Unit 4

Regular Expressions

Reading: Sipser, chapter 1

Overview

- Regular expressions (RE)
- Equivalence of RE and RL

Regular Expressions

- Regular languages (RL) are often described by means of algebraic expressions called *regular expressions (RE)*.
- In arithmetic we use the +, * operations to construct expressions: (2+3)*5
- The value of the arithmetic expression is the number 25.
- The value *of a regular expression* is a *regular language*.

Regular operations

- In *RE* we use *regular operations* to construct expressions describing *regular languages*:

$$\boxed{(0+1)^* \circ 0}$$

where :

- $r+s$ means r OR s
- r^* means Kleene star of r
- $r \circ s$ (or rs) means concatenation of r and s

Formal definition

- A set of regular expressions over an alphabet Σ is defined inductively as follows:

Basis:

ε, $\emptyset$, and σ (for all $\sigma \in \Sigma$) are regular expressions.

Induction:

If *r* and *s* are RE then the following expressions are also RE:

- (r)
- r+s
- $r \circ s$
- r*

Examples over Σ={a,b}

ε, a, a+b, b*, (a+b)b, ab*, (ab)* , a*+b*

- To avoid using many parentheses, the operations have the following priority hierarchy:
    1. * - highest (do it first)
    2. $\circ$
    3. \+ - lowest (do it last)
- Example: b + ab* =(b+(a$\circ$(b*)))
- **Common notations**: $r \circ s = rs$, $r^+ = rr^*$ and $\Sigma = (\sigma_1 + \sigma_2 + \cdots)$.

Regular expressions and regular languages

We associate each regular expression *r* with a regular language *L(r)* as follows:

- $L(\emptyset)=\emptyset$,
- $L(\varepsilon)=\{\varepsilon\}$,
- $L(\sigma)=\{\sigma\}$ for each $\sigma\in\Sigma$,
- $L(r+s)=L(r)\cup L(s)$,
- $L(r\circ s)=L(r)\circ L(s)$,
- $L(r^*)=(L(r))^*$.

Examples over Σ={0,1}

In Class: Describe each language as a regular expression.

L_1 = { w | w has a single 1}

L_2 = { w | w has at least one 1}

L_3 = { w | w contains the string 110}

L_4 = { w | |w| mod 2 =0}

L_5 = { w | w starts with 1}

L_6 = { w | w ends with 00}

Examples over Σ={0,1}, cont'

L7 = { w | w starts with 0 and ends with 10}

L8 = { w | w contains the string 010 or the string 101 }

L9 = { w | w starts and ends with the same letter }

L10 = {0,101}

L11 = {w | w does not contain 11 as a substring}

L12 = {w | #1(w) is even}

L13={w | w does not contain 101}

Properties of regular expressions 1

Useful properties of regular expressions:

- $r+s=s+r$
- $r+\varnothing=\varnothing+r=r$
- $r+r=r$
- $r\varnothing=\varnothing r=\varnothing$
- $rr^*=r^+$
- $r\varepsilon=\varepsilon r=r$

Properties of regular expressions 2

- $r(s+t)=rs+rt$
- $r+(s+t)=(r+s)+t$
- $r(st)=(rs)t$
- $r^*=(r^*)^*=r^*r^*=r^*+r$
- $r^*+r^+=r^*$

Equivalence of regular expressions

- To prove that two regular expressions *r* and *s* are equivalent we need to show that

  $L[r] \subseteq L[s]$ and $L[s] \subseteq L[r]$.

- To show that two regular expressions *are not* equivalent we have to find a word that belongs to one expression and does not belong to the other.

Example

Let $r=\varepsilon+(0+1)^*1$

Let $s=(0^*1)^*$

Are *r* and *s* equivalent?

Answer: Yes. We will prove

$L[s] \subseteq L[r]$ and $L[r] \subseteq L[s]$

$L[r] \supseteq L[s]$

$r = \varepsilon + (0+1)^*1$ $\qquad\qquad$ $s = (0^*1)^*$

- Let $w \in L[s] = (0^*1)^*$.
- $w = \varepsilon$ or $w = x_1 x_2 .. x_n$, $n > 0$ such that $x_i \in L[0^*1]$
- If $w = \varepsilon$ then $w \in L[r]$.
- If $w = x_1 x_2 .. x_n$ then we can represent $w = w'1 = x_1 x_2 .. x_{n-1} 0^k 1$ with $k \geq 0$.

However, $w' = x_1 x_2 .. x_{n-1} 0^k \in L[(0+1)^*]$,

implying $w'1 \in L[(0+1)^*]1 \subseteq L[r]$.

$L[r] \subseteq L[s]$

$r=\varepsilon+(0+1)^*1$ $s=(0^*1)^*$

- Let $w \in L[r]=\varepsilon+(0+1)^*1$.
- If $w=\varepsilon$ then $w \in L[s]$ (by definition of *).
- If $w \neq \varepsilon$ then w can be represented as w=w'1 where $w' \in L[(0+1)^*]$. Assume that w' contains k instances of the letter 1. This means that w' can be written as $w'= x_1 1 x_2 1 .. x_k 1 x_{k+1}$ where $x_i \in 0^*$

But then $w=w'1=$
$=(x_1 1)(x_2 1)\ldots(x_{k+1} 1)=(0^*1)(0^*1)\ldots(0^*1)$

So $w \in L[(0^*1)^*]$.

Another example

Are *r* and *s* equivalent?

$r=(0+1)^*1+0^*$

$s=(1+0)(0^*1)^*$

Answer: No.

- Consider the word $w = \varepsilon$.
- $w \in L[r]=(0+1)^*1+0^*$, because $w \in 0^*$.
- But $w \notin L[s] =(1+0)(0^*1)^*$, as all words in L[s] have at least one letter.

Equivalence of RE with FA

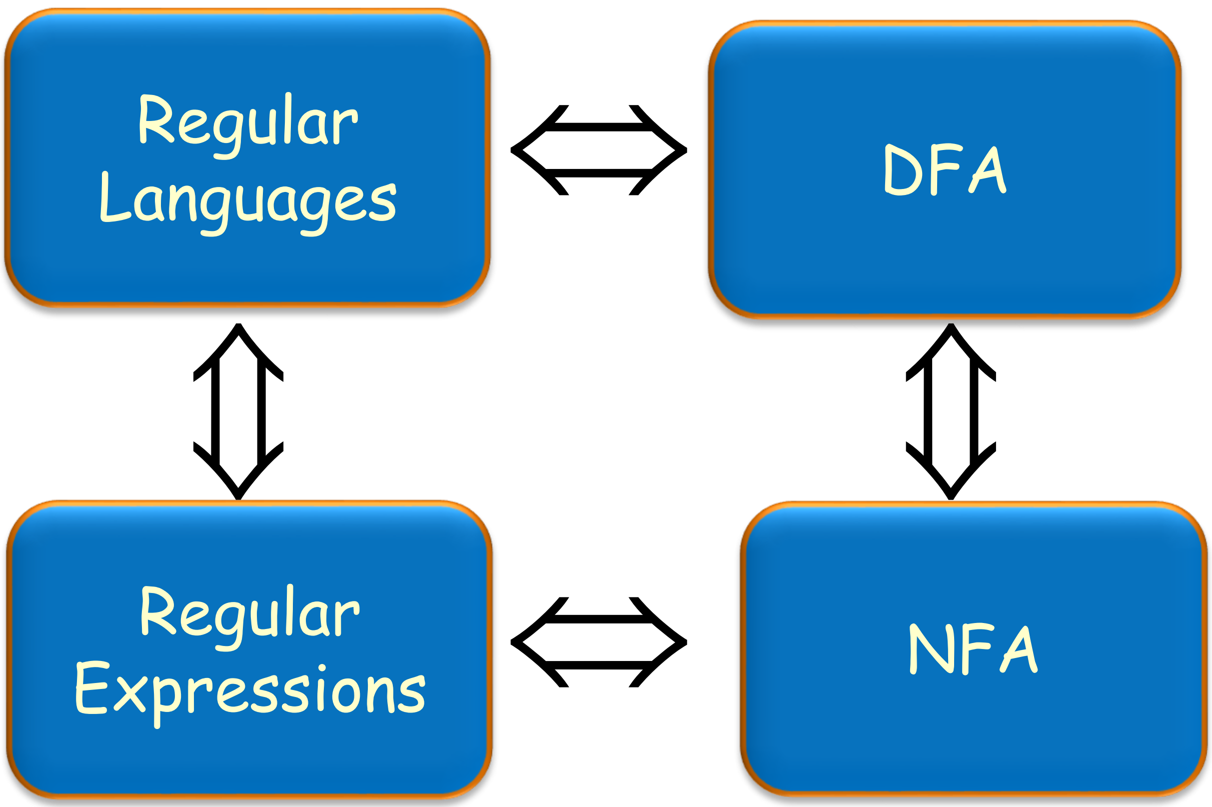
- Regular expressions and finite automata are equivalent in terms of the languages they describe.

Theorem:

A language is regular iff some regular expression describes it.

This theorem has two directions. We prove each direction separately.

Equivalences so far…



Converting RE into FA

Claim: If a language is described by a regular expression, then it is regular.

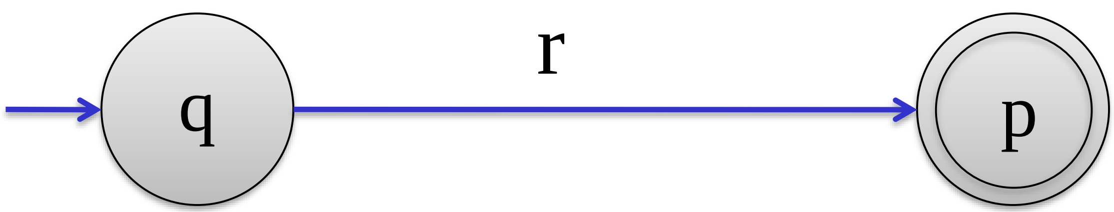
Proof idea:

Given a RE *r*, build an NFA by transforming *r* into a non-deterministic finite automaton *A* s.t. *L(A)*=*L(r)*.

Converting RE into RL

RE to FA Algorithm

- Given *r* we start the algorithm with a NFA *A* having a start state, a single accepting state and an edge labeled *r* :

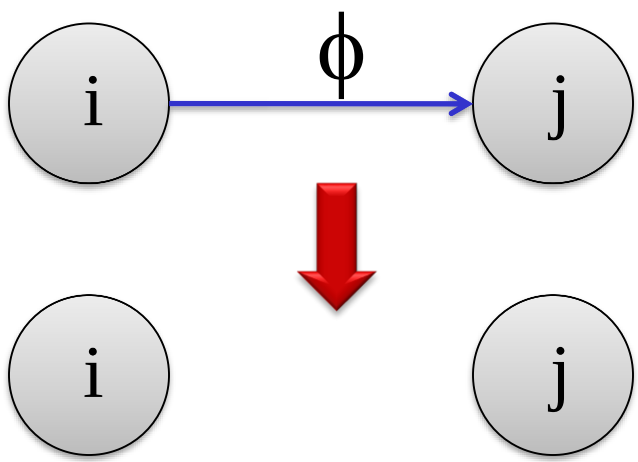


Note: We assume at a moment that edges can be labeled with RE, not just letters.

RE to FA Algorithm (cont.)

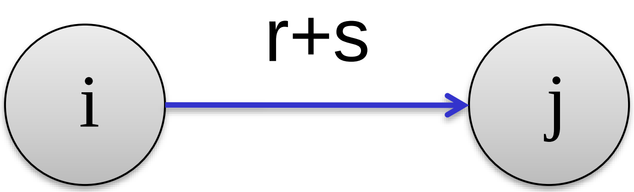
We transform this machine into an NFA *A* by applying the following rules (in order) until all the edges are labeled with either a letter $\sigma \in \Sigma$ or ε:

1. If an edge is labeled $\varnothing$, then delete this edge.

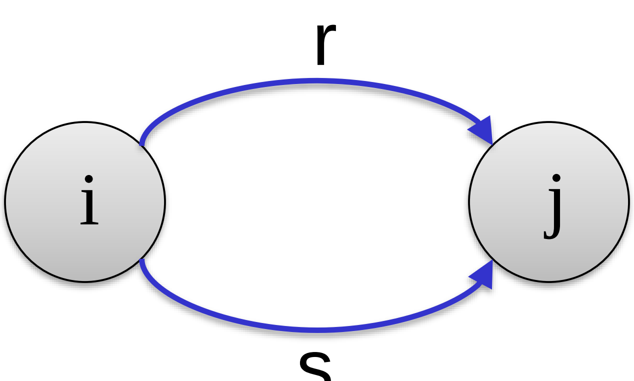


RE to FA Algorithm (cont.)

2. Transform any diagram of the type

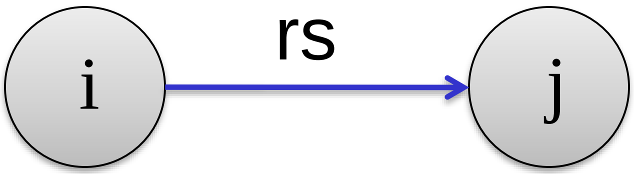


into the diagram

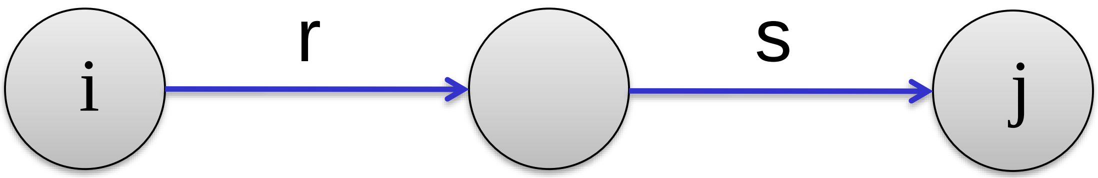


RE to FA Algorithm (cont.)

3.Transform any diagram of the type

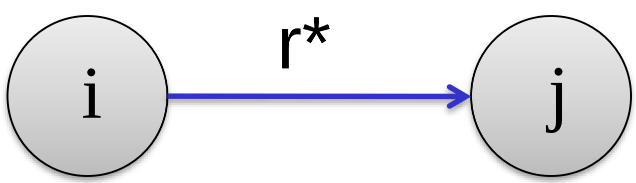


into the diagram

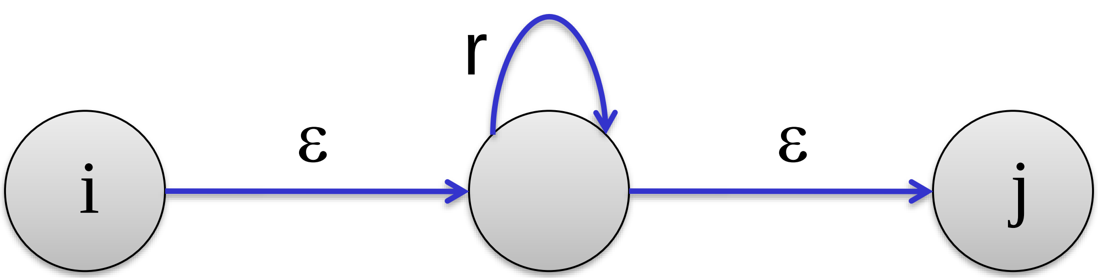


RE to FA Algorithm (cont.)

4. Transform any diagram of the type



into the diagram

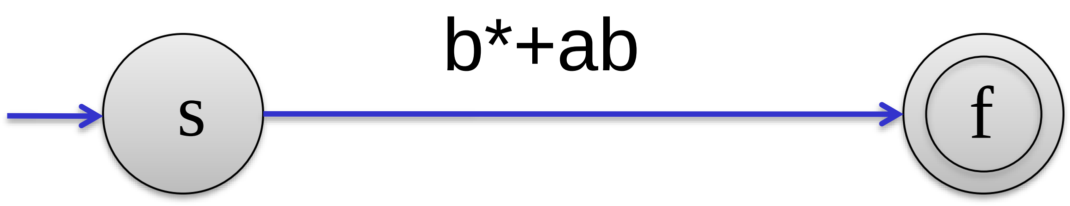


Example:

Construct a NFA for the regular expression

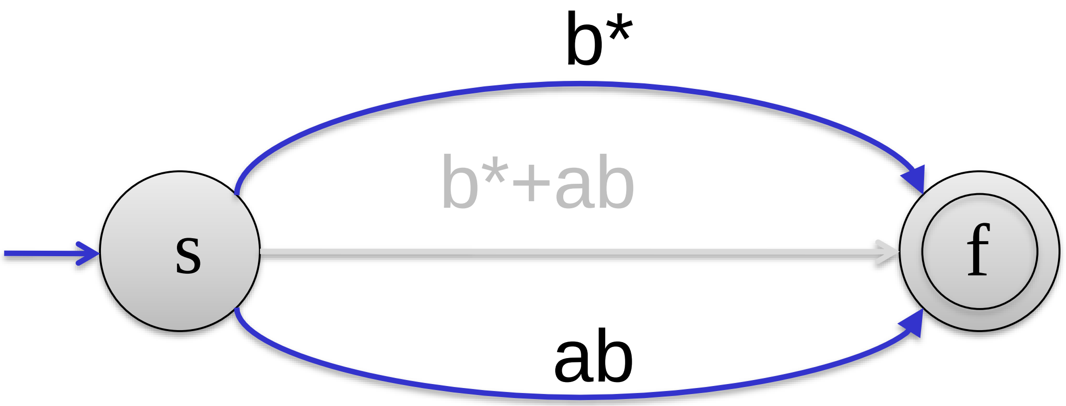
b*+ ab.

Solution: We start by drawing the diagram:



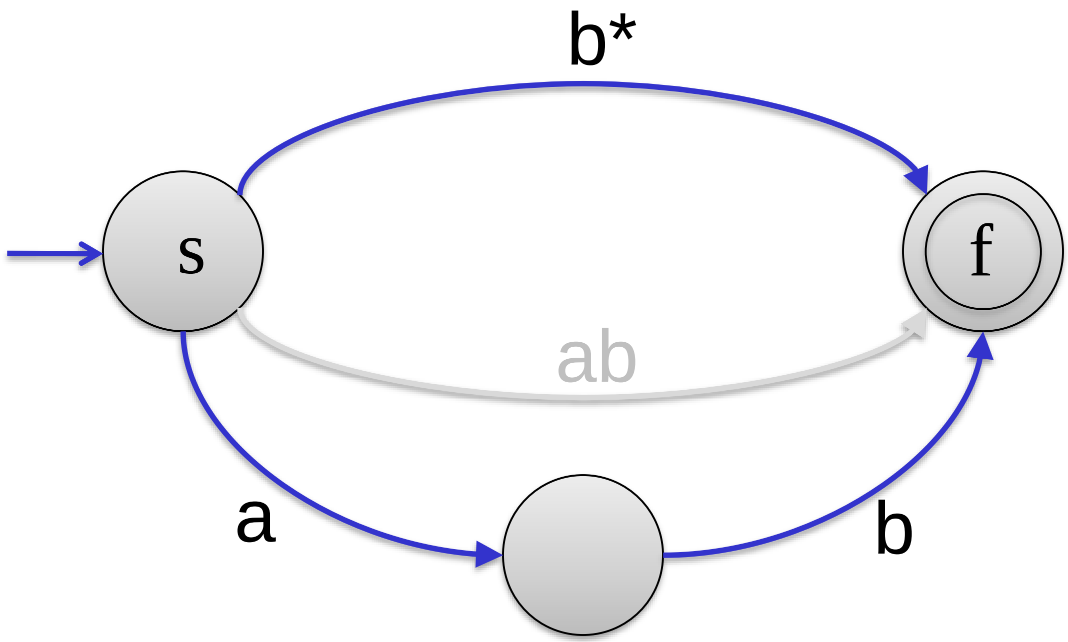
Example (cont.)

Next we apply rule 2 for b*+ab:



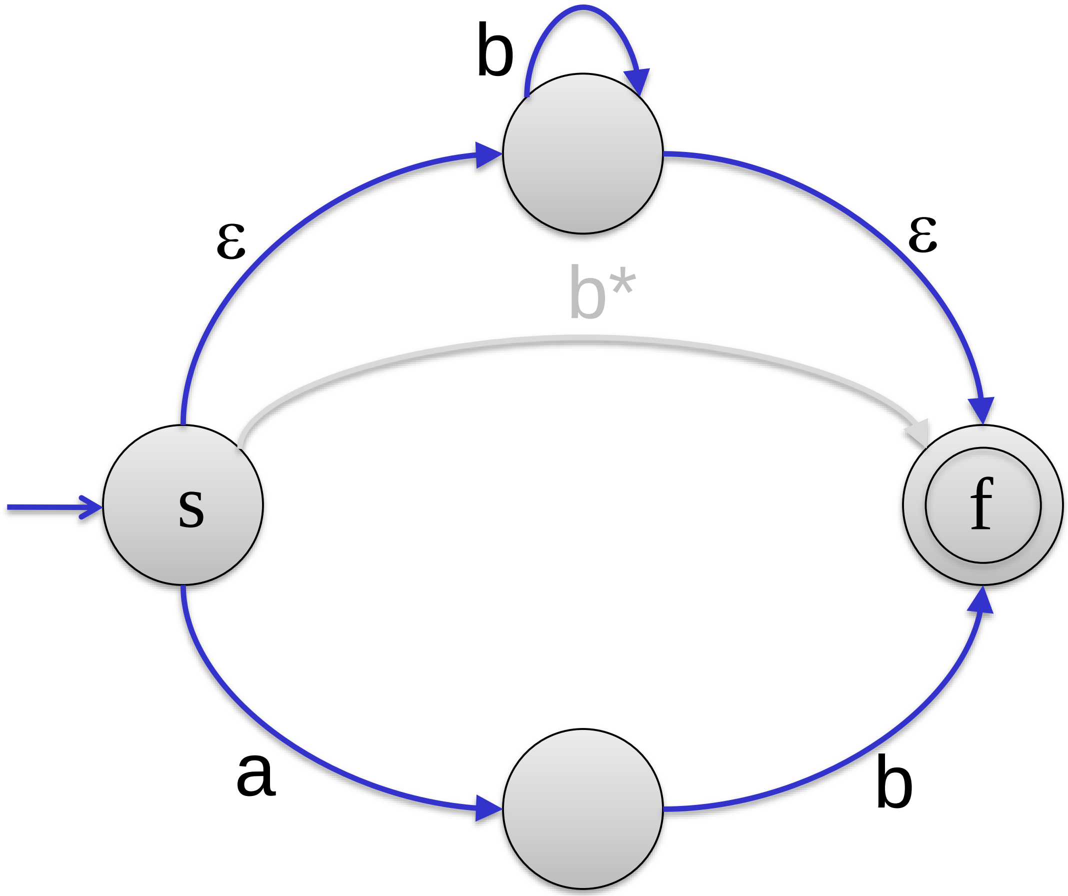
Example (cont.)

Next we apply rule 3 for ab:

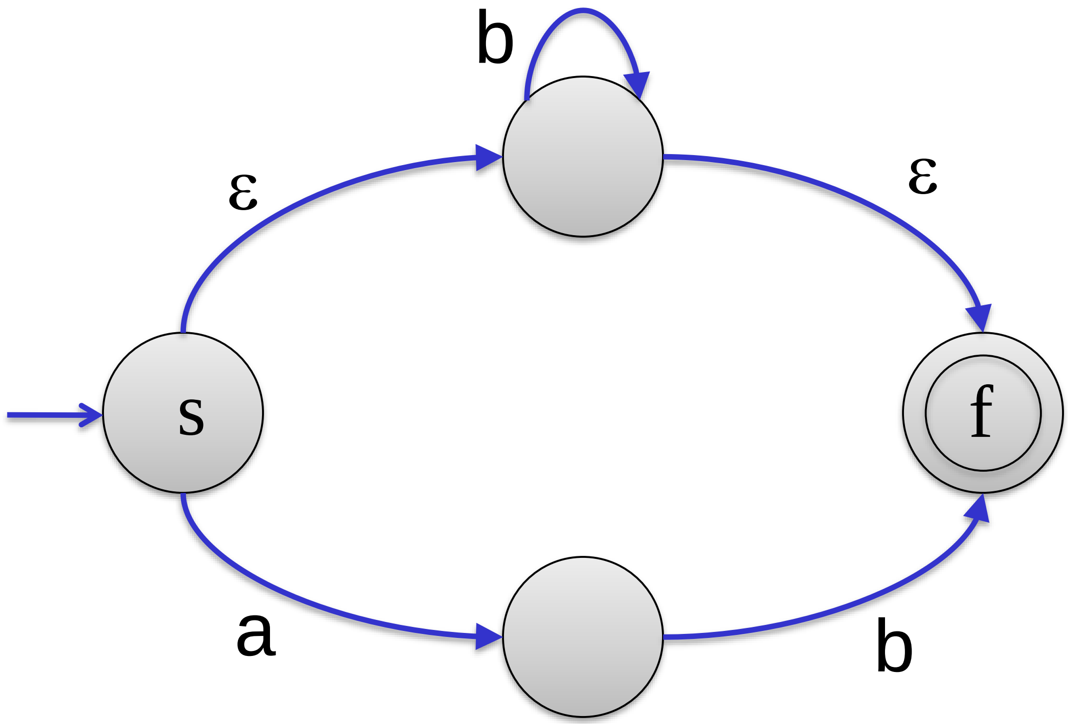


Example (cont.)

Next we apply rule 4 for b*:



The final NFA



Example 2: Draw an NFA for (ab+a)*

Example 3: Draw an NFA for (0+1)*101 (0+1)*

Converting FA into RE

Claim: If a language is regular, then it can be described by a regular expression.

Proof idea:

Transform some DFA M into a regular expression r s.t. $L(r)=L(M)$.

Converting RL into RE

Converting FA into RE

- The algorithm will perform a sequence of transformations that deform the DFA into new machines with edges labeled with regular expressions
- It stops when the machine has:
  1. two states: Start, Finish
  2. one edge with a regular expression on it.

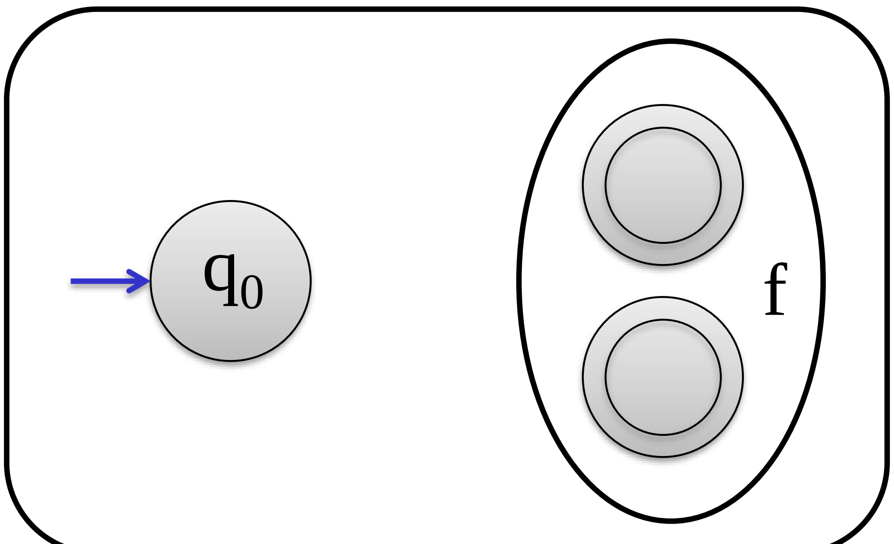
Generalized NFA

- Before we start we first convert the DFA into a *Generalized NFA* (GNFA):
  - GNFA might have RE as labels.
  - GNFA has a *single* accept state.
  - The start state has only arrows going out.
  - The accept state has only arrows coming in.
  - Except for the start and accept state, one arrow goes from every state to every other state (including itself), and in each direction.

Converting DFA into GNFA

The input is a DFA $N=(Q, \Sigma, \delta, q_0, F)$.
Perform the following steps:

1. Create a new start state *s* and draw a new edge labeled with ε from s to the q_0. Add $\delta(s,\varepsilon)= q_0$.

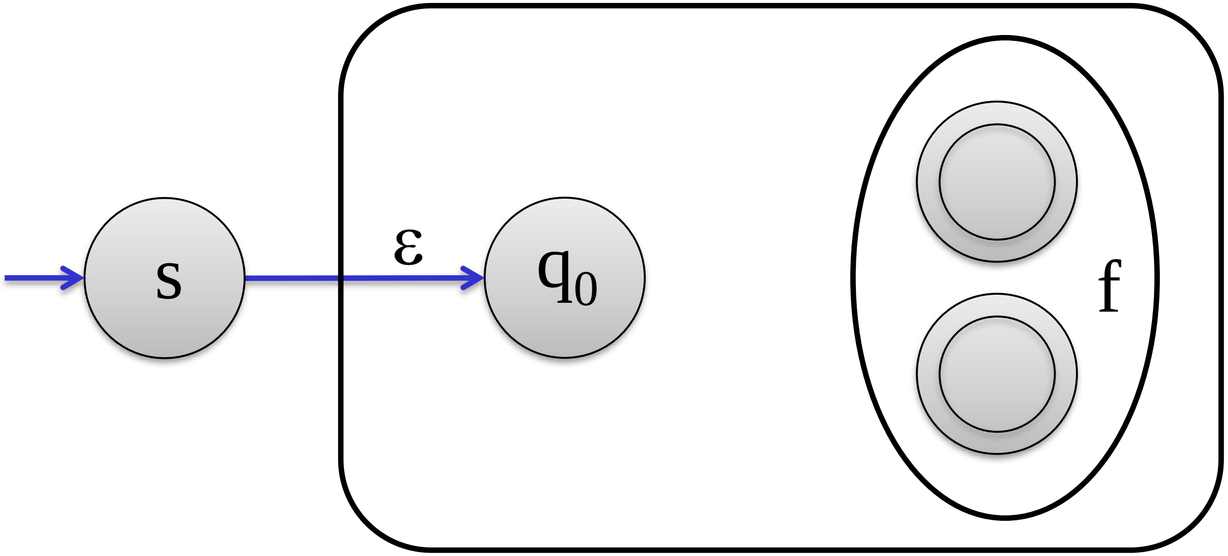


Converting DFA into GNFA

The input is a DFA $N=(Q, \Sigma, \delta, q_0, F)$.

Perform the following steps:

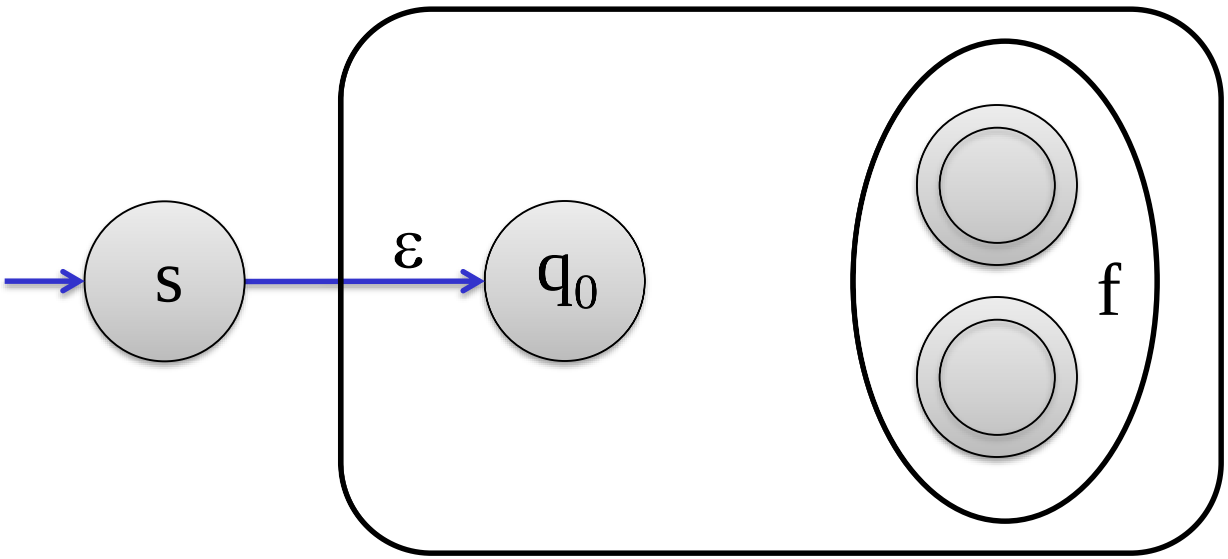
1. Create a new start state *s* and draw a new edge labeled with ε from s to the q_0. Add $\delta(s,\varepsilon)= q_0$.



Converting DFA into GNFA (cont.)

2. Create a new single accepting state *f* and draw new edges labeled ε from all the original accepting states to *f*.

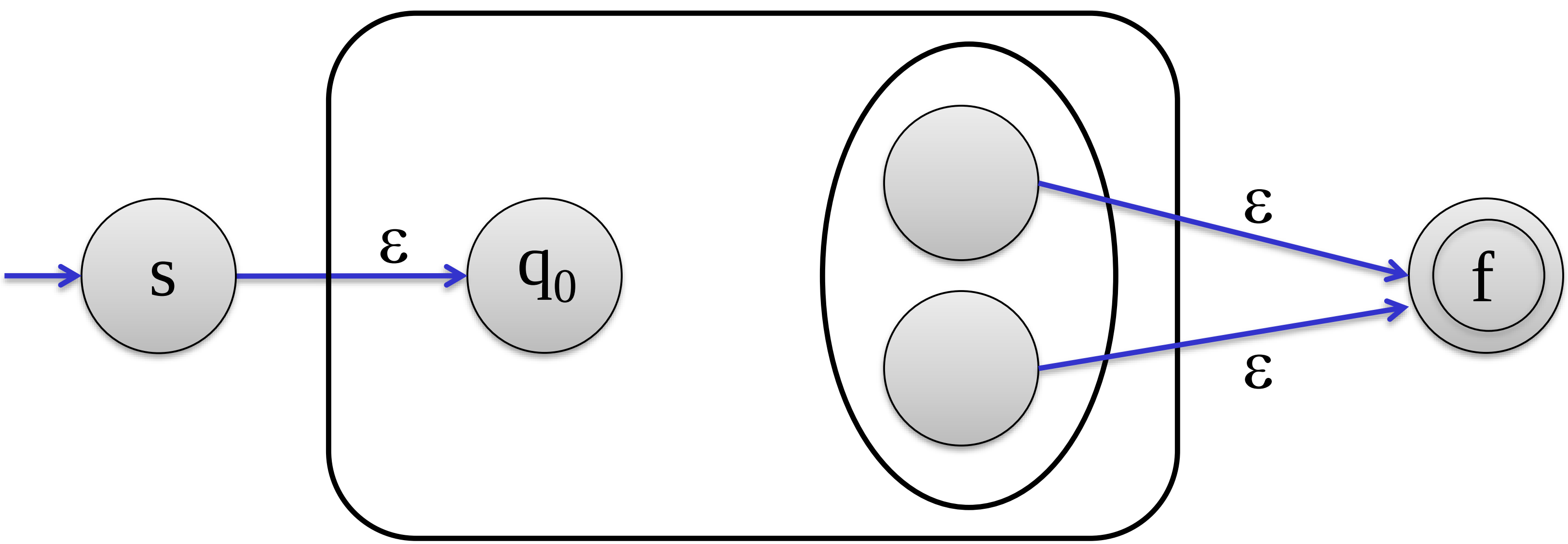
Formally: For each $q \in F$, add $\delta(q,\varepsilon) = f$ and update $F = \{f\}$.



Converting DFA into GNFA (cont.)

2. Create a new single accepting state *f* and draw new edges labeled ε from all the original accepting states to *f*.

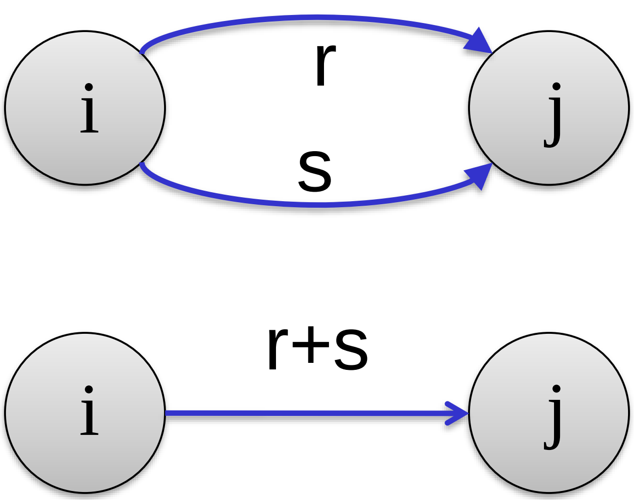
Formally: For each $q \in F$, add $\delta(q,\varepsilon) = f$ and update $F = \{f\}$.

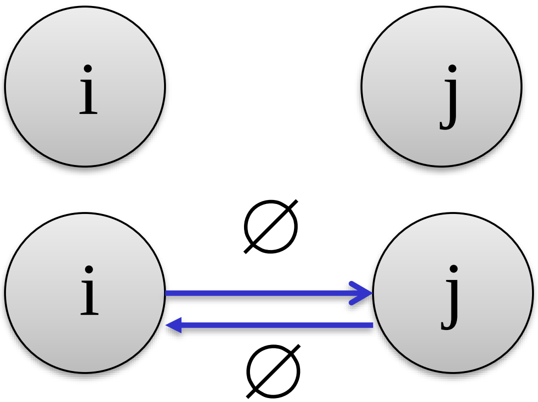


Converting DFA into GNFA (cont.)

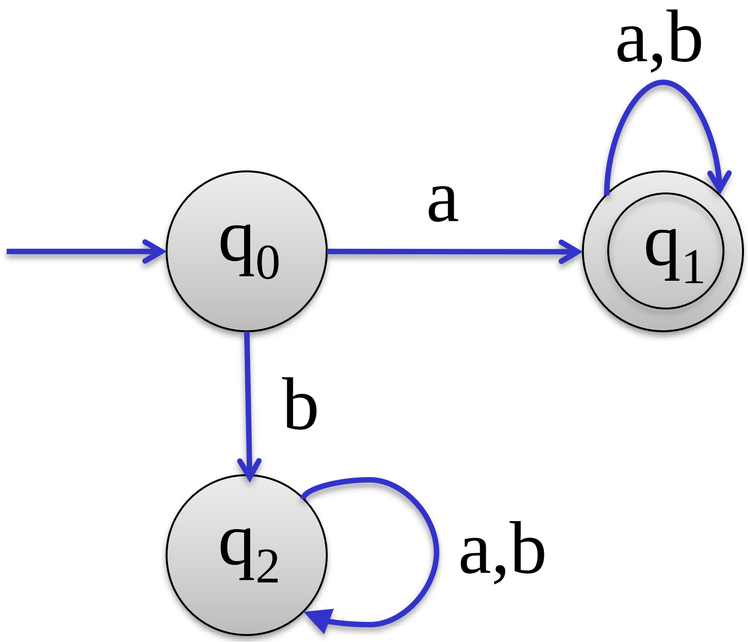
3. For each pair of states *i* and *j* that have more than one edge between them (in the same direction), replace all the edges by a *single* edge labeled with the RE formed by the sum of the labels of these edges.
4. If there is no edge <i,j> then label(i,j)=∅

(including edges <j,j>)

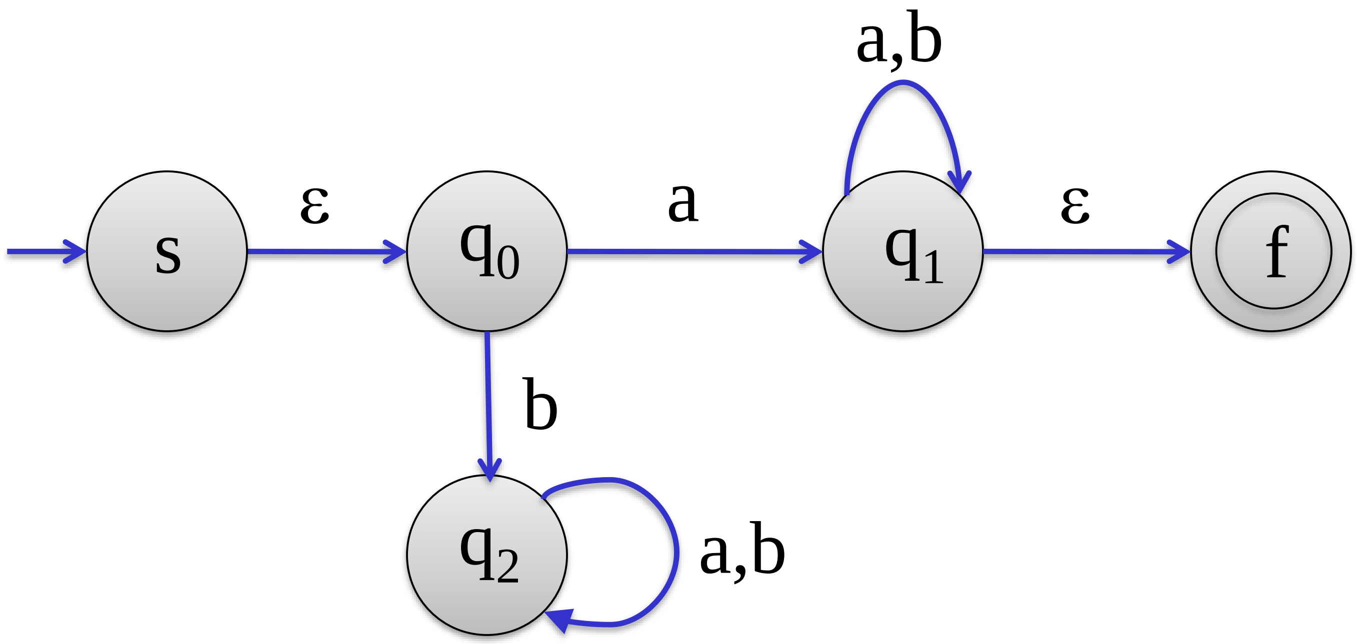




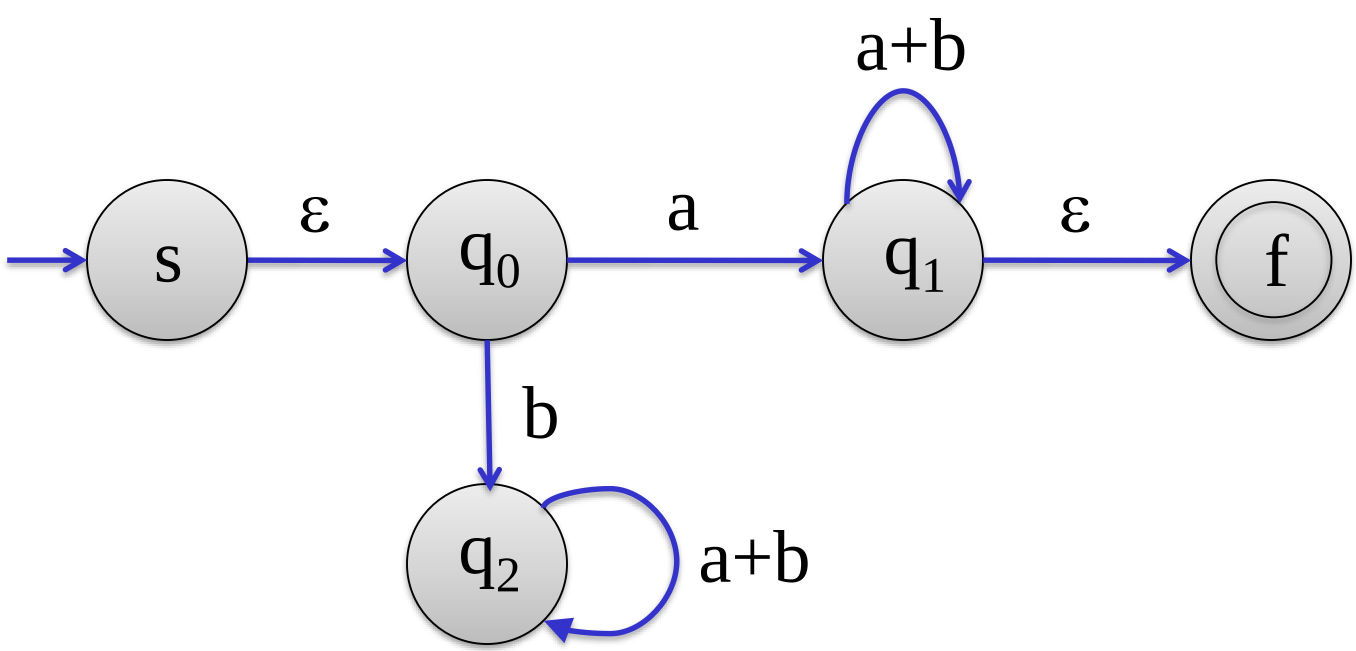
Example: DFA to GNFA



Example: DFA to GNFA (cont.)



Example: DFA to GNFA (cont.)



Converting GNFA into RE

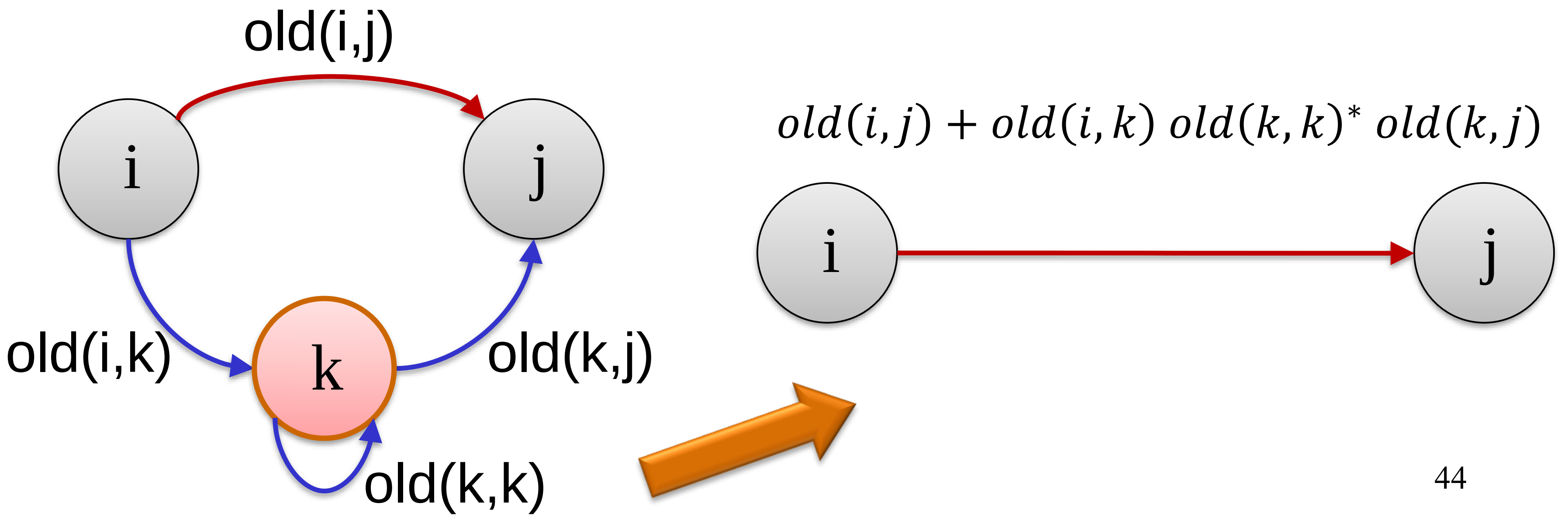
- Let old(i,j) denote the label on edge <i,j> of the current GNFA.
- Construct a sequence of new GNFAs by *eliminating one state at a time* (≠ s,f) until the only two states remaining are *s* and *f*.
- When a state is eliminated, a new (equivalent) machine is constructed.
- The state elimination order is arbitrary.
- The label on <s,f> in the final machine is the required *RE.*

Converting GNFA into RE (cont.)

Eliminating state k

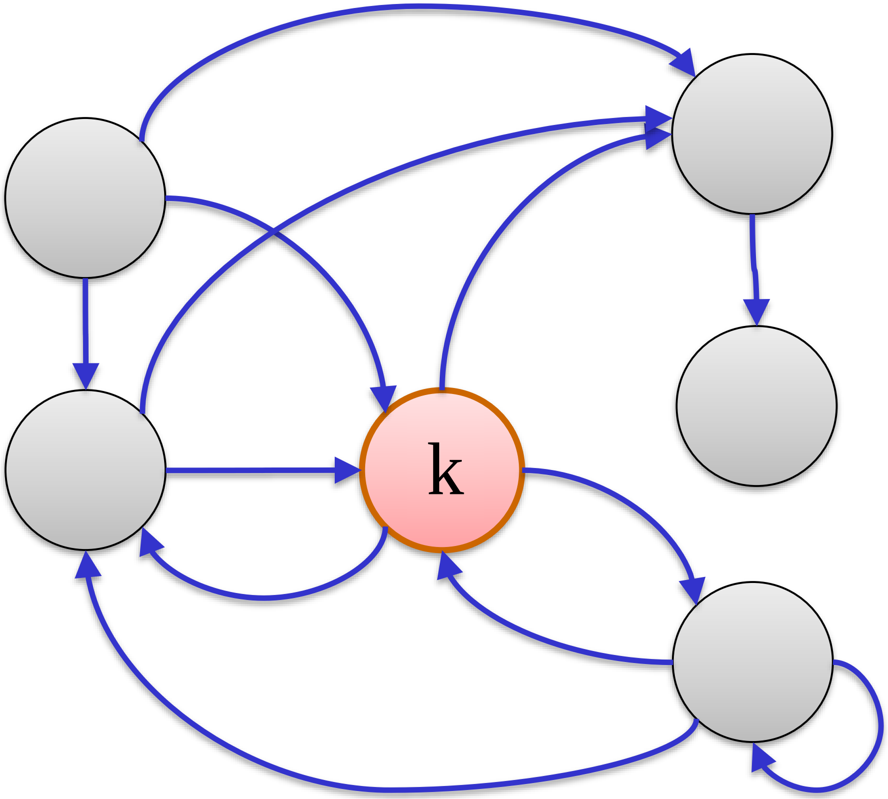
- For each pair of states (i,j) where i,j≠k, the label of (i,j) will be updated as follows:

$$new(i,j) = old(i,j) + old(i,k)\ old(k,k)^*\ old(k,j)$$



Converting GNFA into RE (cont.)

Note, that only arrows between states (i, j) $i, j \neq k$ that are connected (with no $\emptyset$) through the deleted state (k) should be updated.

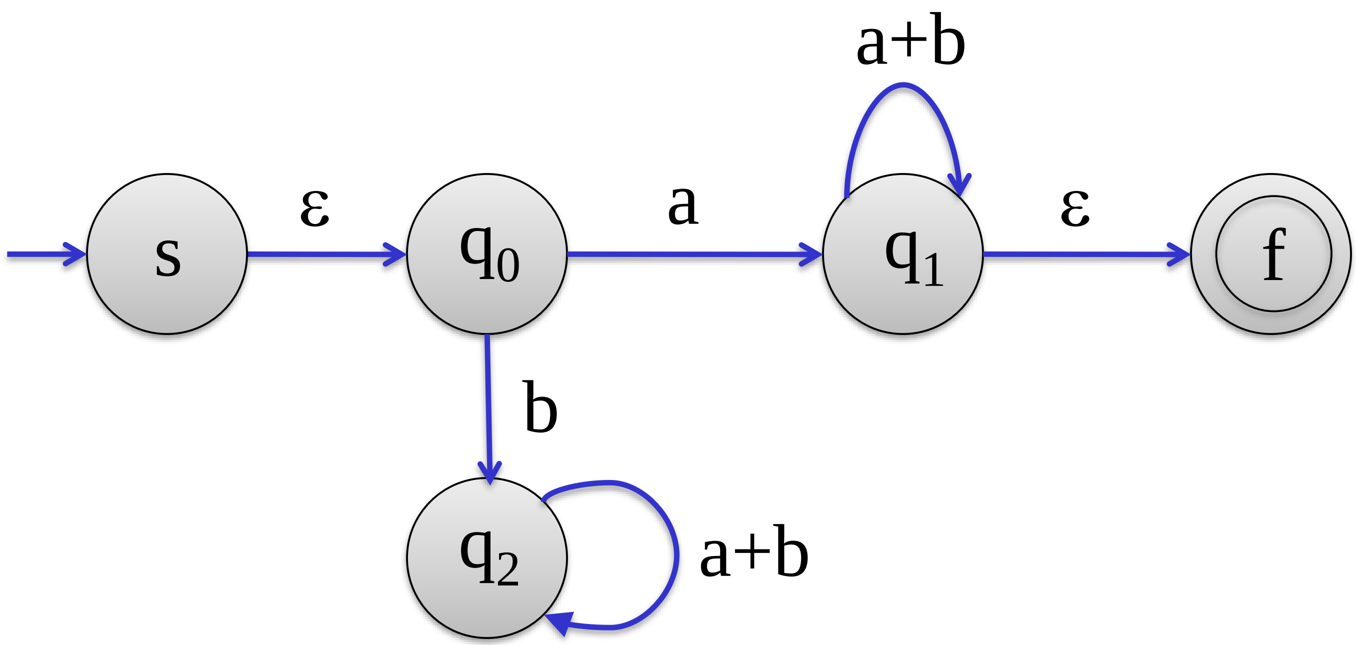


Converting GNFA into RE (cont.)

- The *states* of the new machine are those of the current machine with state k eliminated.
- The *edges* of the new machine are those edges (i,j) for which new(i,j) has been calculated.

- The algorithm terminates when *s* and *f* are the only two remaining states. The RE *new(s,f)* represents the language of the original automaton.

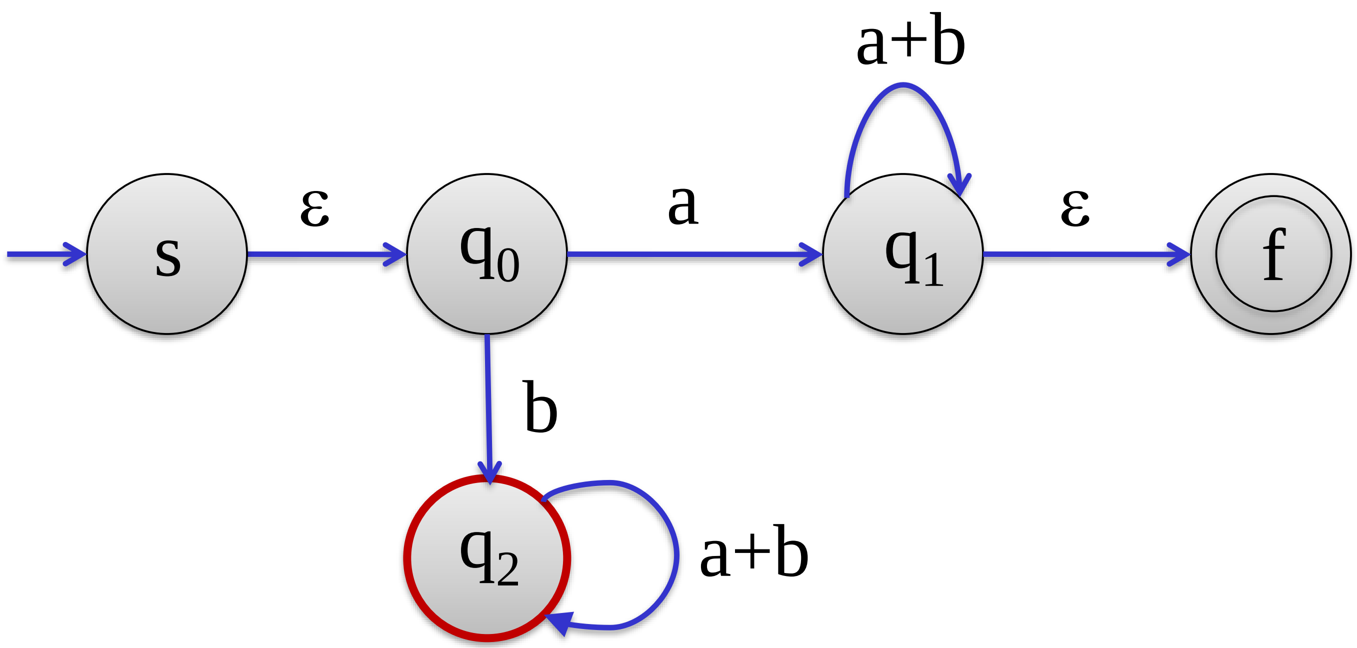
Example: GNFA to RE



Example: GNFA to RE

Eliminating state q_2

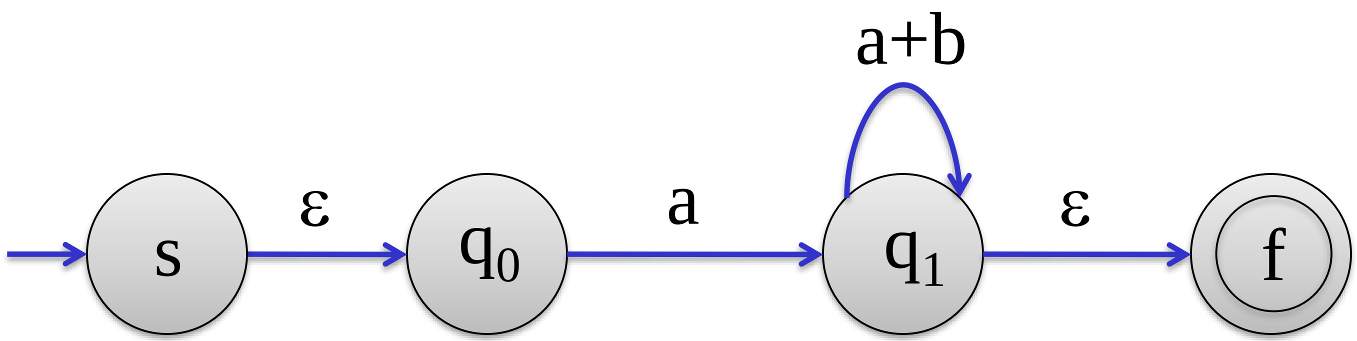
- No paths pass through q_2. There are no states that are connected through q_2. So no need to change anything after deletion of q_2.



Example: GNFA to RE

Eliminating state q_2

- No paths pass through q_2. There are no states that are connected through q_2. So no need to change anything after deletion of q_2.

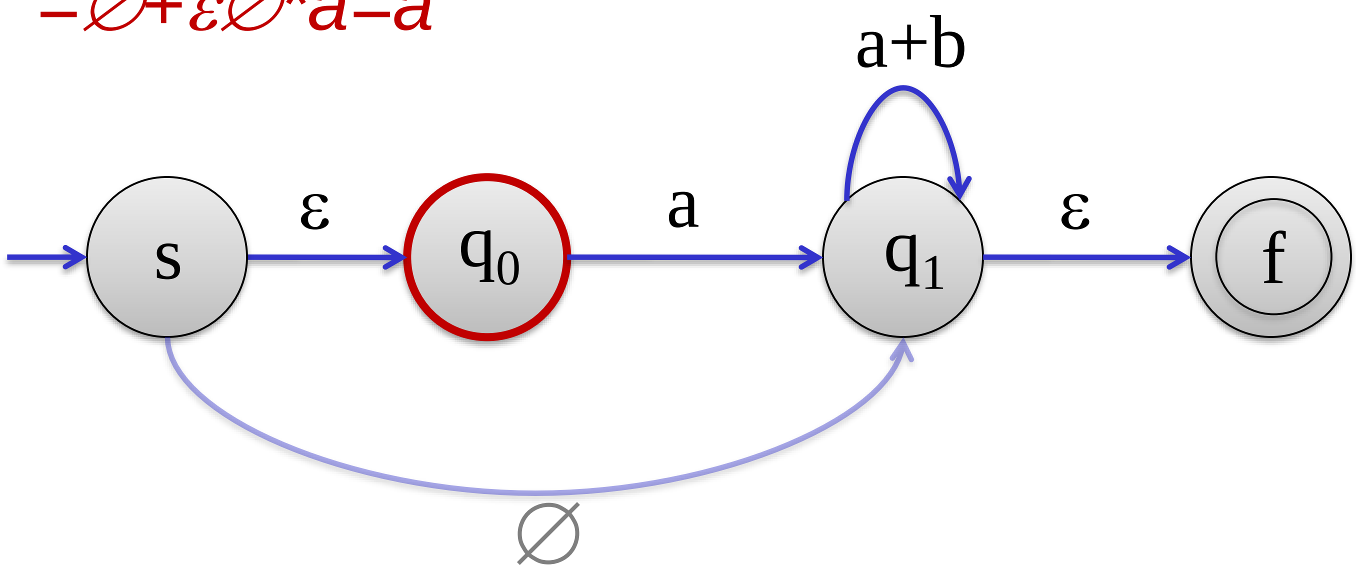


Example: GNFA to RE

Eliminating state q_0

- The only path through q_0 is $s \rightarrow q_1$.
- We add an edge that is labeled by regular expression associated with the deleted edges.

$$new(s,q_1)=old(s,q_1)+old(s,q_0)old(q_0,q_0)^{*}old(q_0,q_1)=$$
$$=\varnothing+\varepsilon\varnothing^{*}a=a$$

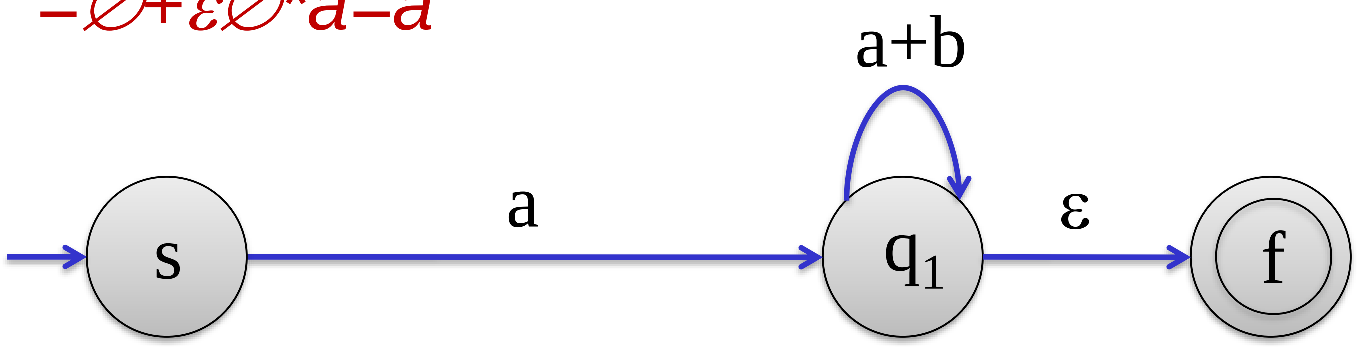


Example: GNFA to RE

Eliminating state q_0

- The only path through q_0 is $s \rightarrow q_1$.
- We add an edge that is labeled by regular expression associated with the deleted edges.

$new(s,q_1)=old(s,q_1)+old(s,q_0)old(q_0,q_0)^*old(q_0,q_1)=$
$=\varnothing+\varepsilon\varnothing^*a=a$



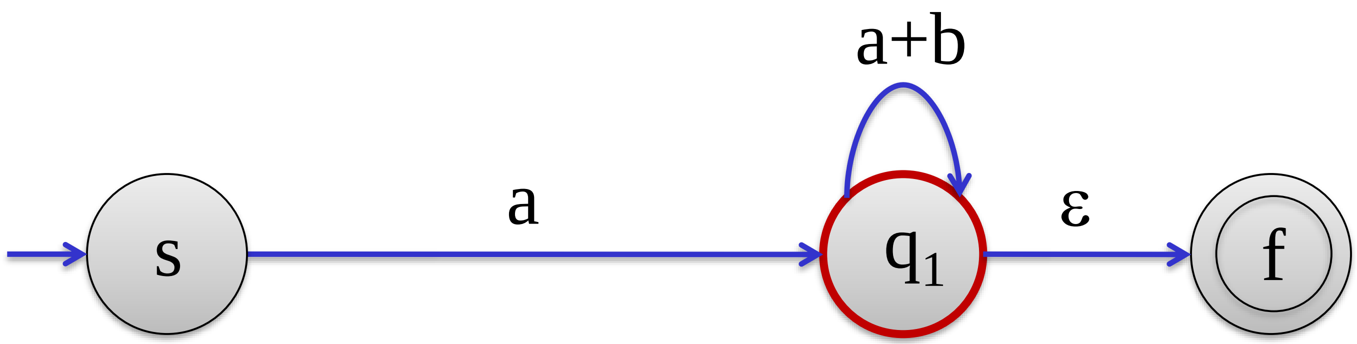
Example: GNFA to RE

Eliminating state q_1

- The only path through q_1 is $s \to f$.

  *new(s,f)=old(s,f)+old(s,q₁)old(q₁,q₁)*old(q₁,f)=*

$= \varnothing + a(a+b)^*\varepsilon = a(a+b)^*$



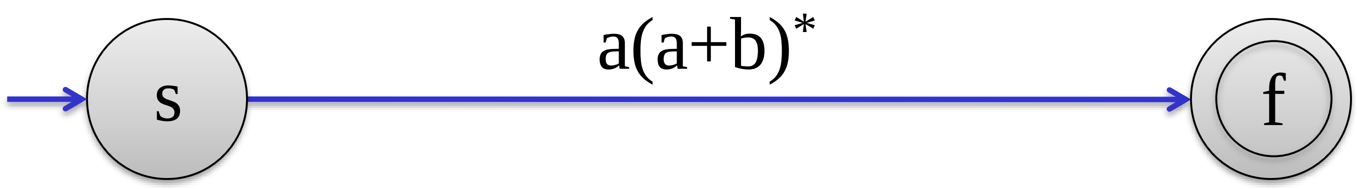
Example: GNFA to RE

Eliminating state q_1

- The only path through q_1 is $s \rightarrow f$.

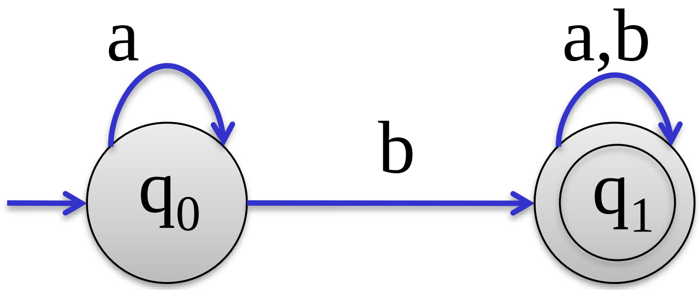
  $new(s,f)=old(s,f)+old(s,q_1)old(q_1,q_1)^*old(q_1,f)=$

$=\varnothing+a(a+b)^*\varepsilon = a(a+b)^*$



Example II

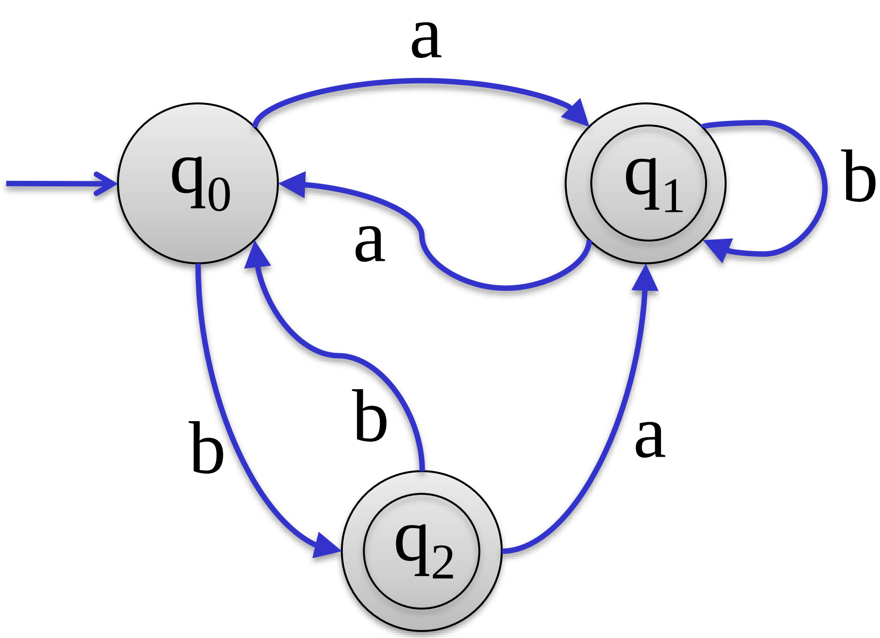
What is the regular expression of L(A)?



Solution: In class

Example III

What is the regular expression of L(A)?



Solution: In class

End of Unit 4