Regular Expressions and Regular Languages

Operations on Languages

Remember: A ***language*** is a set of strings

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

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

Powers: $L^0 = \{\epsilon\},\ L^1 = L,\ L^{k+1} = L.L^k$

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

Operations on Languages - Examples

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

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

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

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

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

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

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

Regular Expressions

- *Regular Expressions* are an algebraic way to describe languages.
- *Regular Expressions* describe exactly the *regular languages*.
- If E is a regular expression, then L(E) is the regular language it defines.
- A regular expression is built up of simpler regular expressions (using defining rules)
- 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)

Regular Expressions - Definition

Regular expressions over alphabet Σ

	Reg. Expr. E	Language it denotes L(E)
Basis 1:	$\mathbf{\Phi}$	$\{\}$
Basis 2:	$\boldsymbol{\varepsilon}$	$\{\varepsilon\}$
Basis 3:	$\mathbf{a} \in \mathbf{\Sigma}$	$\{a\}$

Note:

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

Regular Expressions - Definition

Induction 1 – or : If E_1 and E_2 are regular expressions, then $\mathbf{E_1+E_2}$ is a regular expression, and $L(E_1+E_2) = L(E_1) \cup L(E_2)$.

Induction 2 – concatenation: If E_1 and E_2 are regular expressions, then $\mathbf{E_1E_2}$ is a regular expression, and $L(E_1E_2) = L(E_1)L(E_2)$ where $L(E_1)L(E_2)$ is the set of strings *wx* such that *w* is in $L(E_1)$ and *x* is in $L(E_2)$.

Induction 3 – Kleene Closure: If E is a regular expression, then **E*** is a regular expression, and $L(\mathbf{E^*}) = (L(E))^*$.

Induction 4 – Pranteheses: If E is a regular expression, then **(E)** is a regular expression, and $L(\ \mathbf{(E)}\ ) = L(E)$.

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

$\mathbf{ab^{*}+c}$ means $\mathbf{(a((b)^{*}))+(c)}$

Regular Expressions - Examples

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

- $L(\mathbf{01}) = \{01\}$. $L(\mathbf{01}) = L(\mathbf{0})\,L(\mathbf{1}) = \{0\}\{1\} = \{01\}$
- $L(\mathbf{01}+\mathbf{0}) = \{01, 0\}$. $L(\mathbf{01}+\mathbf{0}) = L(\mathbf{01}) \cup L(\mathbf{0}) = (L(\mathbf{0})\,L(\mathbf{1})) \cup L(\mathbf{0})$
  $= (\{0\}\{1\}) \cup \{0\} = \{01\} \cup \{0\} = \{01,0\}$
- $L(\mathbf{0}(\mathbf{1}+\mathbf{0})) = \{01, 00\}$.
  - Note order of precedence of operators.
- $L(\mathbf{0}^*) = \{\epsilon, 0, 00, 000, \ldots\}$.
- $L((\mathbf{0}+\mathbf{10})^*(\epsilon+\mathbf{1}))$ = all strings of 0's and 1's without two consecutive 1's.
- $L((\mathbf{0+1})(\mathbf{0+1})) = \{00,01,10,11\}$
- $L((\mathbf{0+1})^*)$ = all strings with 0 and 1, including the empty string

Regular Expressions - Examples

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

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

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

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

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

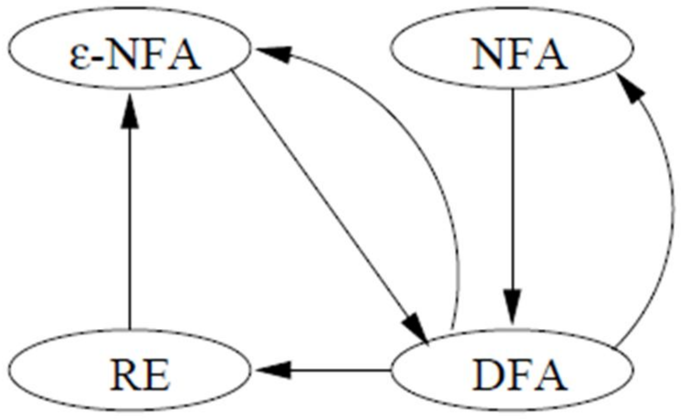
$\mathbf{(0+1)^{*}00\ (0+1)^{*}}$

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

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

Equivalence of FA's and Regular Expressions

- We have already shown that DFA's, NFA's, and ε-NFA's all are equivalent.
- To show FA’s equivalent to regular expressions we need to establish that
  1. For every DFA A we can construct a regular expression R, s.t. L(R) = L(A).
  2. For every regular expression R there is a ε-NFA A (a DFA A), s.t. L(A) = L(R).



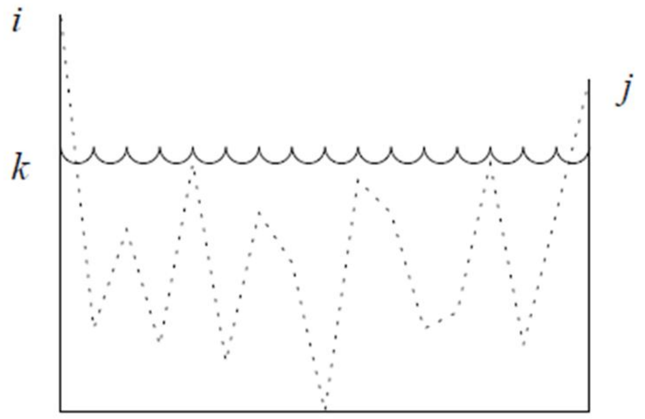
$$\begin{tikzcd}
\varepsilon\text{-NFA} \arrow[dr] & \text{NFA} \arrow[d] \\
\text{RE} \arrow[u] & \text{DFA} \arrow[l] \arrow[ul, bend right] \arrow[u, bend right]
\end{tikzcd}$$

From DFA's to Regular Expressions

Theorem 3.4: For every DFA A = (Q, Σ, δ, q_0, F) there is a regular expression R, s.t. L(R) = L(A).

Proof:

- Let the states of A be {1,2,...,n} with 1 being the start state.
- Let $R_{ij}^{(k)}$ be a regular expression describing the set of labels (strings) of all paths in A from state i to state j going through intermediate states {1,2,...,k} only.
  - Note that the beginning and end points of the path are not "intermediate." so there is no constraint that i and/or j be less than or equal to k.



$R_{ij}^{(k)}$ Definition -Basis

Basis: k = 0, i.e. no intermediate states.

Case 1: i ≠ j

$$R_{ij}^{(0)} = \bigoplus_{\{a \in \Sigma : \delta(i,a)=j\}} a$$

a) If there is no such symbol a, then $R_{ij}^{(0)} = \emptyset$.

b) If there is exactly one such symbol a, then $R_{ij}^{(0)} = \mathbf{a}$.

c) If there are symbols $a_1, a_2, \ldots, a_k$ that label arcs from state i to state j, then $R_{ij}^{(0)} = \mathbf{a}_1 + \mathbf{a}_2 + \cdots + \mathbf{a}_k$.

Case 2: i = j

$$R_{ii}^{(0)} = \left(\bigoplus_{\{a \in \Sigma : \delta(i,a)=i\}} a \right) + \epsilon$$

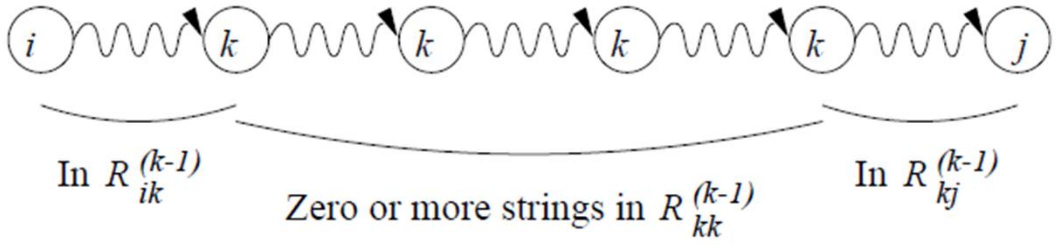
$R_{ij}^{(k)}$ Definition -Induction

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)}\left(R_{kk}^{(k-1)}\right)^* R_{kj}^{(k-1)}$$

Case1: The path does not. go through state k at all. In this case, the label of the path is in the language of $R_{ij}^{(k-1)}$

Case 2: The path goes through state k at, least once.

- The first goes from state i to state k without passing through k,
- the last piece goes from k to j without passing through k, and
- all the pieces in the middle go from k to itself, without passing through k.

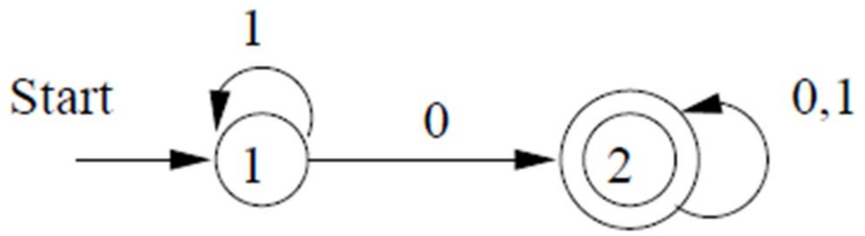


$R_{ij}^{(k)}$ Definition

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)}\left(R_{kk}^{(k-1)}\right)^{*}R_{kj}^{(k-1)}$$

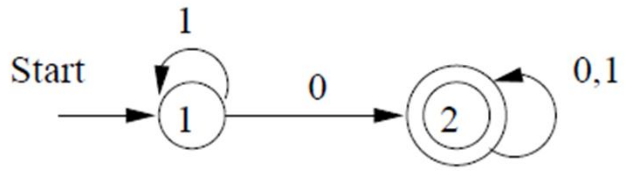
- If we construct these expressions in order of increasing superscript, then since each $R_{ij}^{(k)}$ depends only on expressions with a smaller superscript, then all expressions are available when we need them.
- Eventually, we have $R_{ij}^{(n)}$ for all i and j. We may assume that state 1 is the start state, although the accepting states could be any set of the states.
- The regular expression for the language of the automaton is then the sum (union) of all expressions $R_{ij}^{(n)}$ such that state j is an accepting state.

Example



$R_{11}^{(0)}$	$\epsilon + 1$
$R_{12}^{(0)}$	0
$R_{21}^{(0)}$	$\emptyset$
$R_{22}^{(0)}$	$\epsilon + 0 + 1$

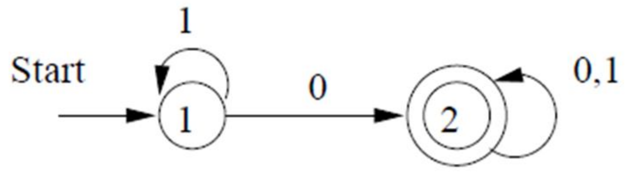
Example $R_{ij}^{(1)}$



$$R_{ij}^{(1)} = R_{ij}^{(0)} + R_{i1}^{(0)}\left(R_{11}^{(0)}\right)^* R_{1j}^{(0)}$$

	By direct substitution	Simplified
$R_{11}^{(1)}$	$\epsilon + 1 + (\epsilon + 1)(\epsilon + 1)^*(\epsilon + 1)$	1^*
$R_{12}^{(1)}$	$0 + (\epsilon + 1)(\epsilon + 1)^*0$	1^*0
$R_{21}^{(1)}$	$\emptyset + \emptyset(\epsilon + 1)^*(\epsilon + 1)$	$\emptyset$
$R_{22}^{(1)}$	$\epsilon + 0 + 1 + \emptyset(\epsilon + 1)^*0$	$\epsilon + 0 + 1$

Example $R_{ij}^{(2)}$



$$R_{ij}^{(2)} = R_{ij}^{(1)} + R_{i2}^{(1)}\left(R_{22}^{(1)}\right)^* R_{2j}^{(1}$$

	By direct substitution	Simplified
$R_{11}^{(2)}$	$1^* + 1^*0(\epsilon + 0 + 1)^*\emptyset$	1^*
$R_{12}^{(2)}$	$1^*0 + 1^*0(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)$	$1^*0(0 + 1)^*$
$R_{21}^{(2)}$	$\emptyset + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*\emptyset$	$\emptyset$
$R_{22}^{(2)}$	$\epsilon + 0 + 1 + (\epsilon + 0 + 1)(\epsilon + 0 + 1)^*(\epsilon + 0 + 1)$	$(0 + 1)^*$

The final regular expression equivalent to DFAis constructed by taking the union of all the expressions where the first state is the start state and the second state is accepting.

With 1 as the start state and 2 as the only accepting state, we need only the expression $R_{12}^{(2)}$

$R_{12}^{(2)}$ = **1*0(0+1)***

Some Simplification Rules

$(\varepsilon+R)^* = R^*$

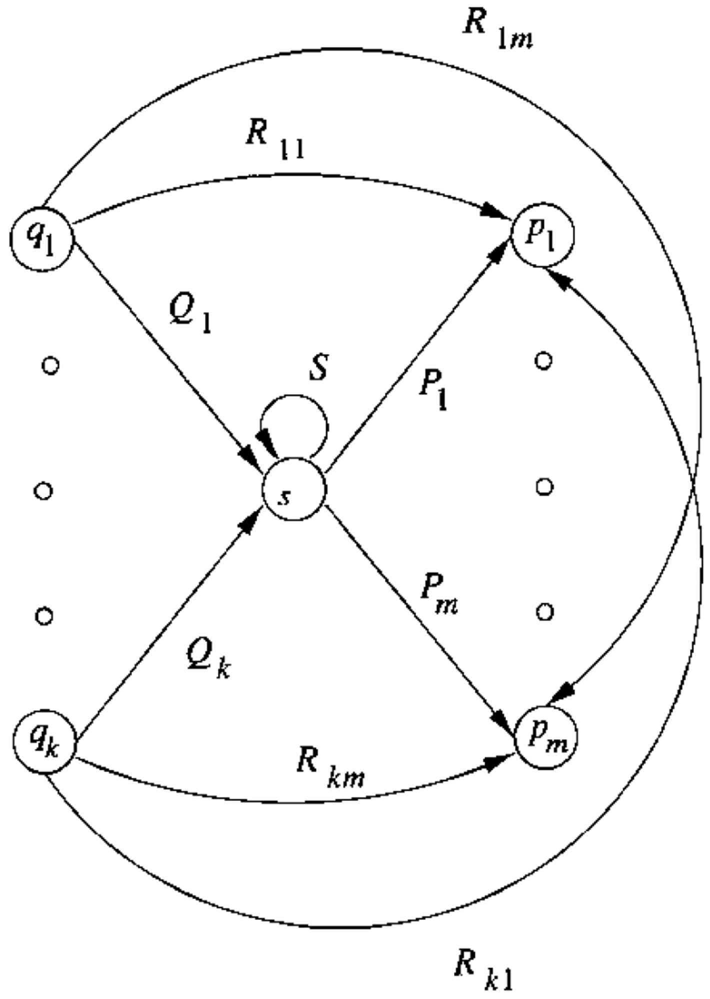
$\Phi R = R\Phi = \Phi$ Φ is an annihilator for concatenation.

$\Phi+R = R+\Phi = R$ Φ is the identity for union.

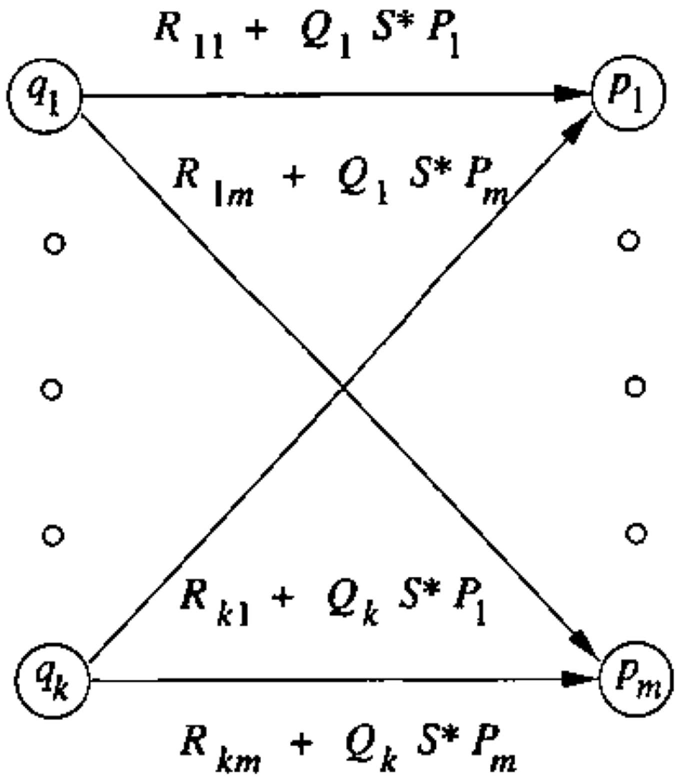
Converting DFA's to Regular Expressions by Eliminating States

- The previous method is expensive since we have to construct about n^3 expressions.
- There is more efficient way to convert DFA's to Regular Expressions by eliminating states.
- When we eliminate a state s. all the paths that went through s no longer exist in the automaton.
  - If the language of the automaton is not to change, we must include, on an arc that goes directly from q to p, the labels of paths that went from some state q to state p, through s.
  - Since the label of this arc may now involve strings, rather than single symbols, and there may even be an infinite number of such strings, we cannot simply list the strings as a label. Regular expressions are, finite way to represent all such strings.
  - Thus, automata will have regular expressions as labels.
  - The language of the automaton is the union over all paths from the start state to an accepting state of the language formed by concatenating the languages of the regular expressions along that path.

Converting DFA's to Regular Expressions by Eliminating States



➔ **Eliminate the state s**



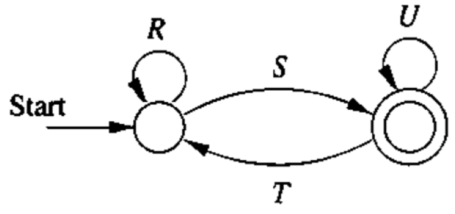
label the edges with regex's instead of symbols

Converting DFA's to Regular Expressions by Eliminating States

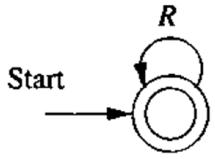
To construct a RegExp from a DFA

1. For each accepting state q, apply the above reduction process to produce an equivalent automaton with regular-expression labels on the arcs. Eliminate all states except q and the start state q_0.

2. If $q \neq q_0$, a two-state automaton will be created (CASE 1)



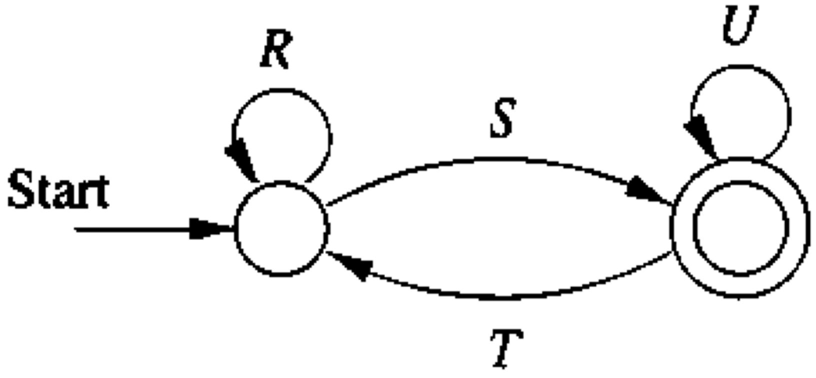
3. If $q = q_0$, a single-state automaton will be created (CASE 2)



4. The desired regular expression is the sum (union) of all the expressions derived from the reduced automata for each accepting state, by rules (2) and (3).

Converting DFA's to Regular Expressions by Eliminating States

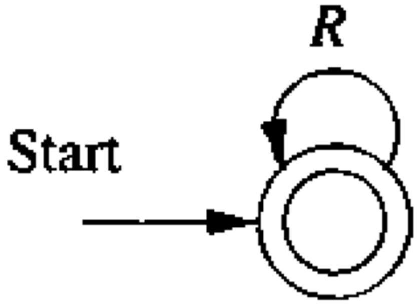
CASE 1: If $q \neq q_0$, a two-state automaton will be created



It accepts the regular expression:

(R+SU*T)*SU*

CASE 2: If $q = q_0$, a single-state automaton will be created

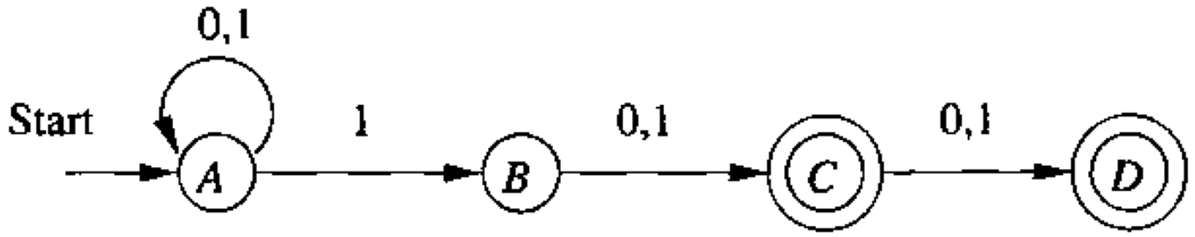


It accepts the regular expression:

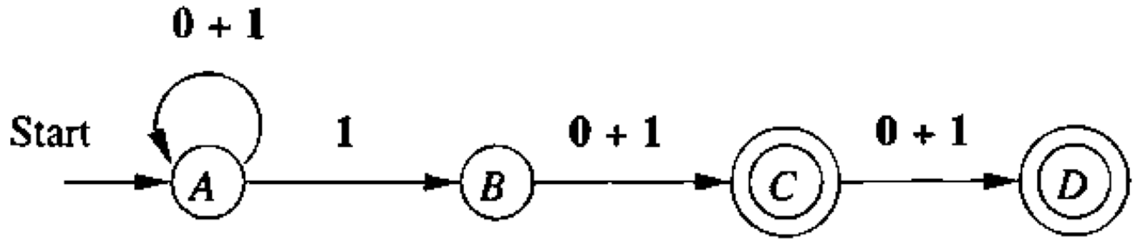
R*

Example

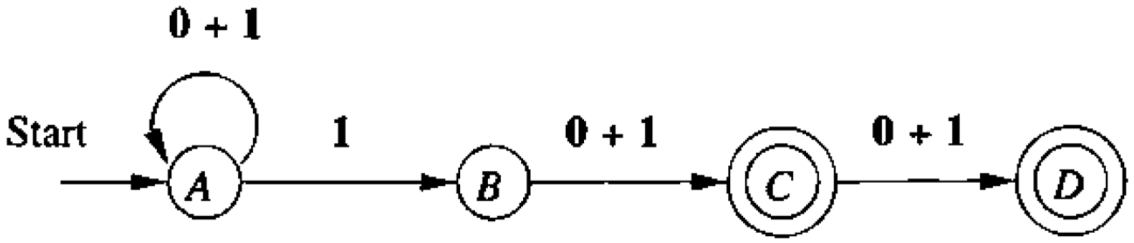
Convert a NFA to a regular expression



⇓ Replace all symbols on arcs with regular expressions

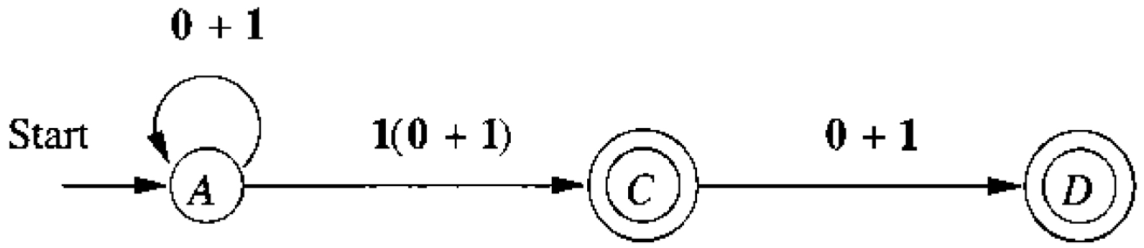


Example

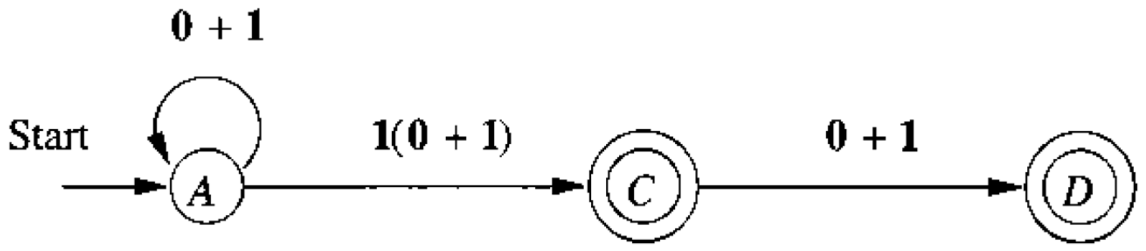


⇓ Eliminate the state B

$$\mathbf{NewArc}_{AC} = \mathbf{Arc}_{AC} + \mathbf{Arc}_{AB}\,\mathbf{Arc}_{BB}^{*}\,\mathbf{Arc}_{BC}$$
$$= \Phi + 1\,\Phi^{*}\,(0+1)$$
$$= 1\,(0+1)$$

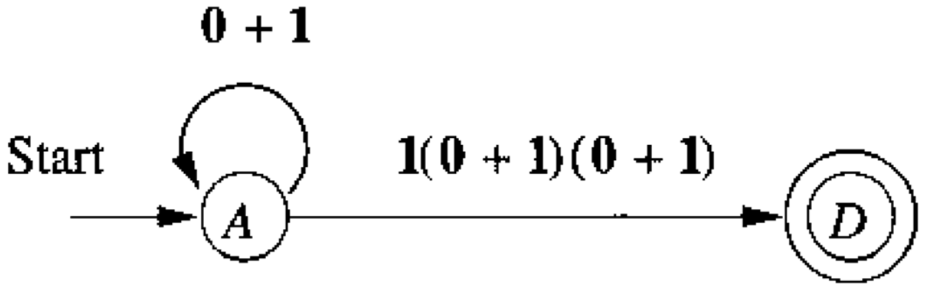


Example

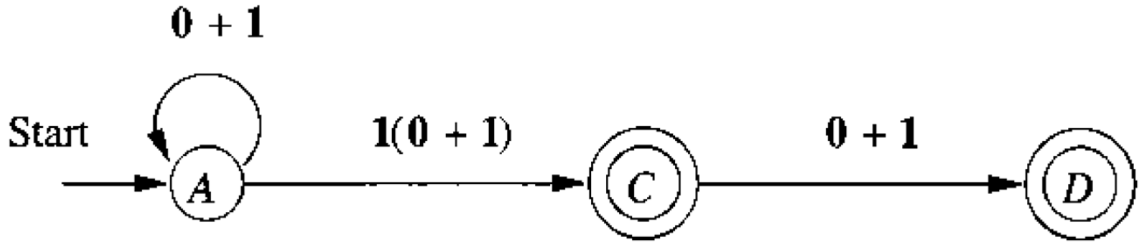


$\Downarrow$ Eliminate the state C

$$\begin{aligned}\mathbf{NewArc_{AD}} &= \mathbf{Arc_{AD} + Arc_{AC}\, Arc_{CC}{}^*\, Arc_{CD}} \\ &= \mathbf{\Phi + 1(0+1)\ \Phi^*\ (0+1)} \\ &= \mathbf{1\ (0+1)\ (0+1)}\end{aligned}$$

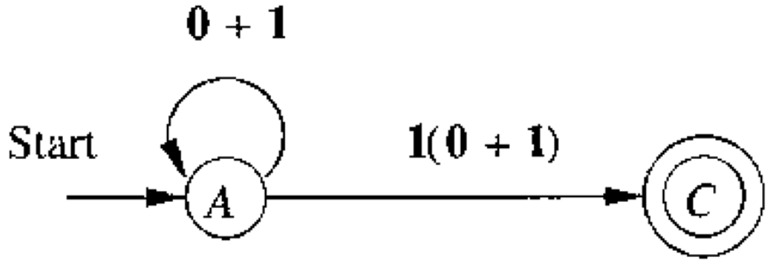


Example

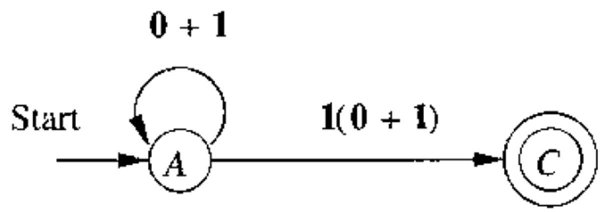


$\Downarrow$ Eliminate the state D

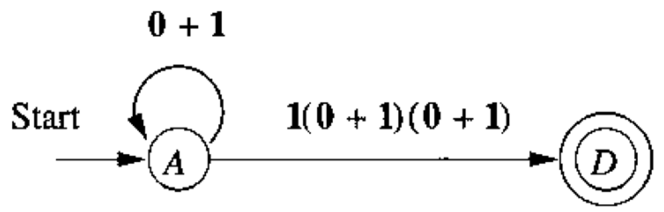
$$\begin{aligned} \mathbf{NewArc_{AC}} &= \mathbf{Arc_{AC} + Arc_{AD}\, Arc_{DD}^{*}\, Arc_{DC}} \\ &= \mathbf{1(0+1) + \Phi\, \Phi^{*}\, \Phi} \\ &= \mathbf{1\,(0+1)} \end{aligned}$$



Example - Result



$$RE = (Arc_{AA}+Arc_{AC}\ Arc_{CC}^*\ Arc_{CA})^* Arc_{AC} Arc_{CC}^*$$
$$= ((0+1)+1(0+1)\ \Phi^*\ \Phi)^*\ 1(0+1)\ \Phi^*$$
$$= (0+1)^*1(0+1)$$



$$RE = (Arc_{AA}+Arc_{AD}\ Arc_{DD}^*\ Arc_{DA})^* Arc_{AD} Arc_{DD}^*$$
$$= ((0+1)+1(0+1)\ (0+1)\Phi^*\ \Phi)^*\ 1(0+1)\ (0+1)\ \Phi^*$$
$$= (0+1)^*1(0+1)\ (0+1)$$

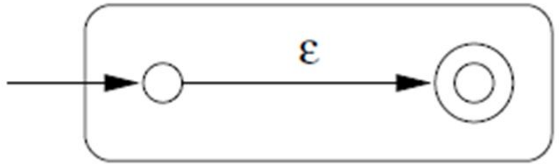
Final Reg Exp = $(0+1)^*1(0+1) + (0+1)^*1(0+1)\ (0+1)$

From Regular Expressions to ε-NFA's

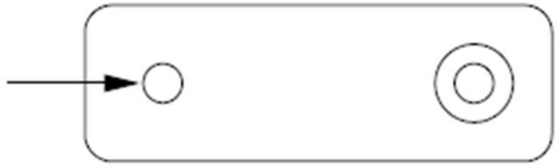
Theorem 3.7: For every regex R we can construct and ε-NFA A, s.t. L(A) = L(R).

Proof: By structural induction:

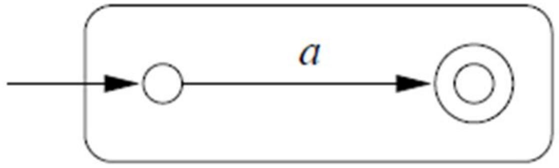
Basis: Automata for ϵ, $\emptyset$, and $\boldsymbol{a}$.



(a)

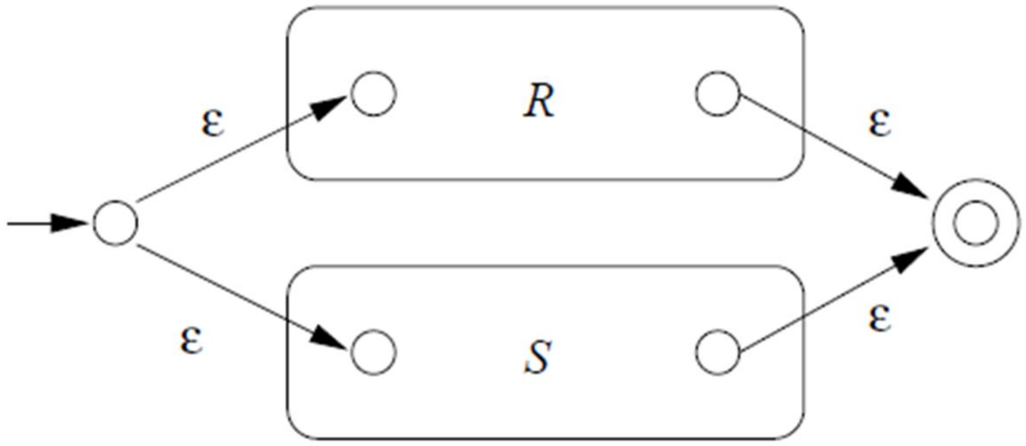


(b)

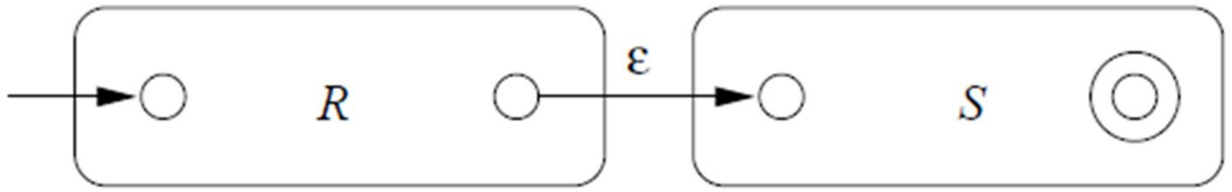


(c)

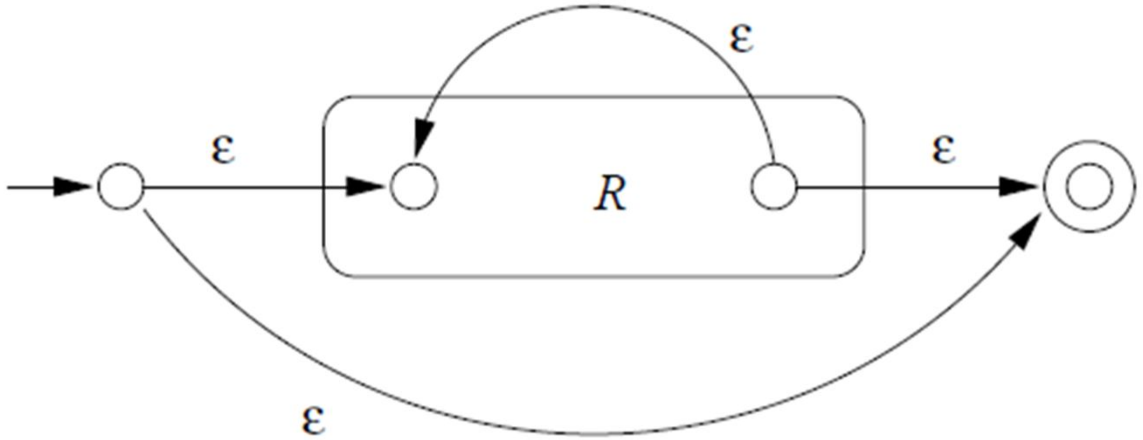
From Regular Expressions to ε-NFA's – R+S



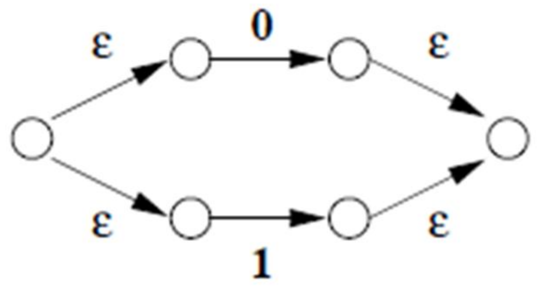
From Regular Expressions to ε-NFA's – RS



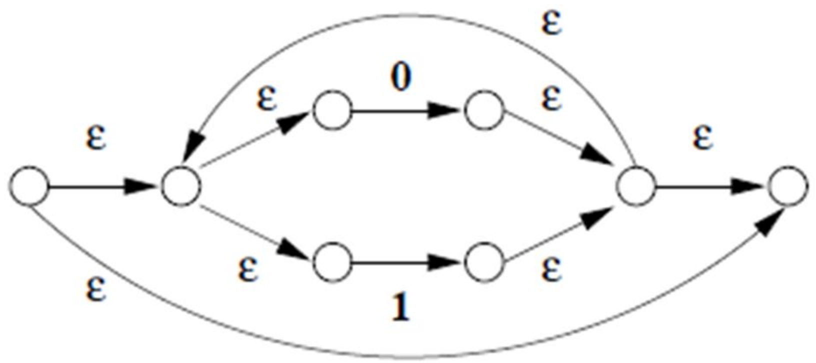
From Regular Expressions to ε-NFA's – R*



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

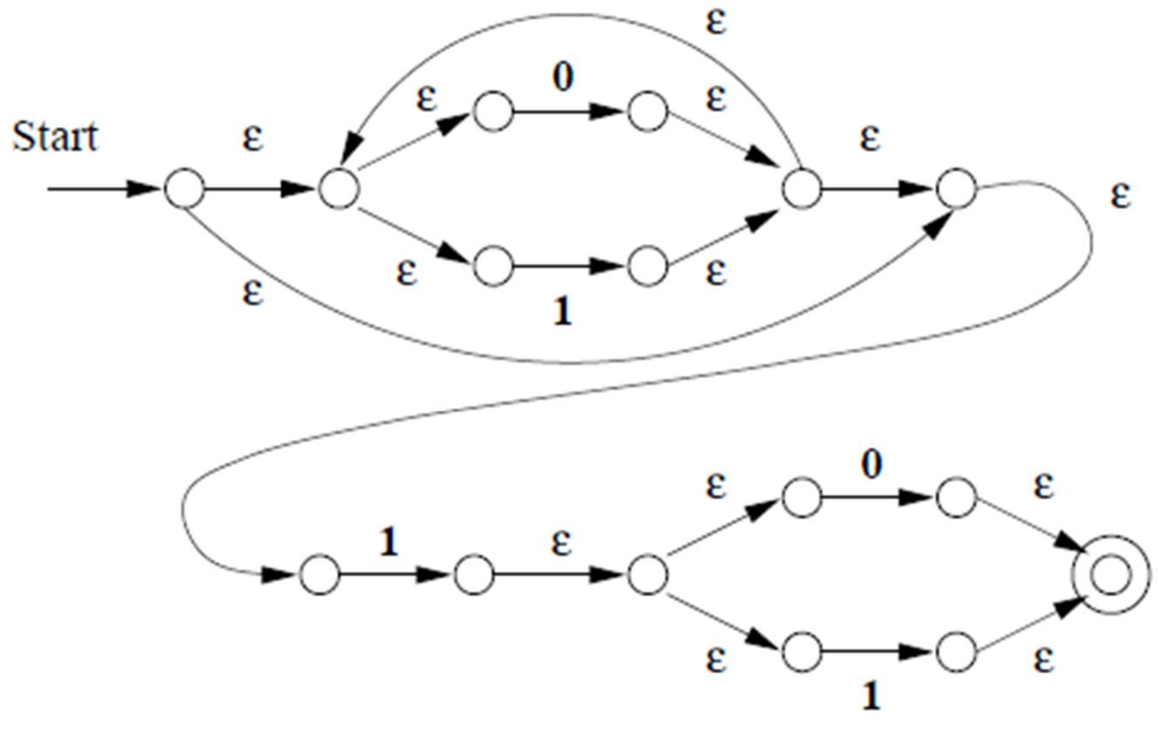


(a)



(b)

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



(c)

Algebraic Laws for Languages – Associativity and Commutativity

- **Commutativity** is the property of an operator that says we can switch the order of its operands and get the same result.
- **Associativity** is the property of an operator that allows us to regroup the operands when the operator is applied twice.

Union is commutative: $M \cup N = N \cup M$

Union is associative: $(M \cup N) \cup R = M \cup (N \cup R)$

Concatenation is associative: $(M\,N)\,R = M\,(N\,R)$

Concatenation is NOT commutative,
i.e., there are M and Nsuch that **MN ≠ NM**

Algebraic Laws for Languages – Identities and Annihilators

- An **identity** for an operator is a value such that when the operator is applied to the identity and some other value, the result is the other value.

- An **annihilator** for an operator is a value such that when the operator is applied to the annihilator and some other value, the result is the annihilator.

Φ is identity for union: $\Phi \cup N = N \cup \Phi = N$

{ε} is left and right identity for concatenation: $\{\varepsilon\} N = N \{\varepsilon\} = N$

Φ is left and right annihilator for concatenation: $\Phi N = N \Phi = \Phi$

Algebraic Laws for Languages – Distributive and Idempotent

- A **distributive law** involves two operators, and asserts that one operator can be pushed down to be applied to each argument of the other operator individually.

Concatenation is left and right distributive over union:

$$R(M \cup N) = RM \cup RN$$
$$(M \cup N)R = MR \cup NR$$

- An operator is said to be **idempotent** if the result of applying it to two of the same values as arguments is that value.

Union is idempotent: $M \cup M = M$

Algebraic Laws for Languages – Closure Laws

Languages	**Regular Expressions**
$\Phi^* = \{\varepsilon\}$	$\Phi^* = \varepsilon$
$\{\varepsilon\}^* = \{\varepsilon\}$	$\varepsilon^* = \varepsilon$
$L^+ = LL^* = L^*L$	$R^+ = RR^* = R^*R$
$L^* = L^+ \cup \{\varepsilon\}$	$R^* = R^+ + \varepsilon$
$L? = L \cup \{\varepsilon\}$	$R? = R + \varepsilon$
$(L^*)^* = L^*$	$(R^*)^* = R^*$

Algebraic Laws for Languages

Theorem: (L*)* = L*

Proof:

$$
\begin{aligned}
w \in (L^*)^* &\iff w \in \bigcup_{i=0}^{\infty}\left(\bigcup_{j=0}^{\infty} L^j\right)^i \\
&\iff \exists k, m \in \mathsf{N} : w \in (L^m)^k \\
&\iff \exists p \in \mathsf{N} : w \in L^p \\
&\iff w \in \bigcup_{i=0}^{\infty} L^i \\
&\iff w \in L^* \qquad \Box
\end{aligned}
$$

Discovering Laws for Regular Expressions

- There is an infinite variety of laws about regular expressions that might be proposed.

- Is there a general methodology that will make our proofs of the correct laws easy? ➔ YES
  - This methodology only works for regular expression operators (concetanation, or, closure)

- **Methodology**: Exp1 = Exp2
  - Replace each variable in the law (in Exp1 and Exp2) with unique symbols to create concrete regular expressions, RE1 and RE2.
  - Check the equality of the languages of RE1 and RE2, ie. L(RE1) = L(RE2)

Discovering Laws for Regular Expressions

Theorem 3.13: Let E be a regular expression with variables $L_1, L_2, \ldots, L_m$. Form concrete regular expression C by replacing each occurrence of L_i by the symbol a_i, for $i = 1, 2, \ldots, m$. Then for any languages $L_1, L_2, \ldots, L_m$, every string w in $L(E)$ can be written $w = w_1 w_2 \cdots w_k$, where each w_i is in one of the languages, say L_{j_i}, and the string $a_{j_1} a_{j_2} \cdots a_{j_k}$ is in the language $L(C)$. Less formally, we can construct $L(E)$ by starting with each string in $L(C)$, say $a_{j_1} a_{j_2} \cdots a_{j_k}$, and substituting for each of the a_{j_i}'s any string from the corresponding language L_{j_i}.

PROOF: The proof is a structural induction on the expression E.

Discovering Laws for Regular Expressions - Example

Law: R(M+N) = RM + RN

Replace R with a, M with b, and N with c.

➔ **a(b+c) = ab + ac**

Then, check whether **L(a(b+c))** is equal to **L(ab+bc)**

If their languages are equal, the law is TRUE.

Since, **L(a(b+c))** is equal to **L(ab+bc)**
➔ **R(M+N) = RM + RN** is a true law

Discovering Laws for Regular Expressions - Example

Law: (M+N)* = (M*N*)*

Replace M with a, and N with b.

➔ **(a+b)* = (a*b*)***

Then, check whether **L((a+b)*)** is equal to **L((a*b*)*)**

Since, **L((a+b)*)** is equal to **L((a*b*)*)**
➔ **(M+N)* = (M*N*)*** is a true law