COM3064
Automata Theory

Week 4: Regular Expressions

Lecturer: Dr. Sevgi YİĞİT SERT
Spring 2023

Resources: Introduction to The Theory of Computation, M. Sipser,
Introduction to Automata Theory, Languages, and Computation, J.E. Hopcroft, R. Motwani, and J.D. Ullman
BBM401 Automata Theory and Formal Languages, İlyas Çiçekli

Regular Expressions

- We used **Finite Automata** to describe **regular languages**.
- We can also use **regular expressions** to describe **regular languages**.

- **Regular Expressions** are an **algebraic** way to describe languages.
- Regular Expressions define exactly the same languages that various forms of automata describe: **the regular languages**.
- If E is a regular expression, then $L(E)$ is the regular language that it defines.
- For each regular expression E, we can create a DFA A such that $L(E) = L(A)$.
- For each a DFA A, we can create a regular expression E such that $L(A) = L(E)$

- A regular expression is built up of simpler regular expressions (using defining rules)

Operations on Languages

- Remember: A language is a set of strings
- We can perform operations on languages.

Union: $L \cup M = \{ w : w \in L \text{ or } w \in M \}$

Concatenation: $L \cdot M = \{ w : w = xy,\ x \in L, y \in M \}$

Powers: $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^{k+1} = L \cdot L^k$

Kleene Closure: $L^* = \bigcup_{i=0}^{\infty} L^i$

Operations on Languages - Examples

$L = \{00,11\}$ $M = \{1,01,11\}$

$L \cup M = \{00,11,1,01\}$

$L \cdot M = \{001,0001,0011,111,1101,1111\}$

$L^0 = \{\varepsilon\}$ $L^1 = L = \{00,11\}$ $L^2 = \{0000,0011,1100,1111\}$

$L^* = \{\varepsilon, 00, 11, 0000, 0011, 1100, 1111, 000000, 000011, ...\}$

Kleene closures of all languages (except two of them) are infinite.

1. $\emptyset^* = \{\}^* = \{\varepsilon\}$
2. $\{\varepsilon\}^* = \{\varepsilon\}$

Regular Expressions - Definition

A regular expression: $(a + b \cdot c)^* \cdot (c + \emptyset)$

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

Regular Expressions - Definition

Regular expressions over alphabet Σ

Reg. Expr. E	**Language it denotes L(E)**
$\emptyset$	$\{\ \}$
ε	$\{\varepsilon\}$
$a \in \Sigma$	$\{a\}$

Note:

{a} is the language containing one string, and that string is of length 1.

Regular Expressions - Definition

Union Operator: If $\mathbf{E_1}$ and $\mathbf{E_2}$ are regular expressions, then $\mathbf{E_1+E_2}$ is a regular expression, and $\mathbf{L(E_1+E_2)} = \mathbf{L(E_1) \cup L(E_2)}$.

- Sipser's book use union symbol ∪ to represent **or** operator instead of +.

Concatenation Operator: If $\mathbf{E_1}$ and $\mathbf{E_2}$ are regular expressions, then $\mathbf{E_1E_2}$ is a regular expression, and $\mathbf{L(E_1E_2) = L(E_1) \cdot L(E_2)}$ where $L(E_1) \cdot L(E_2)$ is the set of strings *wx* such that *w* is in $L(E_1)$ and *x* is in $L(E_2)$.

Kleene Closure Operator: If **E** is a regular expression, then **E*** is a regular expression, and $\mathbf{L(E^*) = (L(E))^*}$.

Parentheses: If **E** is a regular expression, then **(E), a parenthesized E,** is also a regular expression, denoting the same language as E. Formally, $\mathbf{L((E)) = L(E)}$.

Regular Expressions - Parentheses

- Parentheses may be used wherever needed to influence the grouping of operators.
- We may remove parentheses by using precedence and associativity rules.

Operator	*Precedence*	*Associativity*
*	highest	
concatenation	next	left associative
+	lowest	left associative

$\mathbf{01^* + 1}$ means $\mathbf{(0(1)^*) + 1}$

Regular Expressions - Examples

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

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

Regular Expressions - Examples

Alphabet $\Sigma = \{0,1\}$

Regular Expression: **01**

- $L(\mathbf{01}) = \{01\}$

$L(\mathbf{01}) = L(\mathbf{0})\, L(\mathbf{1}) = \{0\}\{1\} = \{01\}$

Regular Expression: **01+0**

- $L(\mathbf{01}+\mathbf{0}) = \{01, 0\}$

$L(\mathbf{01}+\mathbf{0}) = L(\mathbf{01}) \cup L(\mathbf{0}) = (L(\mathbf{0})\, L(\mathbf{1})) \cup L(\mathbf{0})$

$= (\{0\}\{1\}) \cup \{0\} = \{01\} \cup \{0\} = \{01,0\}$

Regular Expressions - Examples

Alphabet $\Sigma = \{0,1\}$

Regular Expression: **0***

– L(**0***) = {ε, 0, 00, 000,… } = all strings of 0's, including the empty string

Regular Expression: **(0+10)*(ε+1)**

– L((**0**+**10**)*(ε+**1**)) = { ε, 0, 1, 00, 01, 10, 000, 101, 1010, 10101,… } = all strings of 0's and 1's without two consecutive 1's.

Regular Expressions - Examples

Language: All strings of 0's and 1's starting with 0 and ending with 1

$0(0+1)^*1$

Language: All strings of 0's and 1's with at least two consecutive 0's

$(0+1)^*00\,(0+1)^*$

Regular Expressions - Examples

Regular Expression: **(0+1)(0+1)**

Regular Expression: **$(0+1)^*$**

Language: All strings of 0's and 1's without two consecutive 0's

Language: All strings of 0's and 1's with even number of 0's

Converting Regular Expressions to NFA

Theorem: Every language defined by a regular expression is also defined by a finite automata.

- This theorem says that **every language represented by a regular expression** is a **regular language** (i.e. There is a DFA which recognizes that language)
- In the proof of this theorem, we will create a NFA which recognizes the language of a given regular expression. This means that any language represented by a regular expressions can be recognized by a NFA.
  - Previously, we show how to create an equivalent DFA for a given NFA. This means that any language recognized by a NFA can be recognized by a DFA.

Regular Expressions ⟹ NFA ⟹ DFA ⟹ Regular Languages

Regular Expressions ⟹ Regular Languages

Converting Regular Expressions to NFA

Theorem: Every language defined by a regular expression is also defined by a finite automaton.

Proof:

- Suppose that $L(R)$ is the language of a regular expression R.

- **A NFA construction for a regular expression:** We show that for some NFA N whose language $L(N)$ is equal to $L(R)$, and this NFA N has following properties:
  1. NFA N has exactly one accepting state.
  2. No arcs into the initial state.
  3. No arcs out of the accepting state.

- The **proof is by structural induction on R** following the recursive definition of regular expressions

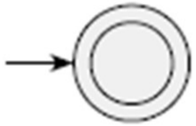
Converting Regular Expressions to NFA - Basis

There are 3 base cases.

a) Regular Expression R = ε

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

NFA N:

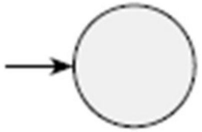


$L(N) = \{\varepsilon\}$

b) Regular Expression R = $\emptyset$

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

NFA N:

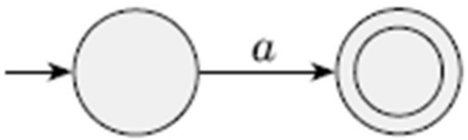


$L(N) = \{\}$

c) Regular Expression R = $a \in \Sigma$

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

NFA N:



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

Converting Regular Expressions to NFA - Induction

Induction Hypothesis:

- We assume that the statement of the theorem is true for immediate subexpressions of a given regular expression.

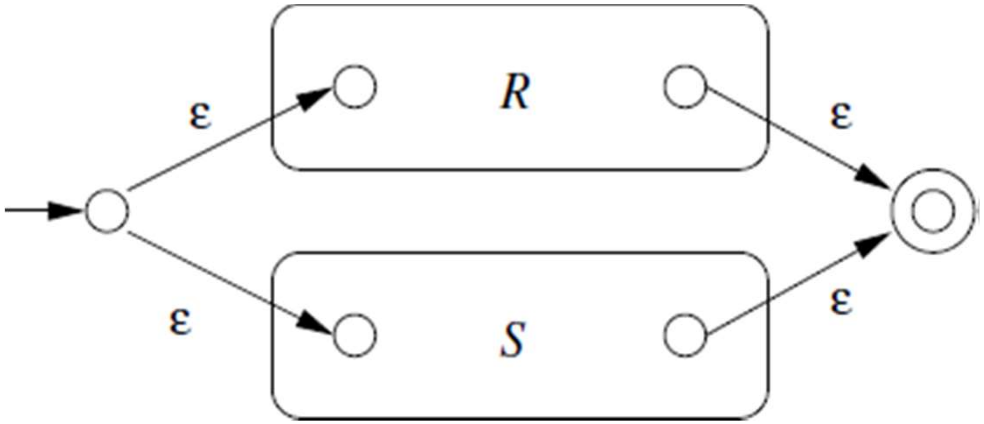
Induction:

- There are four cases for the induction:
  1. R + S
  2. R S
  3. R*
  4. (R)

Converting Regular Expressions to NFA – Induction

Regular Expression: R + S $\boldsymbol{L(R+S) = L(R) \cup L(S)}$

NFA *N*:



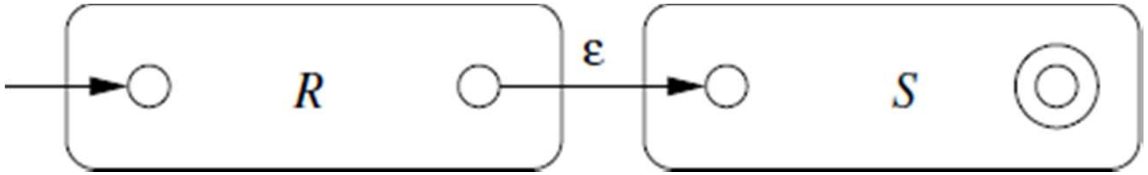
- By IH, we have automata R for regular expression R, and automata S for regular expression S, and **a new automata for R+S is constructed as above**.
- Starting at *new start state*, we can go to *start states of automata R and S*.
- For *some string in L(R) or L(S),* we can reach *accepting state of R or S*.
- From there, we can reach *accepting state of the new automata* by ε–transition.

- **Thus,** $\boldsymbol{L(N) = L(R) \cup L(S)}$

Converting Regular Expressions to NFA – Induction

Regular Expression: R S $\boldsymbol{L(RS) = L(R)\, L(S)}$

NFA *N*:



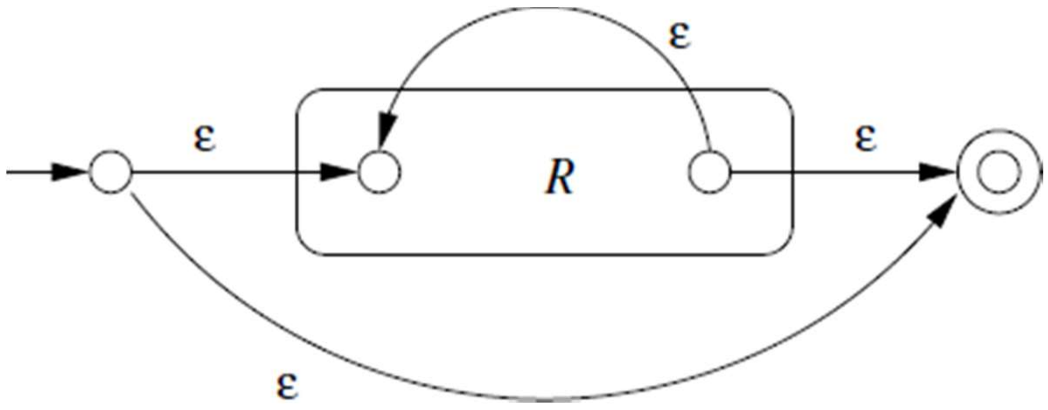
- By IH, we have automata R for regular expression R, and automata S for regular expression S, and **a new automata for RS is constructed as above**.
- Starting at *starting state of R*, we can reach *accepting state of R by recognizing a string in L(R).*
- From *accepting state of R*, we can reach *starting state of S* by ε - transition.
- From *starting state of S*, we can reach *accepting state of S by recognizing a string in L(S).*
- *The accepting state of S is also the accepting state of the new automata N.*

- **Thus,** $\boldsymbol{L(N) = L(R)\, L(S)}$

Converting Regular Expressions to NFA – Induction

Regular Expression: R* $\boldsymbol{L(R^*) = (L(R))^*}$

NFA *N*:



- By IH, we have automata R for regular expression R, and **a new automata for R* is constructed as above**.
- Starting at *new starting state*, we can reach *new accepting state*. ε is in (L(R))*.
- Starting at *new starting state*, we can reach *starting state of R*. From *starting state of R*, we can reach accepting state of R recognizing a string in L(R). We can repeat this one or more times by recognizing strings in L(R), L(R)L(R),….

Thus, $\boldsymbol{L(N) = (L(R))^*}$

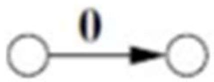
Converting Regular Expressions to NFA – Induction

Regular Expression: (R)

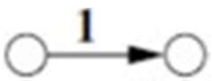
- By IH, we have automata R for regular expression R, and **a new automata for (R) is same as the automata of R**.
- The **automata for R** also serves as the **automata for (R)** since the parentheses do not change the language defined by the expression.

Example: Convert (0+1)*1(0+1) to NFA

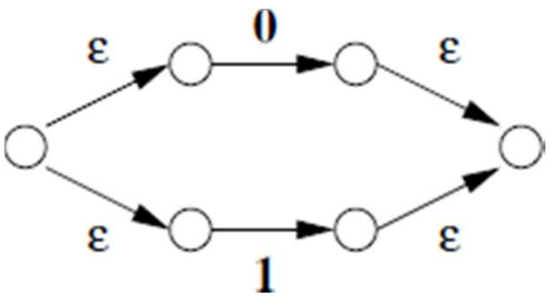
Automata for 0:



Automata for 1:

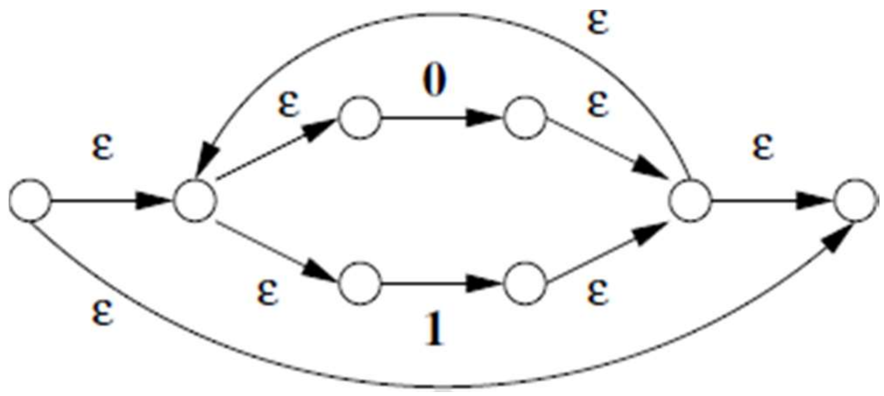


Automata for 0+1:



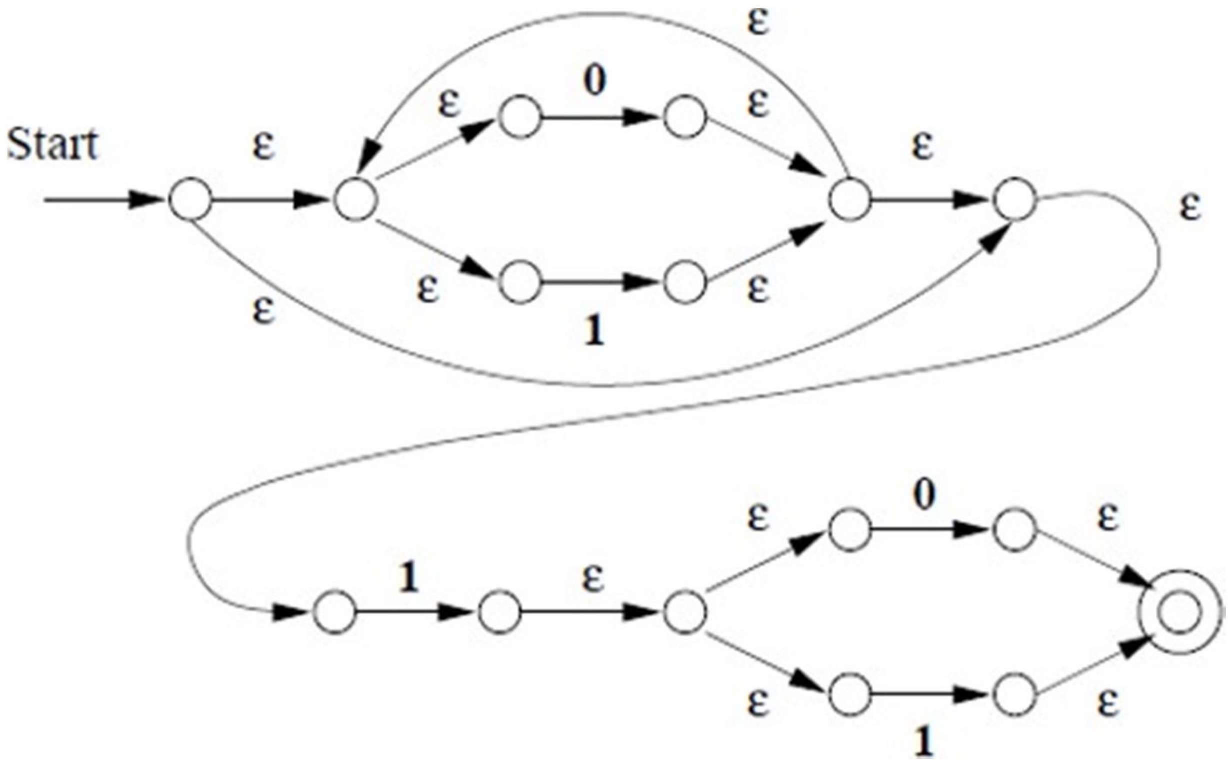
Example: Convert (0+1)*1(0+1) to NFA

Automata for (0+1)*:



Example: Convert (0+1)*1(0+1) to NFA

Automata for (0+1)*1(0+1) :



Converting DFA to Regular Expressions

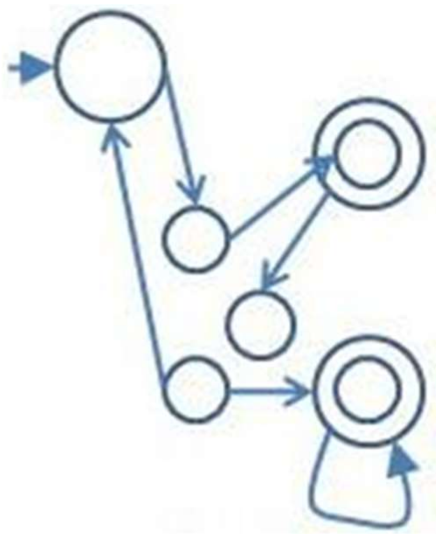
- In order to create a ***regular expression which describes the language of the given DFA***:
- First, we create a **Generalized NFA (GNFA)** from the given DFA
- A GNFA has **generalized transitions** and a **generalization transition** is a ***transition whose label is a regular expression***.
- Then, we will iteratively eliminate states of the GNFA one by one, until only two states (start state and an accepting state) and a single generalized transition is left.
- The label of this single transition (a regular expression) will be the regular expression describes the language of the given DFA.

Converting DFA to GNFA

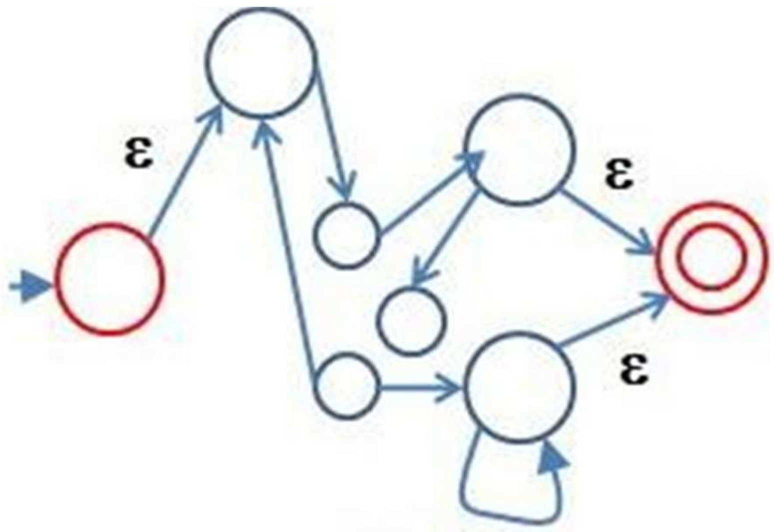
- We will convert the given DFA to **a GNFA in a special form**. We will add two new states to a DFA:
  - A **new start state** with an ε-transition to the original start state, but there will be **no other transitions from any other state to this new start state**.
  - A **new final state** with an ε-transition from all the original final states, but there will be **no other transitions from this new final state to any other state**.
- If the label of the DFA is a single symbol, the corresponding label of the GNFA will be that single symbol: 0 ➔ 0
- If there are more than one symbol on the label of the DFA , the corresponding label of the GNFA will be union (OR) of those symbols: **0,1 ➔ 0+1**
- The previous start and final states will be non-accepting states in this GNFA.

Converting DFA to GNFA

DFA

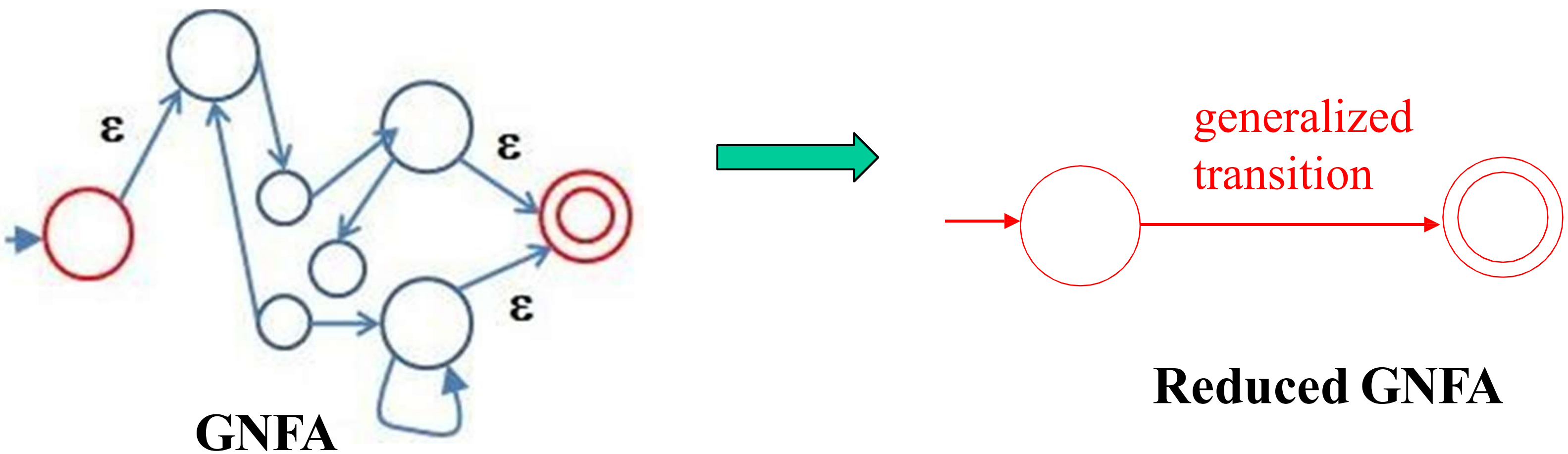


GNFA in a special form



Reducing a GNFA

- We eliminate all states of the GNFA one-by-one leaving only the **start state** and the **final state**.



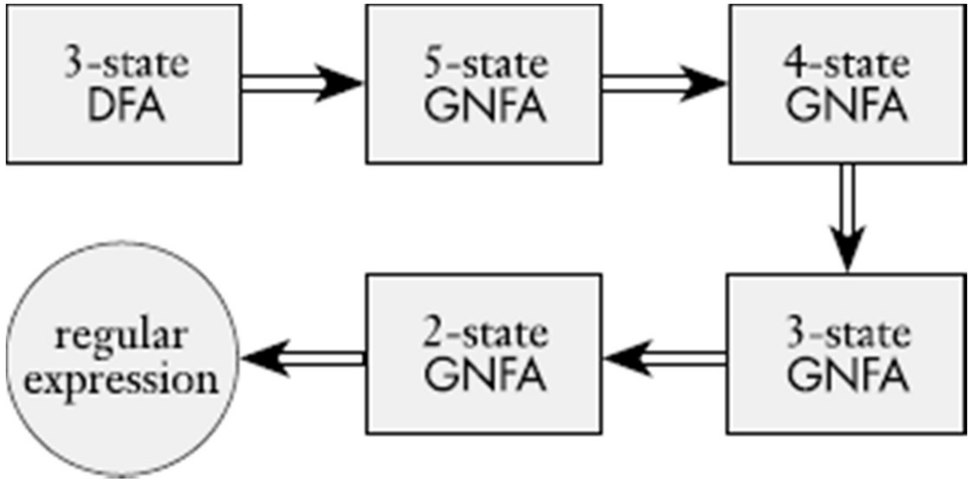
GNFA

Reduced GNFA

- When the GNFA is fully converted, the label of the only generalized transition is the regular expression for the language accepted by the original DFA.

Converting DFA to Regular Expressions

- Assume that our DFA has 3 states.
  - Create a GNFA with 5 states in a special form.
  - Eliminate a state on-by-one until we obtain a GNFA with two states (start state and final state).
  - Label on the arc is the regular expression describing the language of the DFA.



Some Simplification Rules for Regular Expressions

$\emptyset^* = \varepsilon$

$\varepsilon^* = \varepsilon$

$(\varepsilon+\mathbf{R})^* = \mathbf{R}^*$

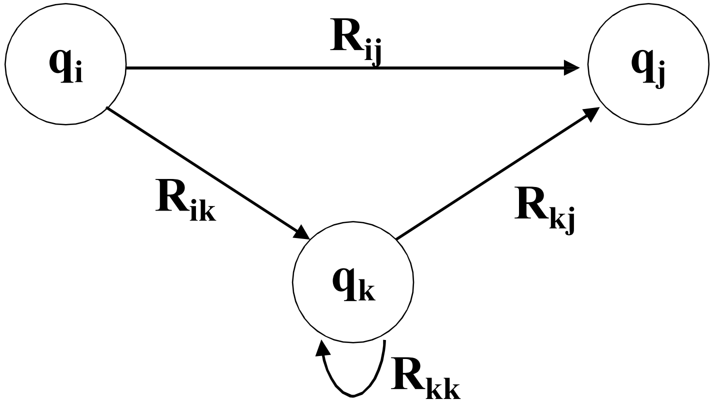
$\varepsilon\mathbf{R} = \mathbf{R}\varepsilon = \mathbf{R}$ ε is the identity for concatenation.

$\emptyset\mathbf{R} = \mathbf{R}\emptyset = \emptyset$ ∅ is an annihilator for concatenation.

$\emptyset+\mathbf{R} = \mathbf{R}+\emptyset = \mathbf{R}$ ∅ is the identity for union.

Eliminating States

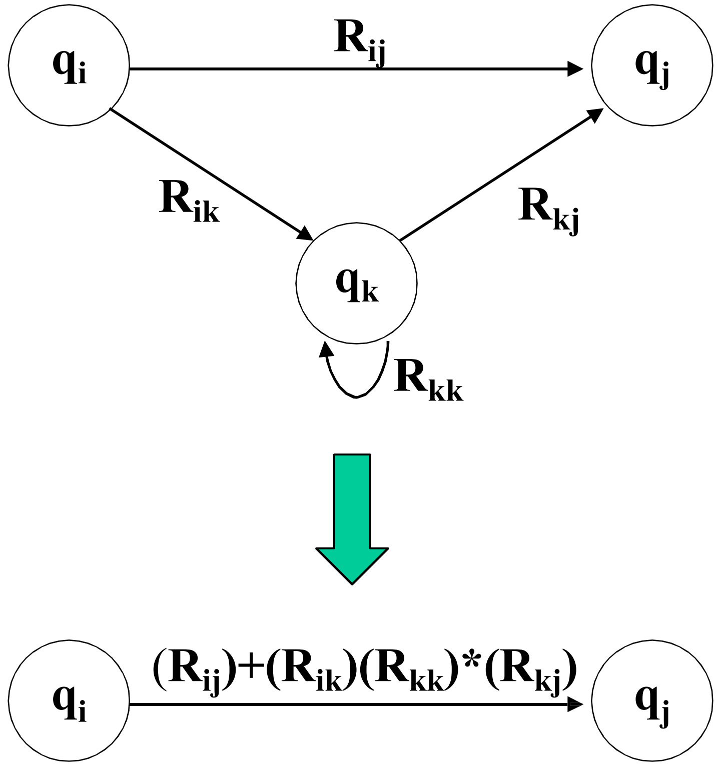
- Suppose we want to eliminate state $\mathbf{q_k}$, and $\mathbf{q_i}$ and $\mathbf{q_j}$ are two of the remaining states (i=j is possible; i.e. $\mathbf{q_i}$ can be equal to $\mathbf{q_j}$).



- How can we modify the transition label between $\mathbf{q_i}$ and $\mathbf{q_j}$ to reflect the fact that $\mathbf{q_k}$ will no longer be there?
  - There are two paths between $\mathbf{q_i}$ and $\mathbf{q_j}$
    - Direct path with regular expression $\mathbf{R_{ij}}$
    - Path via $\mathbf{q_k}$ with the regular expression $\mathbf{(R_{ik})\ (R_{kk})^*\ (R_{kj})}$

Eliminating States

- There are two paths between q_i and q_j
  - Direct path with regular expression R_{ij}
  - Path via q_k with the regular expression $(R_{ik})\ (R_{kk})^*\ (R_{kj})$

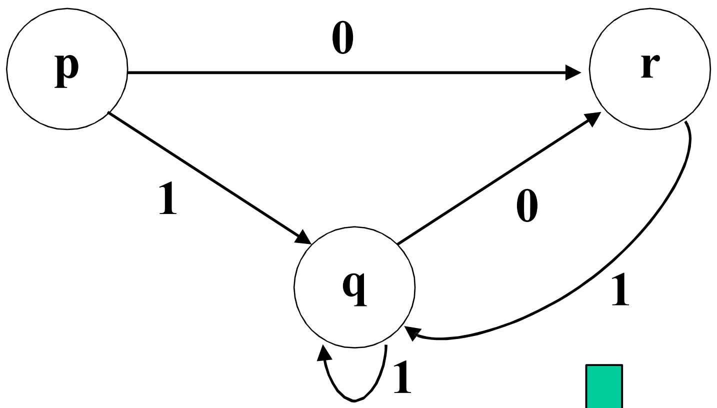


- After removing q_k ,the new label would be

new $(R_{ij}) = (R_{ij}) + (R_{ik})\ (R_{kk})^*\ (R_{kj})$

Eliminating States

- When we are eliminating a state q, we have to update labels of state pairs p and r such that there is a transition from p to q and there is a transition from q to r.
  - p and r can be same state.
- Missing arc labels are ∅

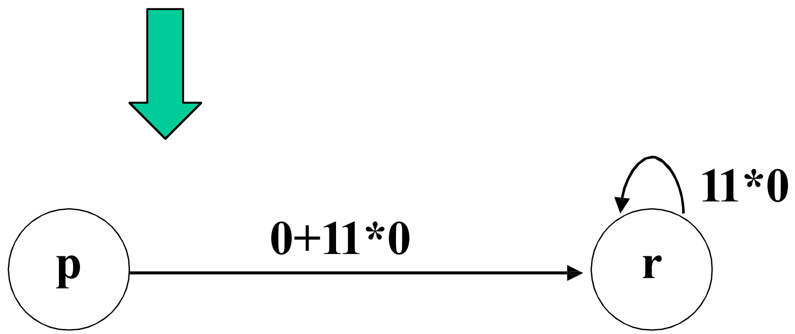


$R_{pp} = R_{pp} + R_{pq} (R_{qq})^* R_{qp} = \emptyset + 1(1)^*\emptyset = \emptyset$

$R_{pr} = R_{pr} + R_{pq} (R_{qq})^* R_{qr} = 0 + 1(1)^*0 = \mathbf{0+11^*0}$

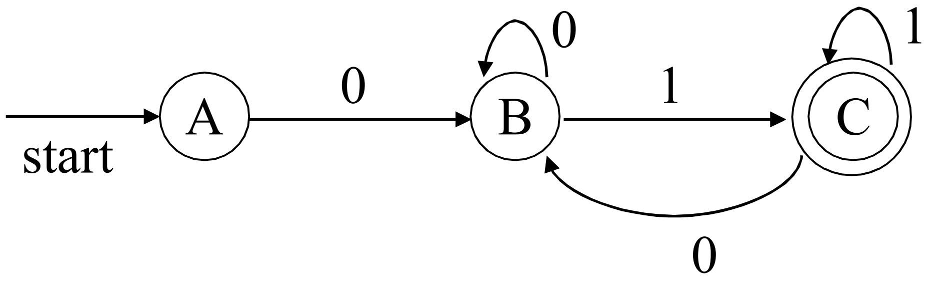
$R_{rr} = R_{rr} + R_{rq} (R_{qq})^* R_{qr} = \emptyset + 1(1)^*0 = \mathbf{11^*0}$

$R_{rp} = R_{rp} + R_{rq} (R_{qq})^* R_{qp} = \emptyset + 1(1)^*\emptyset = \emptyset$

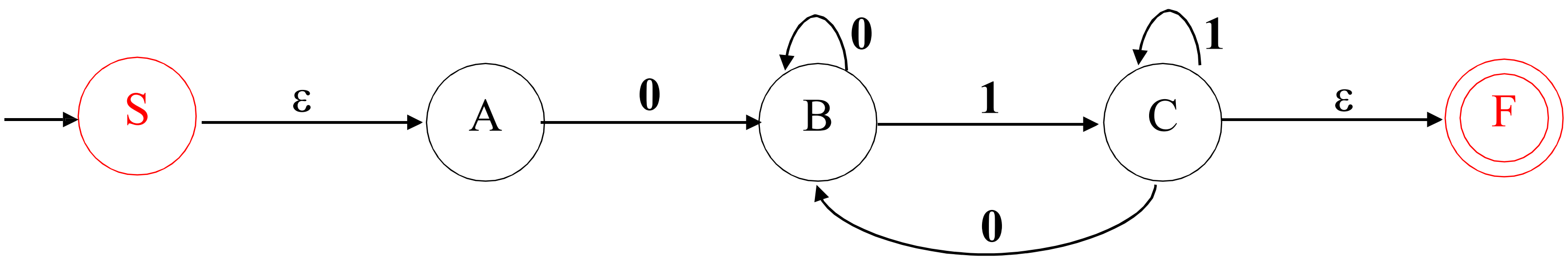


Converting DFA to Regular Expressions: Example

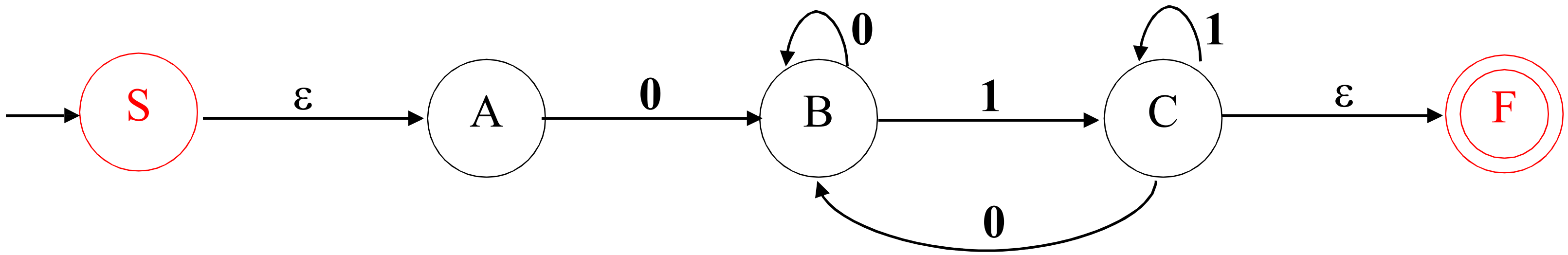
A DFA



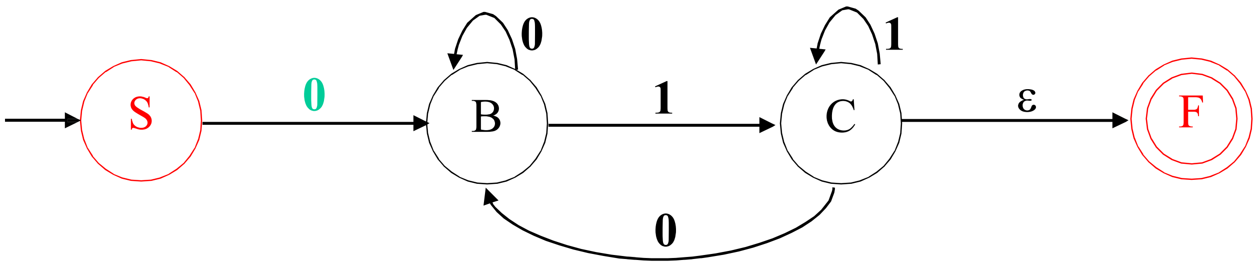
A GNFA in a special form:



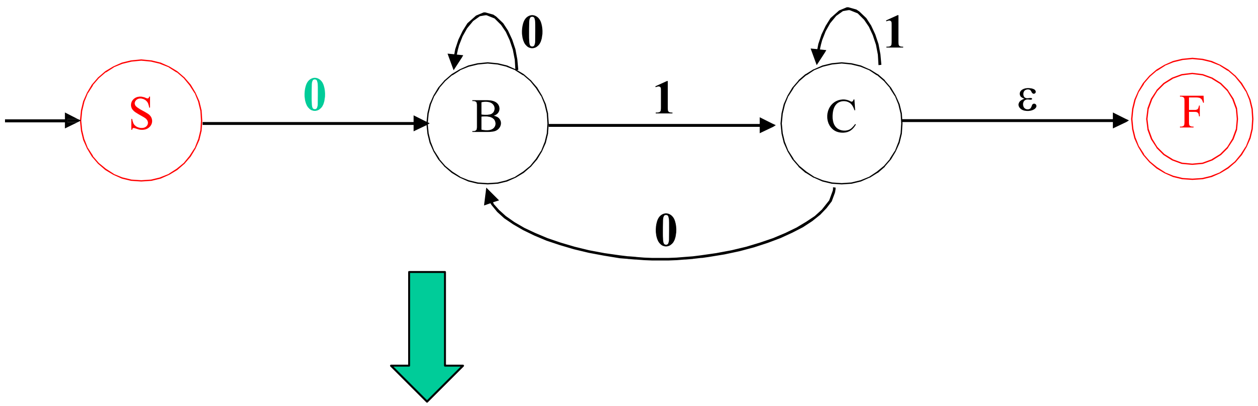
Converting DFA to Regular Expressions: Eliminate A



new $R_{SB} = R_{SB} + R_{SA}\ (R_{AA})^*\ R_{AB} = \emptyset + \varepsilon\ (\emptyset)^*\ 0 = \mathbf{0}$

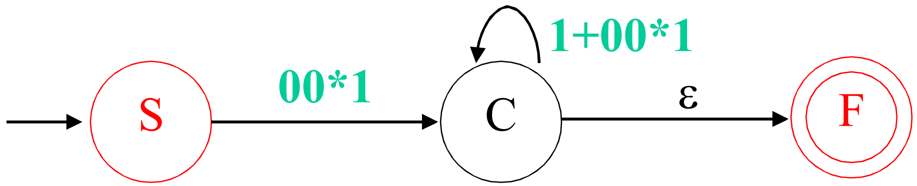


Converting DFA to Regular Expressions: Eliminate B

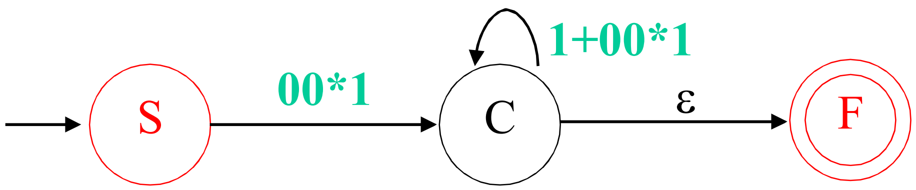


new $R_{SC} = R_{SC} + R_{SB} (R_{BB})^* R_{BC} = \emptyset + 0\ (0)^*\ 1$ = **00*1**

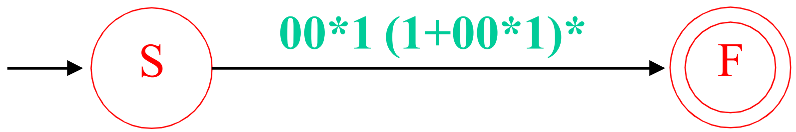
new $R_{CC} = R_{CC} + R_{CB} (R_{BB})^* R_{BC} = 1 + 0\ (0)^*\ 1$ = **1+00*1**

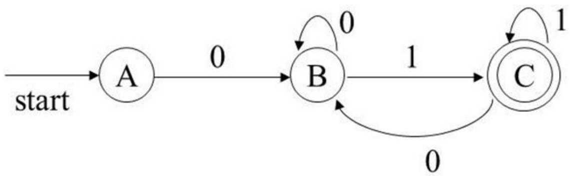


Converting DFA to Regular Expressions: Eliminate C



new $R_{SF} = R_{SF} + R_{SC} (R_{CC})^* R_{CF} = \emptyset + 00^*1\ (1+00^*1)^*\ \varepsilon =$ **00*1 (1+00*1)***

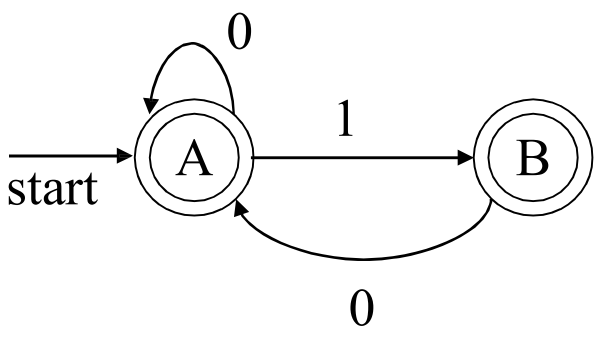




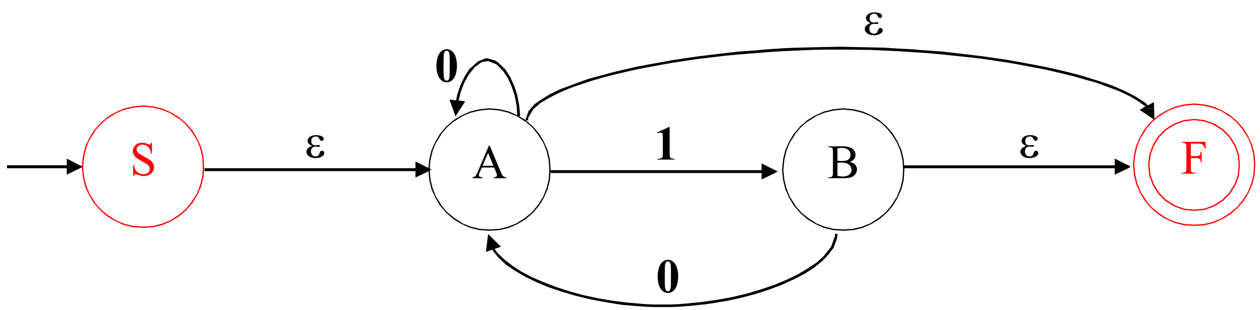
Thus, the regular expression is: 00*1 (1+00*1)*

Converting DFA to Regular Expressions: Example

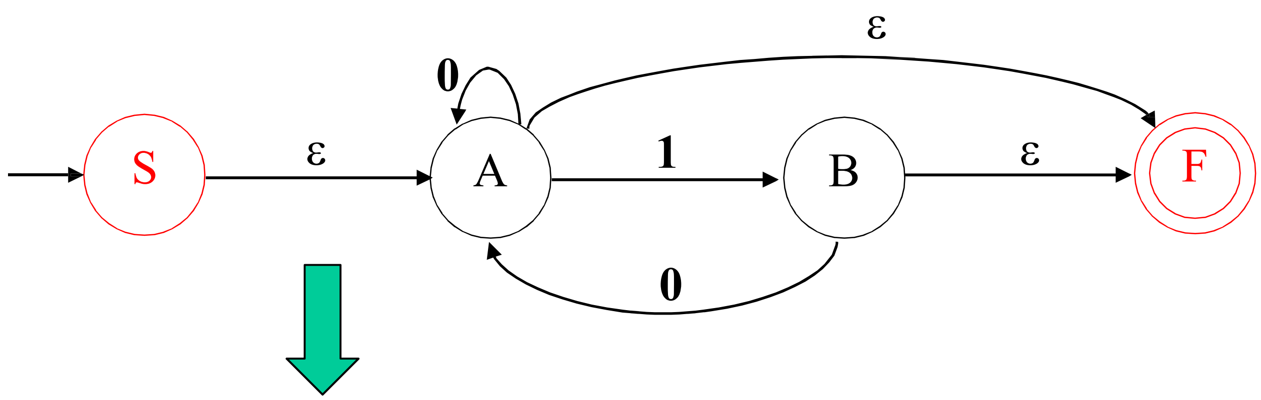
- **A DFA**



- **A GNFA in a special form:**



Converting DFA to Regular Expressions: Eliminate A

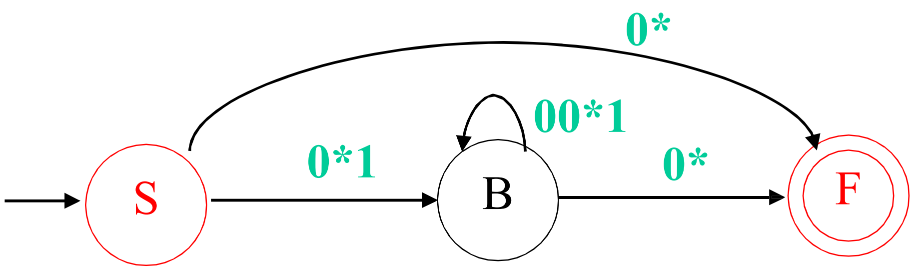


$R_{SF} = R_{SF} + R_{SA} (R_{AA})^* R_{AF} = \emptyset + \varepsilon (0)^* \varepsilon = \mathbf{0^*}$

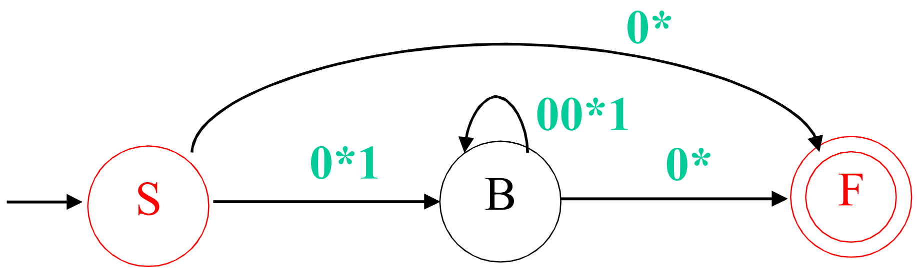
$R_{SB} = R_{SB} + R_{SA} (R_{AA})^* R_{AB} = \emptyset + \varepsilon (0)^* 1 = \mathbf{0^*1}$

$R_{BB} = R_{BB} + R_{BA} (R_{AA})^* R_{AB} = \emptyset + 0 (0)^* 1 = \mathbf{00^*1}$

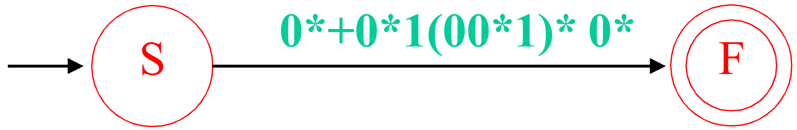
$R_{BF} = R_{BF} + R_{BA} (R_{AA})^* R_{AF} = \varepsilon + 0 (0)^* \varepsilon = \varepsilon + 00^* = \mathbf{0^*}$



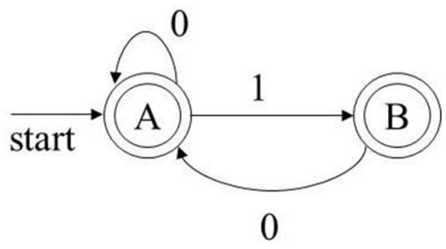
Converting DFA to Regular Expressions: Eliminate B



$R_{SF} = R_{SF} + R_{SB} (R_{BB})^* R_{BF} = 0^* + 0^*1\ (00^*1)^*\ 0^* =$ **0*+0*1(00*1)* 0***



Thus, the regular expression is: **0*+0*1(00*1)*0***

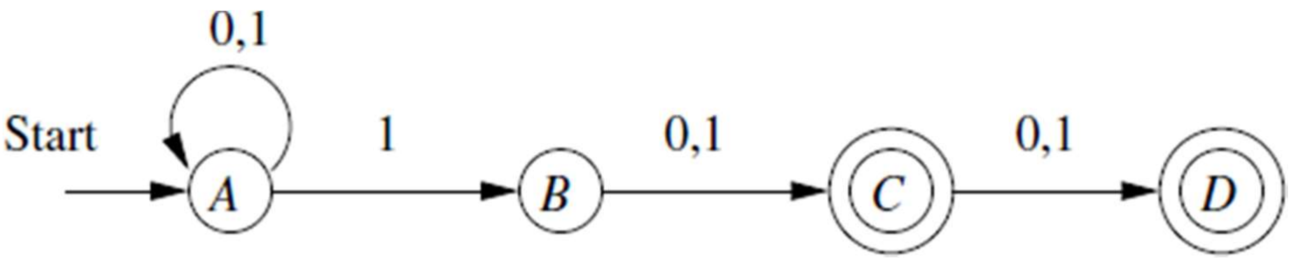


Converting NFA to Regular Expressions

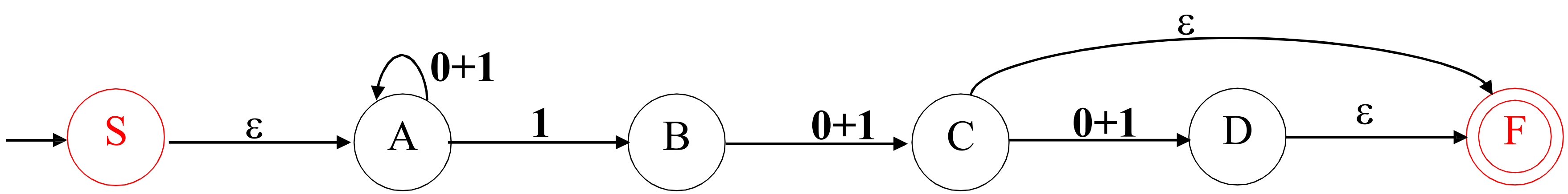
- We can use the conversion by state elimination algorithm for NFA too.
- First, we have to represent the given NFA as a GNFA.
  - If the label is a single symbol, the label of the generalized automata will be that single symbol.
    - 0 ➔ 0 ε ➔ ε
  - If there are more than one symbol, the label will be union (OR) of those symbols.
    - 0,1 ➔ 0+1 0,1,ε ➔ **0+1**+ε

Converting NFA to Regular Expressions: Example

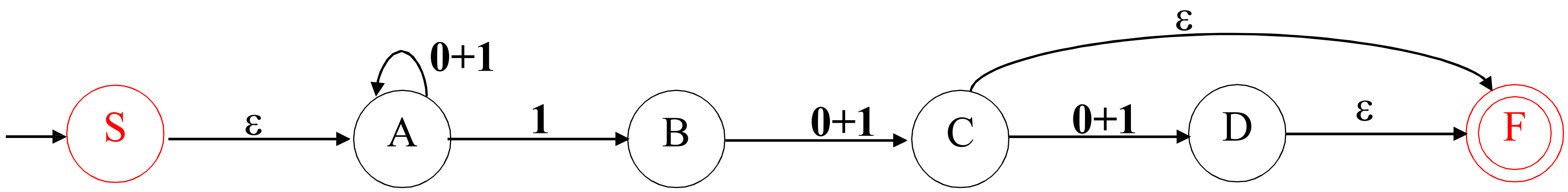
Convert a NFA to a regular expression



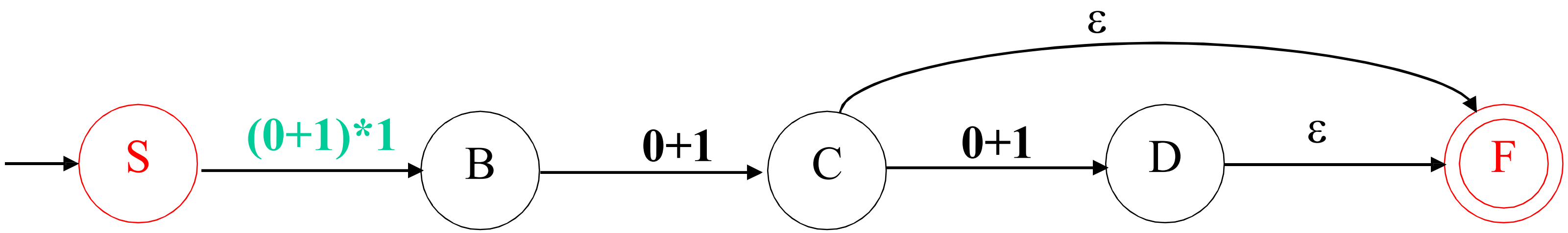
Convert a NFA to a GNFA in a special form.



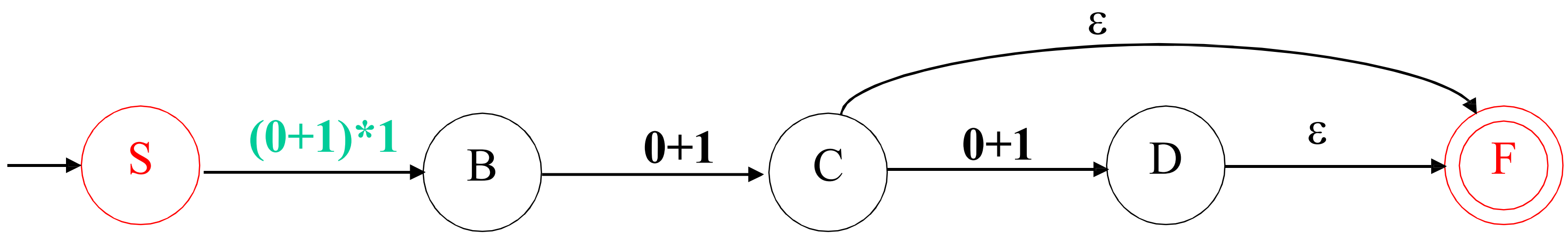
Converting NFA to Regular Expressions: Eliminate A



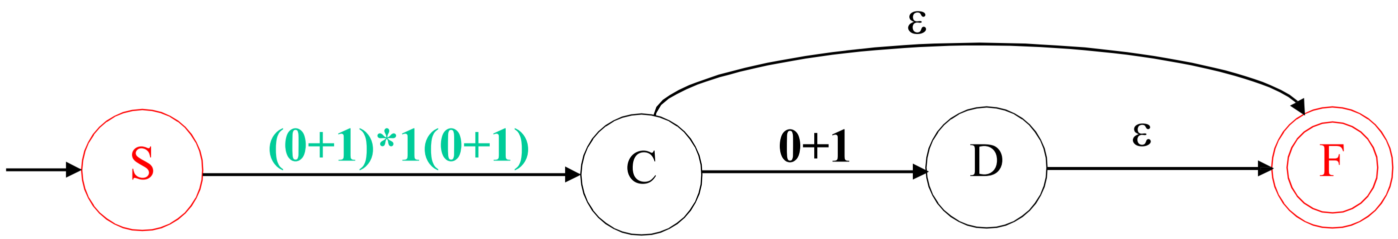
$R_{SB} = R_{SB} + R_{SA} (R_{AA})^* R_{AB} = \emptyset + \varepsilon (0+1)^* 1 = \mathbf{(0+1)^*1}$



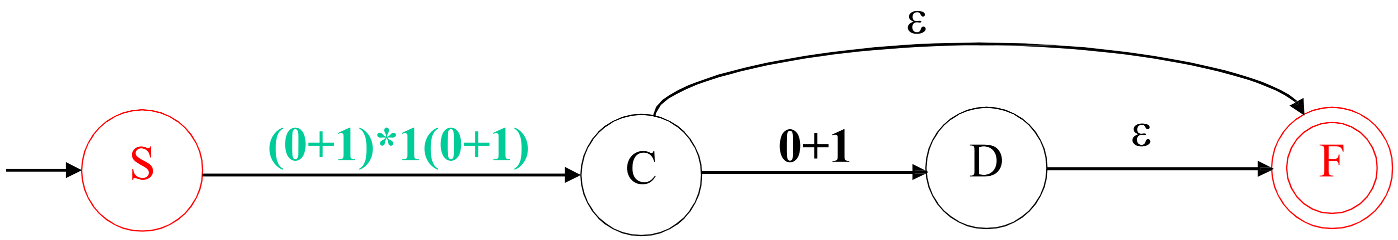
Converting NFA to Regular Expressions: Eliminate B



$R_{SC} = R_{SC} + R_{SB}\,(R_{BB})^*\,R_{BC} = \emptyset + (0+1)^*1\,(\emptyset)^*\,(0+1) =$ **$(0+1)^*1(0+1)$**

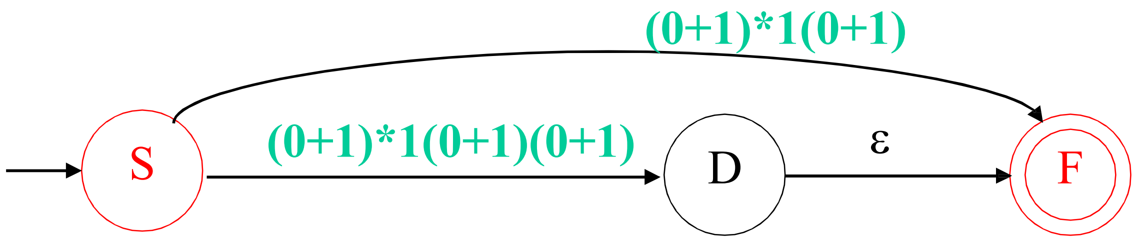


Converting NFA to Regular Expressions: Eliminate C

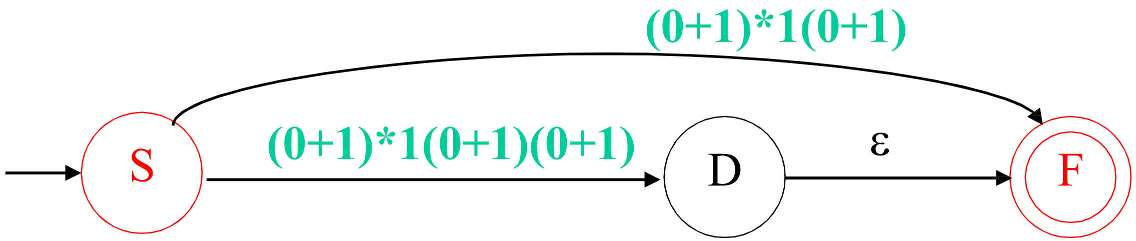


$R_{SD} = R_{SD} + R_{SC} (R_{CC})^* R_{CD} = \emptyset + (0+1)^*1(0+1) (\emptyset)^* (0+1) = \mathbf{(0+1)^*1(0+1)(0+1)}$

$R_{SF} = R_{SF} + R_{SC} (R_{CC})^* R_{CF} = \emptyset + (0+1)^*1(0+1) (\emptyset)^* \varepsilon = \mathbf{(0+1)^*1(0+1)}$

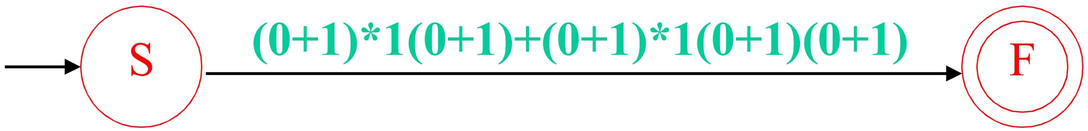


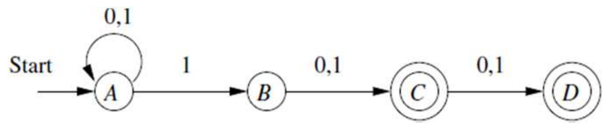
Converting NFA to Regular Expressions: Eliminate D



$$R_{SF} = R_{SF} + R_{SD} (R_{DD})^* R_{DF} = (0+1)^*1(0+1) + (0+1)^*1(0+1)(0+1)\ (\emptyset)^*\ \varepsilon$$

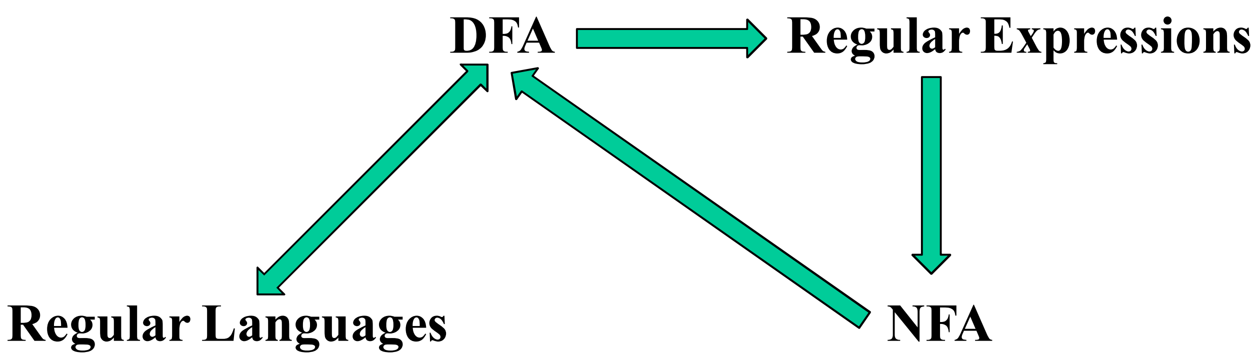
$$= (0+1)^*1(0+1) + (0+1)^*1(0+1)(0+1)$$

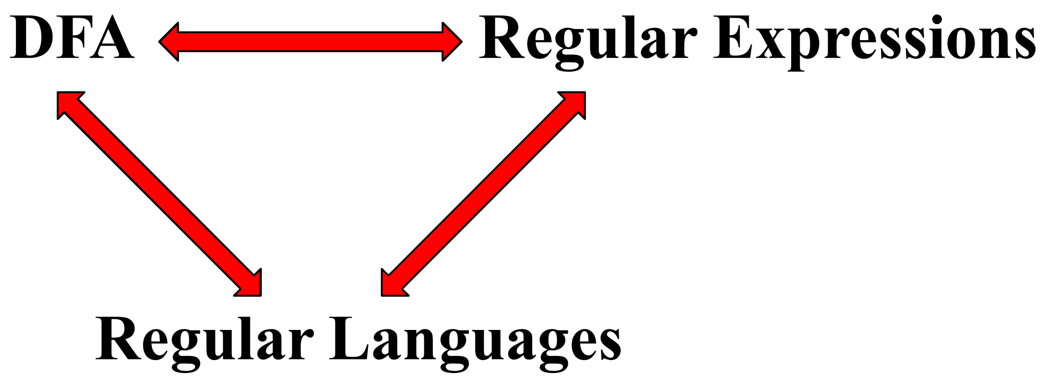




Thus, the regular expression is: **(0+1)*1(0+1)+(0+1)*1(0+1)(0+1)**

Regular Languages, DFA, Regular Expressions





Regular Expressions - Examples

Regular Expression: **(0+1)(0+1)**

– L(**(0+1)(0+1)**) = {00,01,10,11} = all strings of 0's and 1's of length 2.

Regular Expression: **(0+1)***

– L(**(0+1)***) = all strings with 0 and 1, including the empty string

Language: All strings of 0's and 1's without two consecutive 0's

$((\mathbf{1}+\mathbf{01})^*(\varepsilon+\mathbf{0}))$

Language: All strings of 0's and 1's with even number of 0's

$\mathbf{1^*(01^*01^*)^*}$