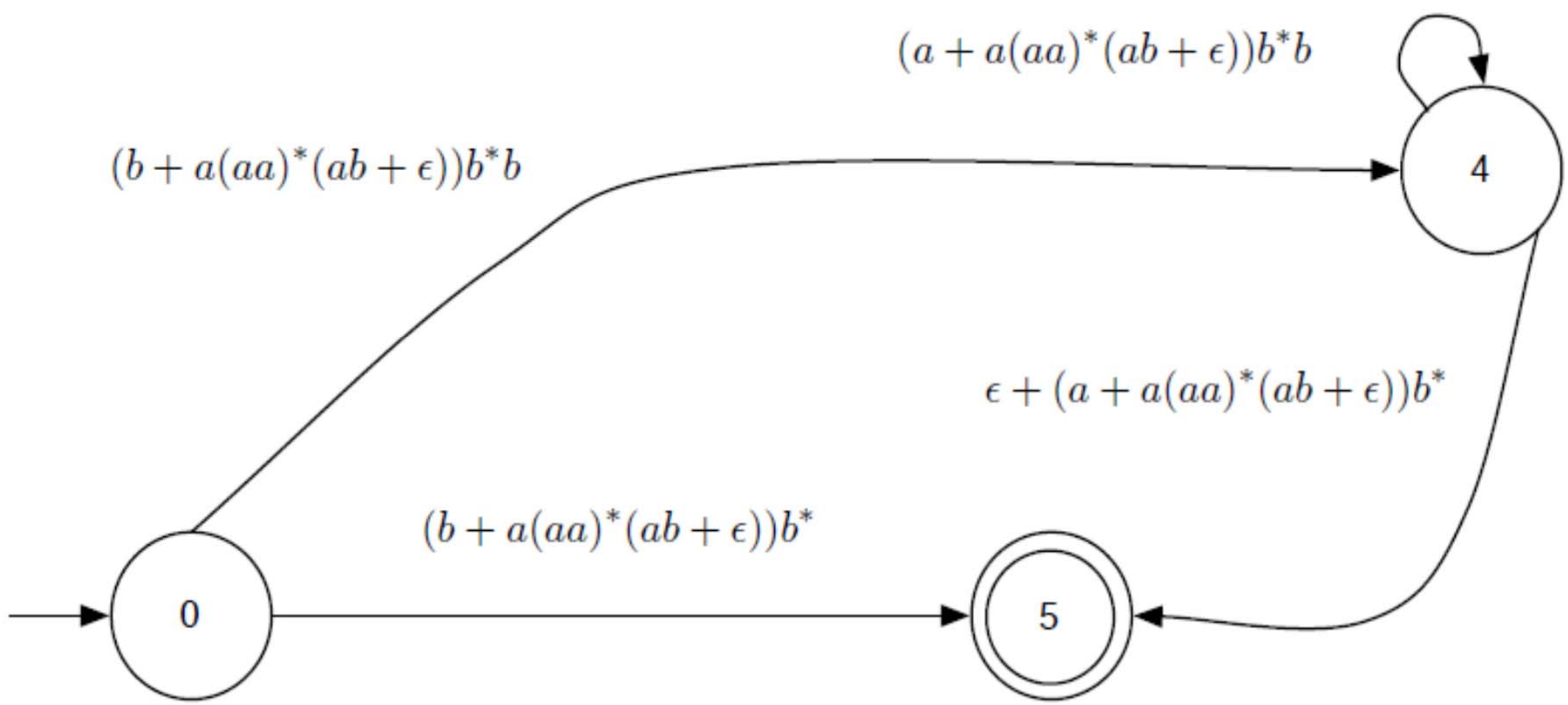
- Eliminating state2, we get



DFA to RE: Example 4

State Elimination – Step 3

- Eliminating state3, we get



DFA to RE: Example 4

State Elimination – Step 3

- Eliminating state4, we get

$$(b + a(aa)^*(ab + \epsilon))b^* +$$

$$((b + a(aa)^*(ab + \epsilon))b^*b)((a + a(aa)^*(ab + \epsilon))b^*b)^*(\epsilon + (a + a(aa)^*(ab + \epsilon))b^*)$$