Bab 3 Regular Expressions

September 2021

Definitions

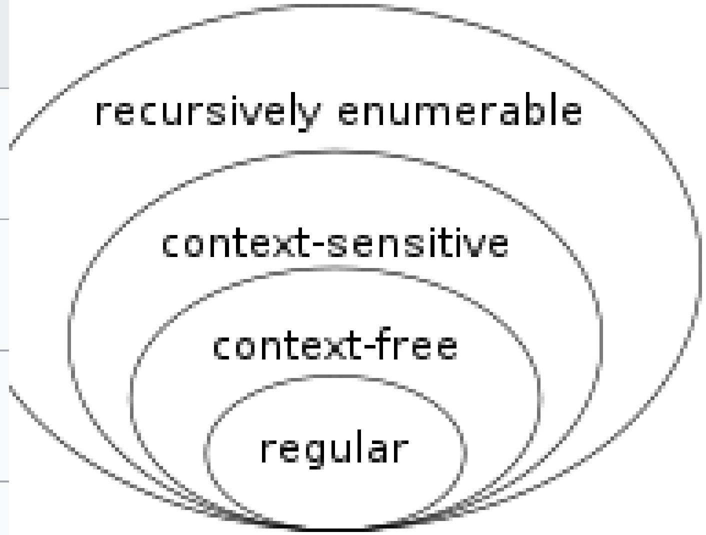
Equivalence to Finite Automata

Informatika

Chomsky Hierarchy

Grammar	Languages	Automaton
Type-0	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine
Type-2	Context-free	Non-deterministic pushdown automaton
Type-3	Regular	Finite state automaton

Regular expression



Chomsky Hierarchy

RE's: Introduction

- ***Regular expressions*** **are an algebraic way to describe languages.**
- **They describe exactly the regular languages.**
- **If E is a regular expression, then L(E) is the language it defines.**
- **We'll describe RE's and their languages recursively.**

Examples: RE's

RE

L

- L(01) = {01}.
- L(01+0) = {01, 0}.
- L(0(1+0)) = {01, 00}.
  - Note order of precedence of operators.
- L(0*) = {ϵ, 0, 00, 000,… }.
- L((0+10)*(ϵ+1)) = all strings of 0's and 1's without two consecutive 1's.

RE's: Definition

- **Basis 1: If *a* is any symbol, then a is a RE, and L(a) = {a}.**
  - Note: {a} is the language containing one string, and that string is of length 1.
- **Basis 2: ϵ is a RE, and L(ϵ) = {ϵ}.**
- **Basis 3: ∅ is a RE, and L(∅) = ∅.**

RE's: Definition – (2)

- **Induction 1: If E_1 and E_2 are regular expressions, then E_1+E_2 is a regular expression, and $L(E_1+E_2) = L(E_1)\cup L(E_2)$.**
- **Induction 2: If E_1 and E_2 are regular expressions, then E_1E_2 is a regular expression, and $L(E_1E_2) = L(E_1)L(E_2)$.**

Concatenation : the set of strings wx such that w
Is in $L(E_1)$ and x is in $L(E_2)$.

RE's: Definition – (3)

- **Induction 3: If E is a RE, then E* is a RE, and L(E*) = (L(E))*.**

Closure, or "Kleene closure" = set of strings $w_1w_2\ldots w_n$, for some $n \geq 0$, where each w_i is in L(E).
Note: when n=0, the string is ϵ.

Precedence of Operators

- Parentheses may be used wherever needed to influence the grouping of operators.
- Order of precedence is * (highest), then concatenation, then + (lowest).

Contoh-contoh RE

- all strings that begin with **a** and end with **b**

  $a\ (a + b)^*\ b$

- all non empty strings of even length

  $(aa + ab + ba + bb)^+$

- all strings with at least one **a**

  $(a + b)^*\ a\ (a + b)^*$

- all strings with at least two **a**'s

  $(a + b)^*\ a\ (a + b)^*\ a\ (a + b)^*$

- All strings with at least two **b**'s.

  $(a + b)^* \, b \, (a + b)^* \, b \, (a + b)^*$

- All strings with exactly two **b**'s.

  $a^* \, b \, a^* \, b \, a^*$

- All strings with at least one **a** and at least one **b**.

  $(a + b)^* \, (ab + ba) \, (a + b)^*$

- All strings which end in a double letter (two **a**'s or two **b**'s).

  $(a + b)^* \, (aa + bb)$

Latihan

- **The set of strings over alphabet {a,b} containing at least one a and at least one b**
- **The set of strings of a and b whose 4^{th} symbol from the right end is a**
- **The set of strings of a and b with at most one pair of consecutive a**

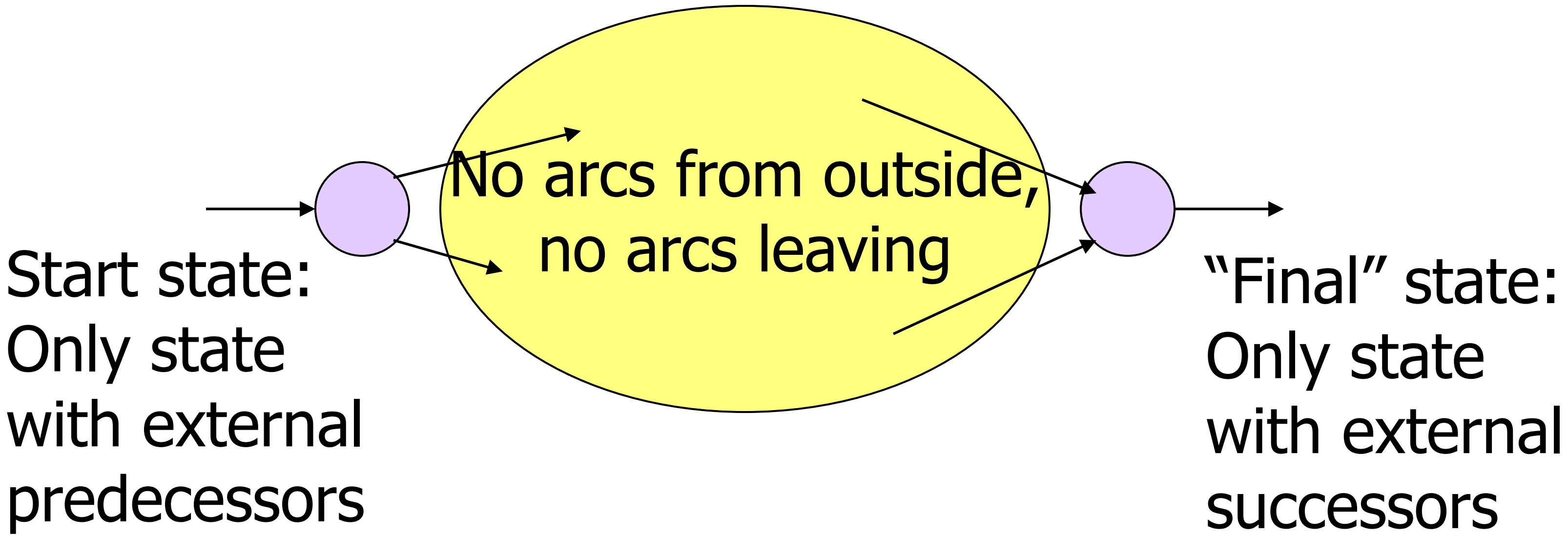
Equivalence of RE's and Automata

- **We need to show that for every RE, there is an automaton that accepts the same language.**
  - Pick the most powerful automaton type: the ϵ-NFA.
- **And we need to show that for every automaton, there is a RE defining its language.**
  - Pick the most restrictive type: the DFA.

Converting a RE to an ϵ-NFA

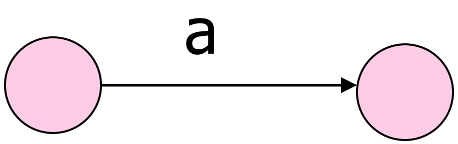
- **Proof is an induction on the number of operators (+, concatenation, *) in the RE.**
- **We always construct an automaton of a special form (next slide).**

Form of ϵ-NFA's Constructed

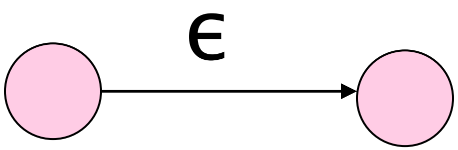


RE to ϵ-NFA: Basis

- **Symbol a:**

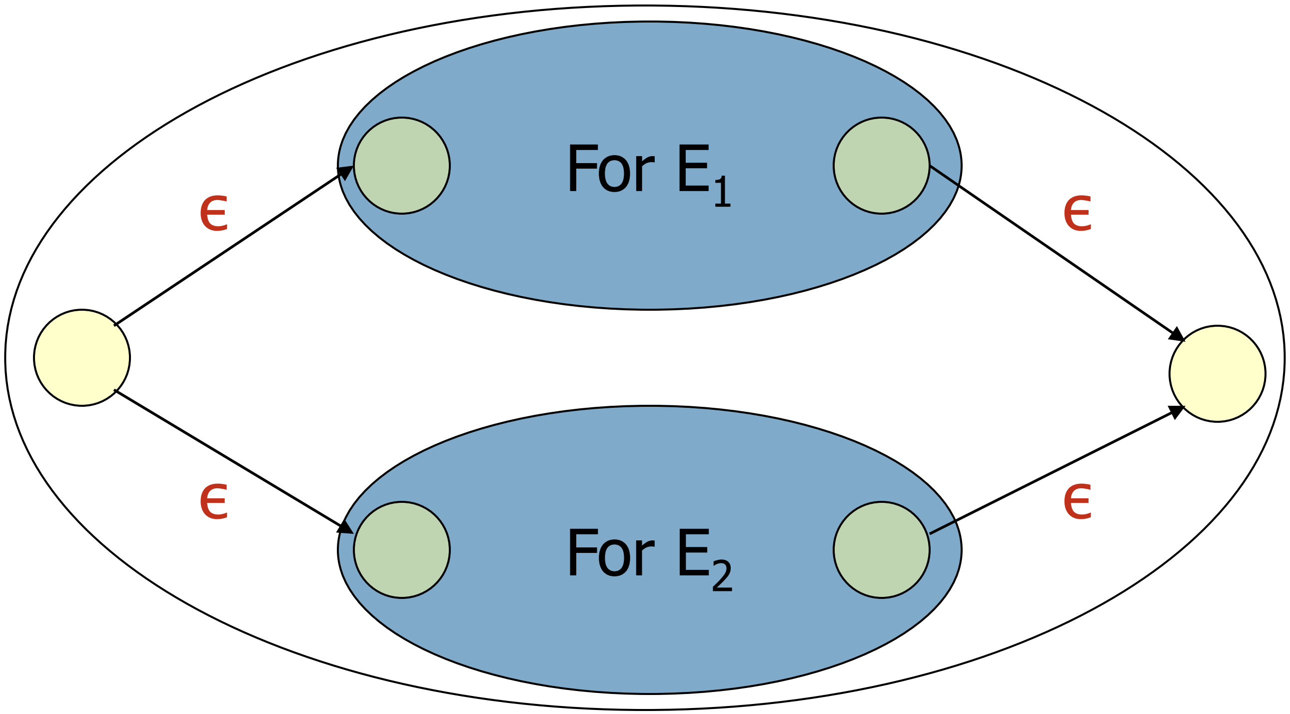


- **ϵ:**



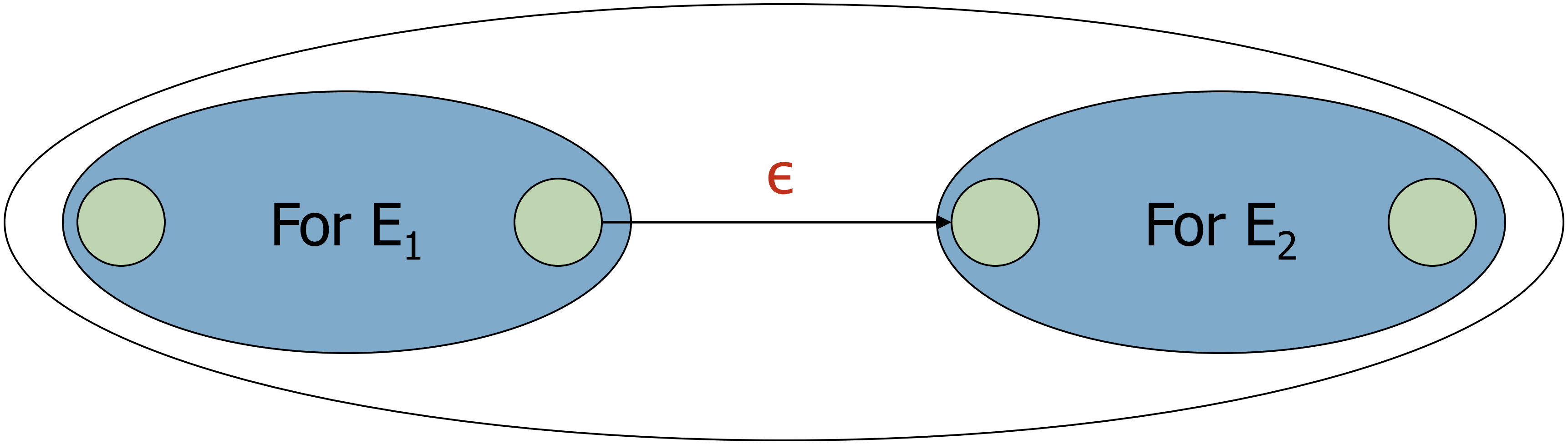
- **∅:**

RE to ϵ-NFA: Induction 1 – Union



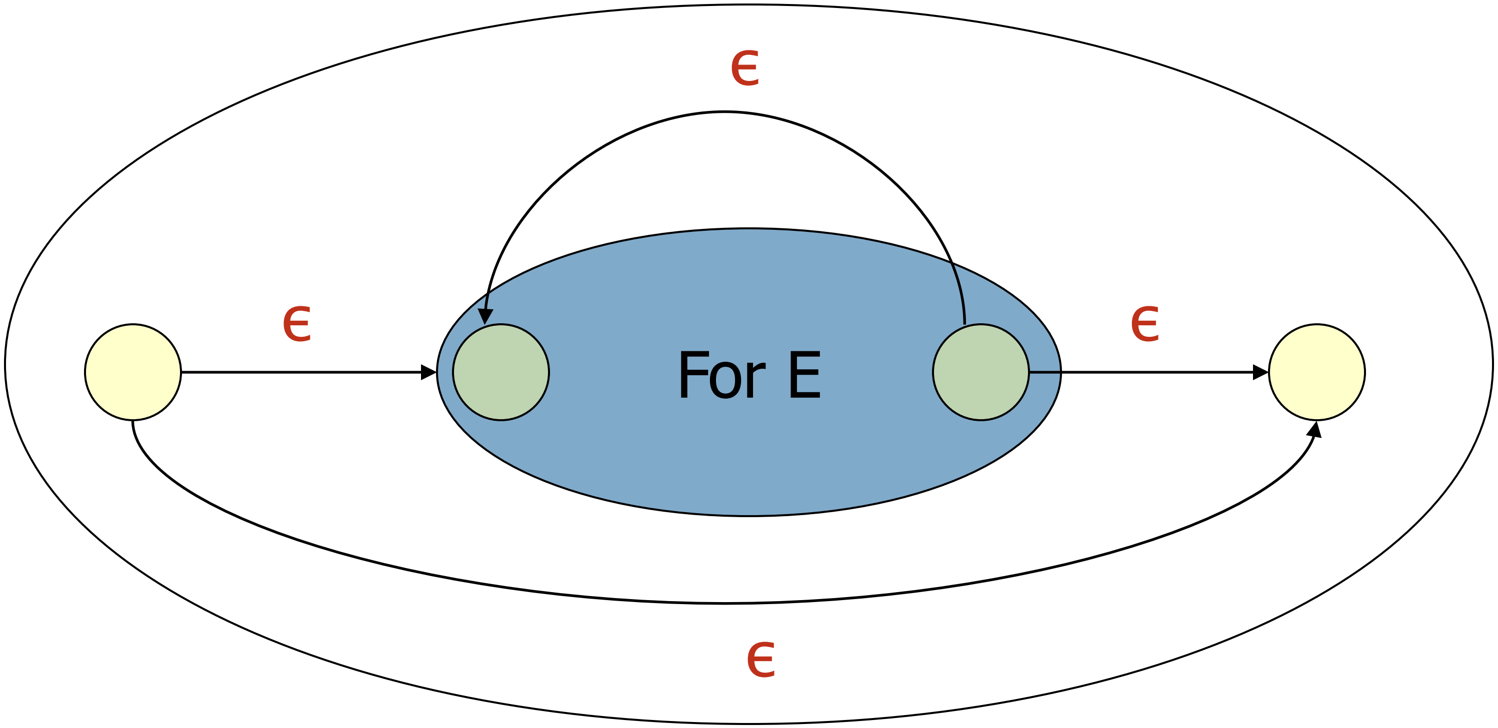
For $E_1 \cup E_2$

RE to ϵ-NFA: Induction 2 – Concatenation



For E_1E_2

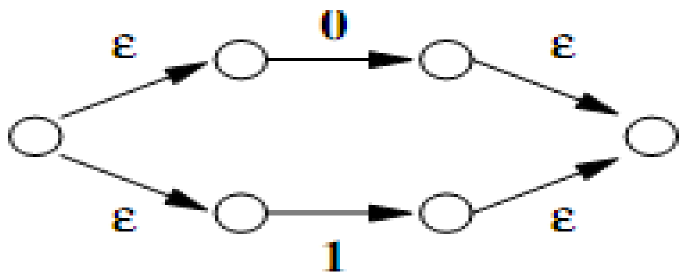
RE to ϵ-NFA: Induction 3 – Closure



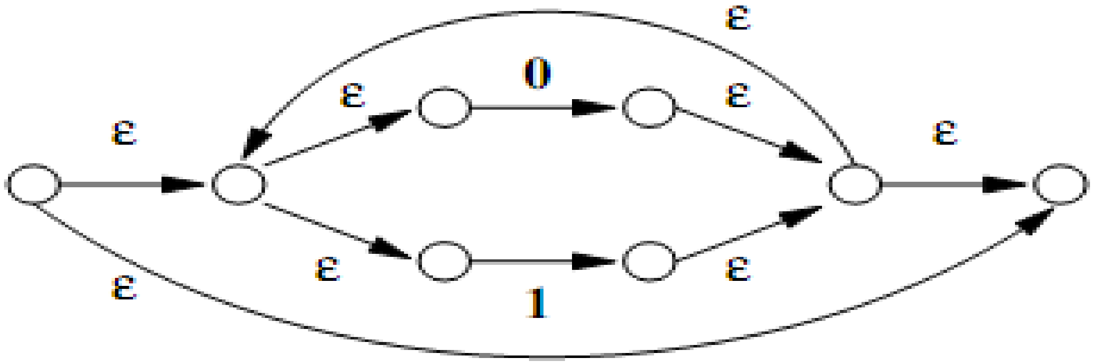
For E*

Contoh RE –NFA epsilon

We convert $(0+1)^*1(0+1)$

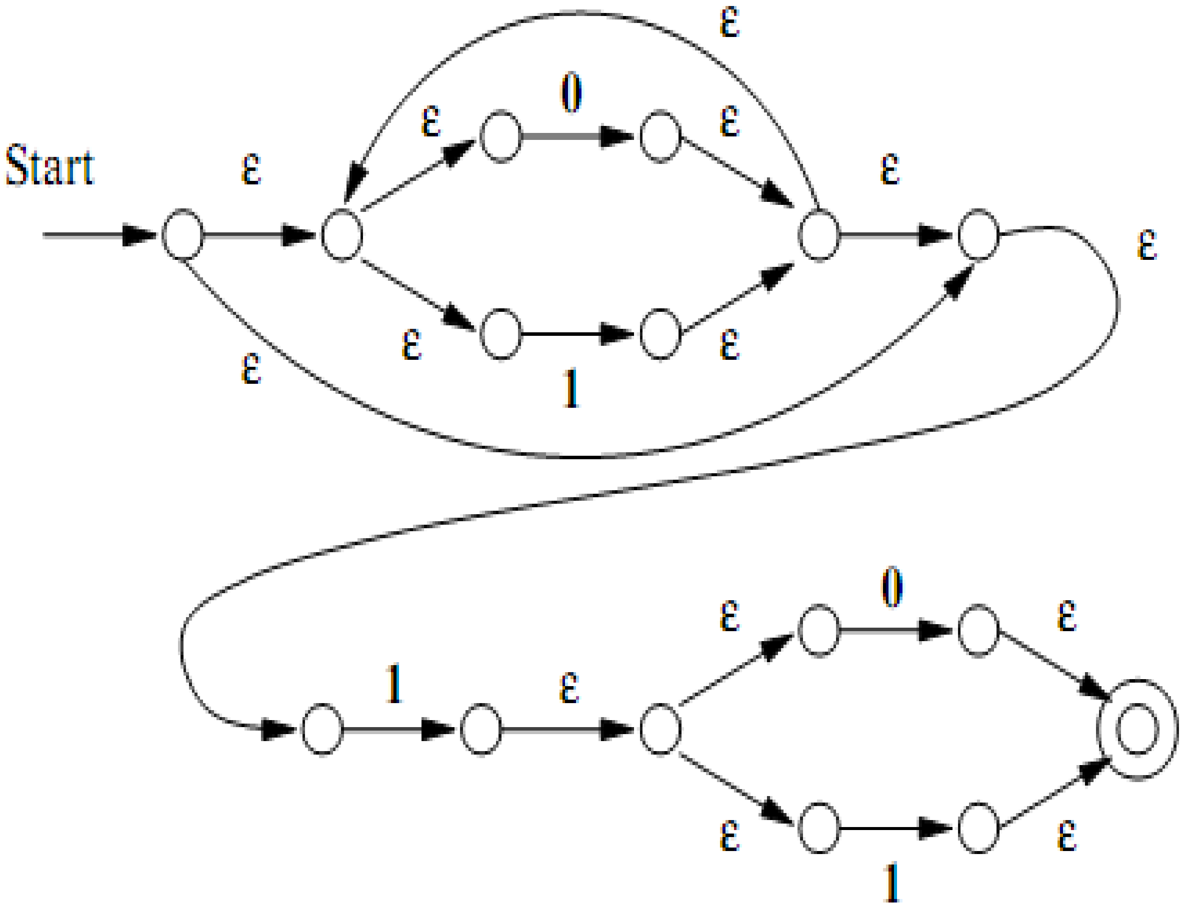


(a)



(b)

(0+1)* 1 (0+1)



(c)

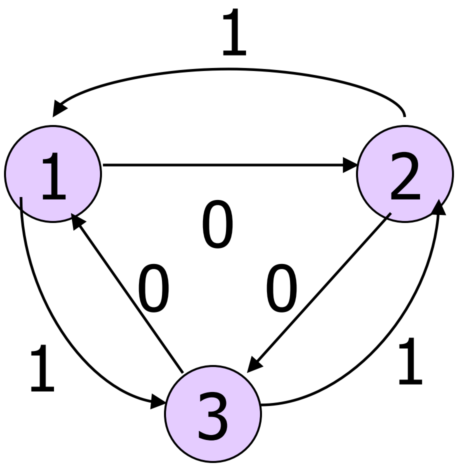
DFA-to-RE

- **A strange sort of induction.**
- **States of the DFA are assumed to be 1,2,…,n.**
- **We construct RE's for the labels of restricted sets of paths.**
  - Basis: single arcs or no arc at all.
  - Induction: paths that are allowed to traverse next state in order.

k-Paths

- **A k-path is a path through the graph of the DFA that goes though no intermediate state numbered higher than k.**
- **Endpoints are not restricted; they can be any state.**

Example: k-Paths



0-paths from 2 to 3:
RE for labels = **0**.

1-paths from 2 to 3:
RE for labels = **0**+**11**.

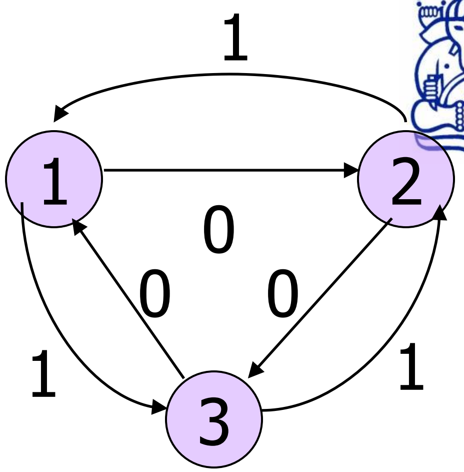
2-paths from 2 to 3:
RE for labels =
(**10**)***0**+**1**(**01**)***1**

3-paths from 2 to 3:
RE for labels = ??

k-Path Induction

- **Let R_{ij}^{k} be the regular expression for the set of labels of k-paths from state i to state j.**
- **Basis: k=0. R_{ij}^{0} = sum of labels of arc from i to j.**
  - ∅ if no such arc.
  - But add ϵ if i=j.

Example: Basis



- $R_{12}^{0} = 0$.
- $R_{11}^{0} = \emptyset + \epsilon = \epsilon$.

k-Path Inductive Case

- **A k-path from i to j either:**
  1. Never goes through state k, or
  2. Goes through k one or more times.

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

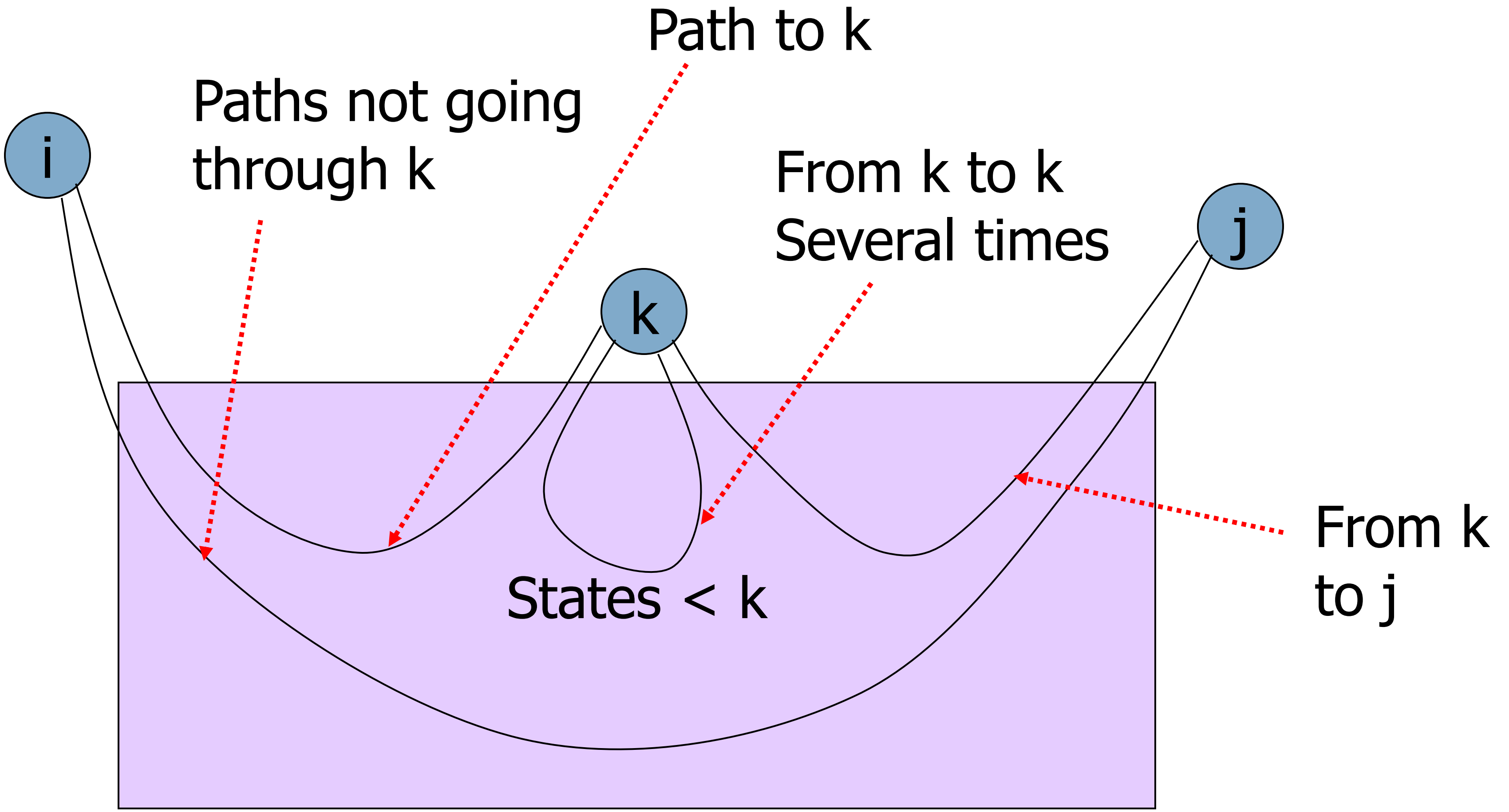
Doesn't go through k

Goes from i to k the first time

Zero or more times from k to k

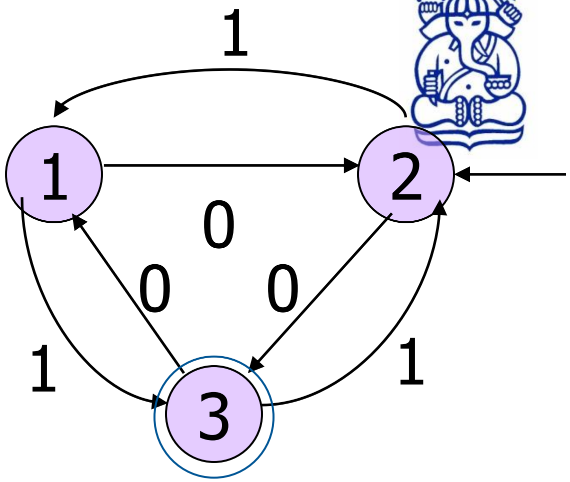
Then, from k to j

Illustration of Induction



Final Step

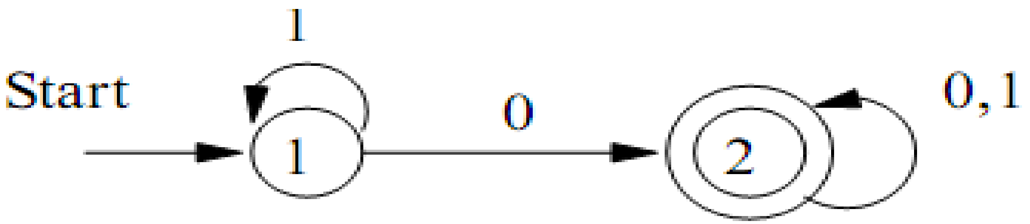
- **The RE with the same language as the DFA is the sum (union) of R_{ij}^{n}, where:**
  1. n is the number of states; i.e., paths are unconstrained.
  2. i is the start state.
  3. j is one of the final states.



Example

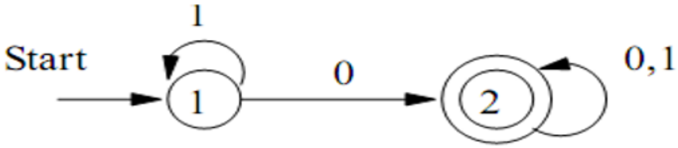
- $R_{23}^3 = R_{23}^2 + R_{23}^2(R_{33}^2)^*R_{33}^2 = R_{23}^2(R_{33}^2)^*$
- $R_{23}^2 = (10)^*0+1(01)^*1$
- $R_{33}^2 = 0(01)^*(1+00) + 1(10)^*(0+11)$
- $R_{23}^3 = [(10)^*0+1(01)^*1]\,[(0(01)^*(1+00) + 1(10)^*(0+11))]^*$

Let's find R for A, where
$L(A) = \{x0y : x \in \{1\}^* \text{ and } y \in \{0,1\}^*\}$



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

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



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

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

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

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

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

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

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

The final regex for A is

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

Observations

There are n^3 expressions $R_{ij}^{(k)}$

Each inductive step grows the expression 4-fold

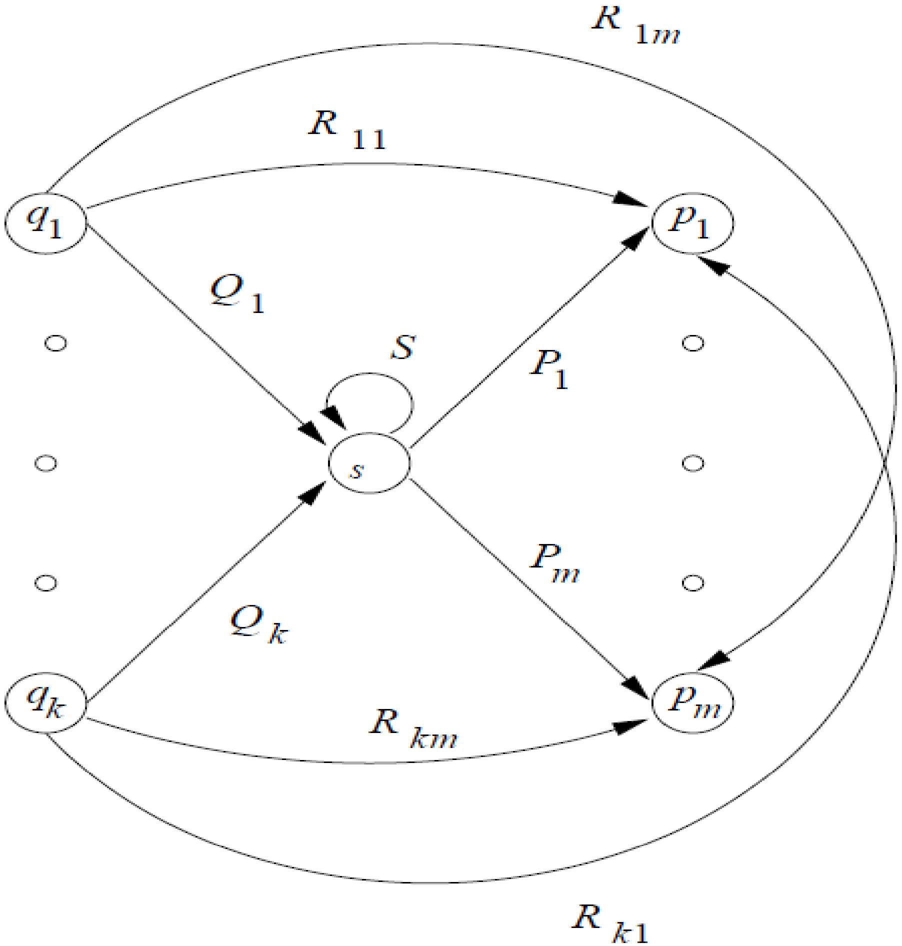
$R_{ij}^{(n)}$ could have size 4^n

For all $\{i, j\} \subseteq \{1, \ldots, n\}$, $R_{ij}^{(k)}$ uses $R_{kk}^{(k-1)}$
so we have to write n^2 times the regex $R_{kk}^{(k-1)}$

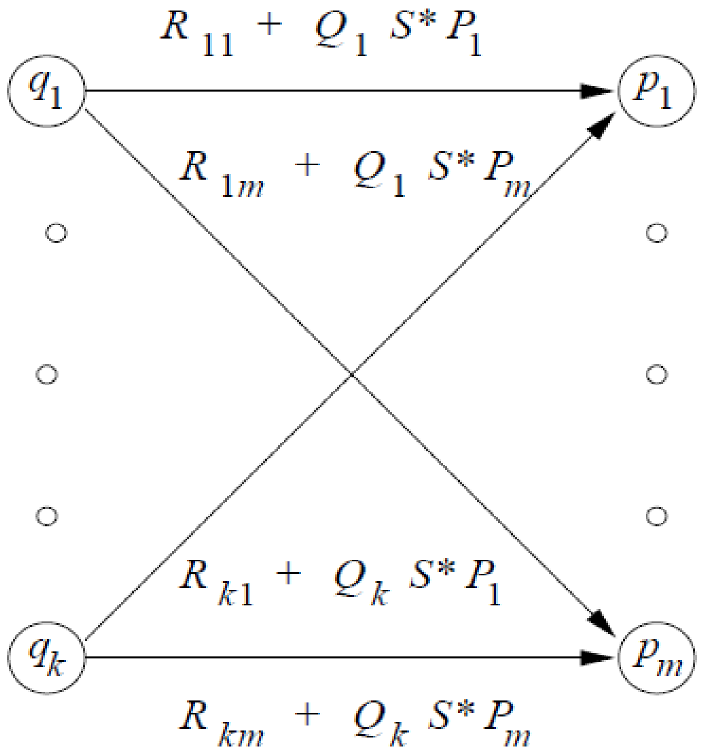
We need a more efficient approach:
the state elimination technique

The state elimination technique

Let's label the edges with regex's instead of symbols

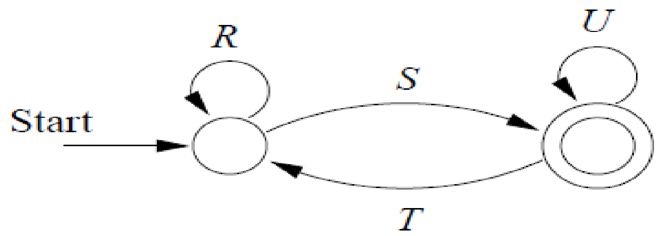


Now, let's eliminate state s.



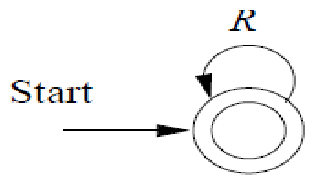
For each accepting state q eliminate from the original automaton all states exept q_0 and q.

For each $q \in F$ we'll be left with an A_q that looks like



that corresponds to the regex $E_q = (R + SU^*T)^*SU^*$

or with A_q looking like

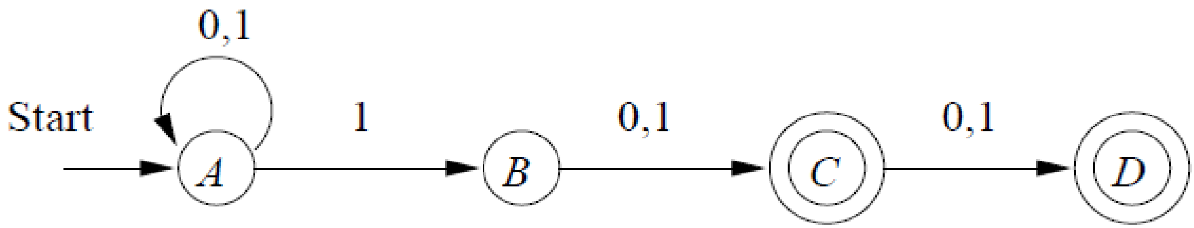


corresponding to the regex $E_q = R^*$

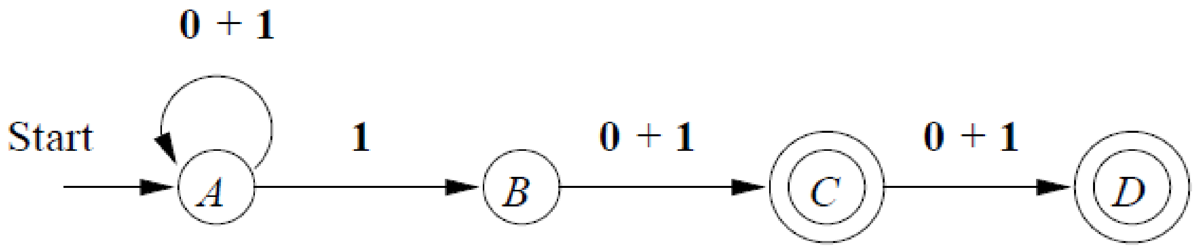
- The final expression is

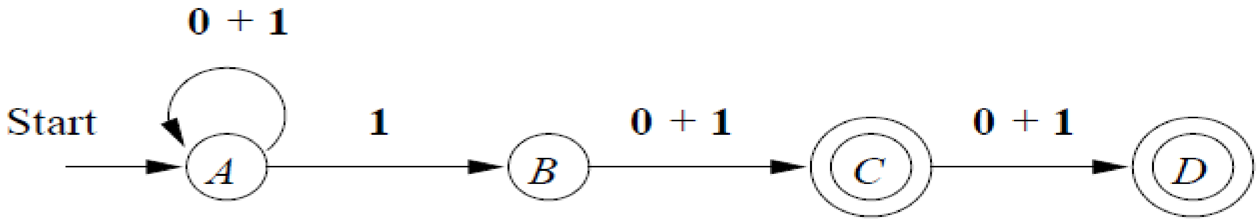
$$\bigoplus_{q \in F} E_q$$

Example: $\mathcal{A}$, where $L(\mathcal{A}) = \{W : w = x1b, \text{ or } w = x1bc, \ \ x \in \{0,1\}^*, \{b,c\} \subseteq \{0,1\}\}$

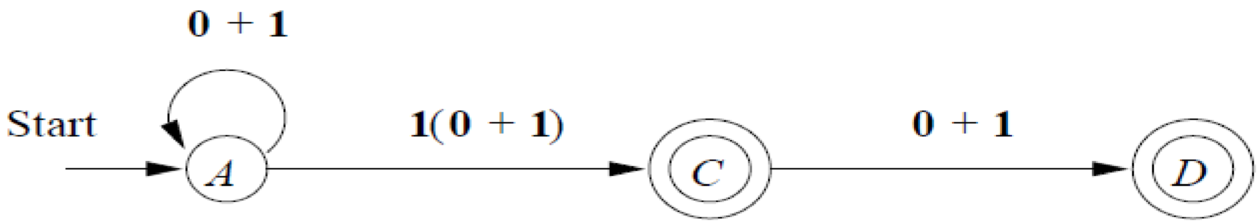


We turn this into an automaton with regex labels

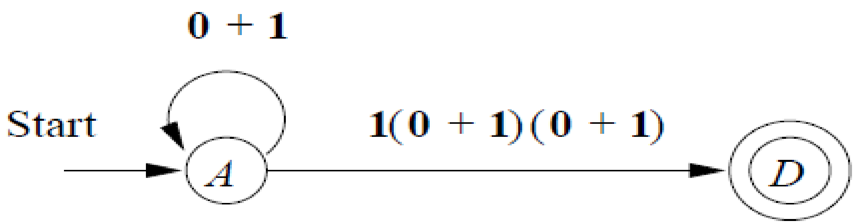




Let's eliminate state B

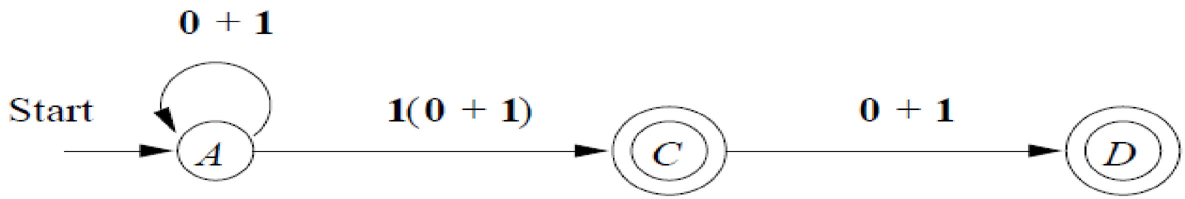


Then we eliminate state C and obtain $\mathcal{A}_D$

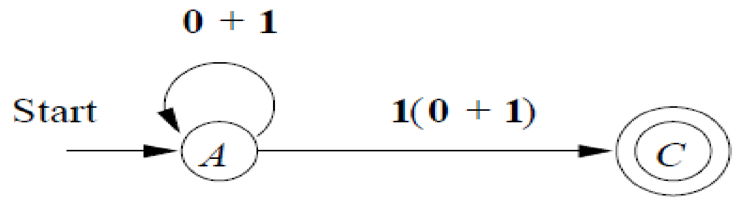


with regex $(0 + 1)^*1(0 + 1)(0 + 1)$

From



we can eliminate D to obtain $\mathcal{A}_C$

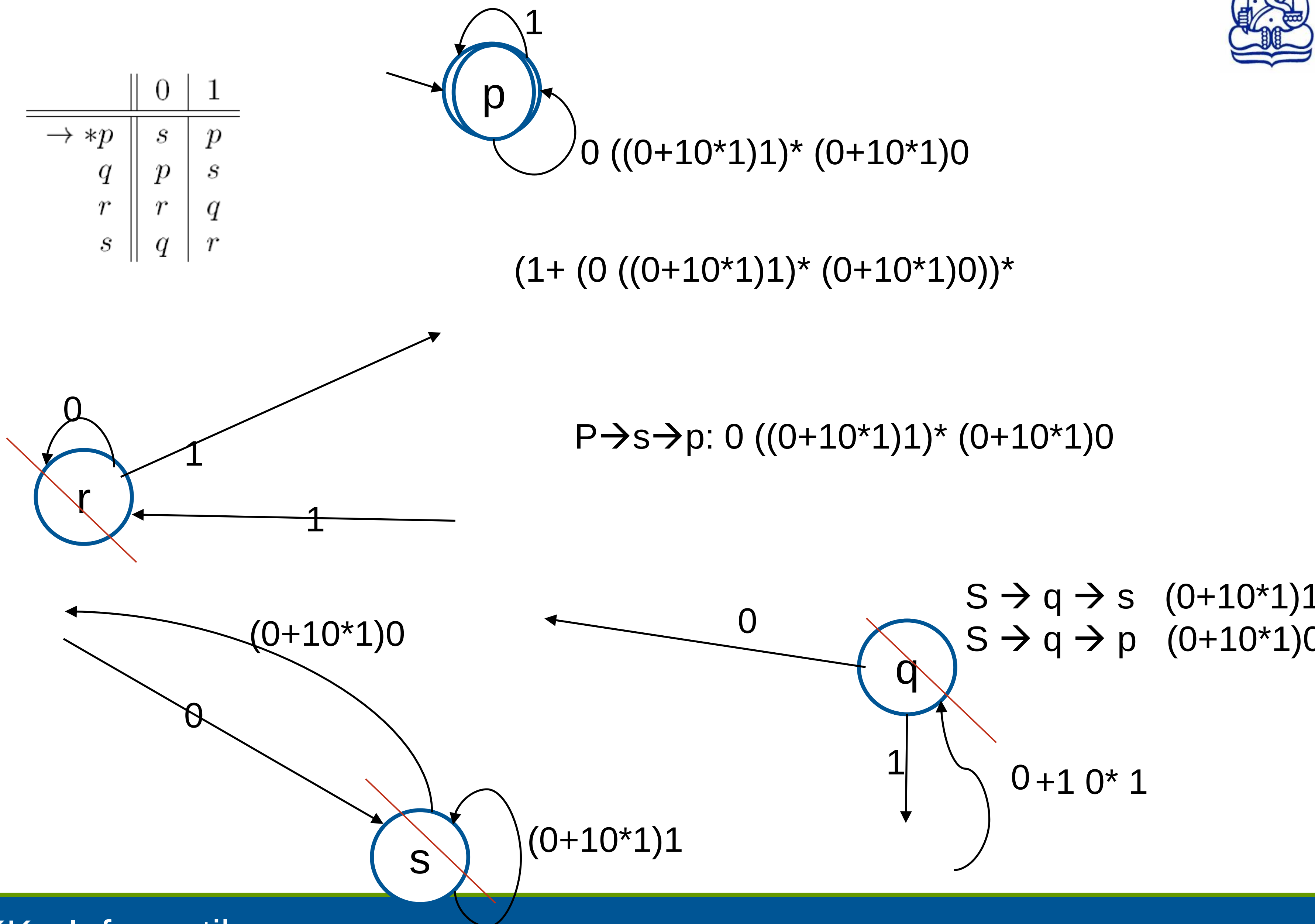


with regex $(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$

- The final expression is the sum of the previous two regex's:

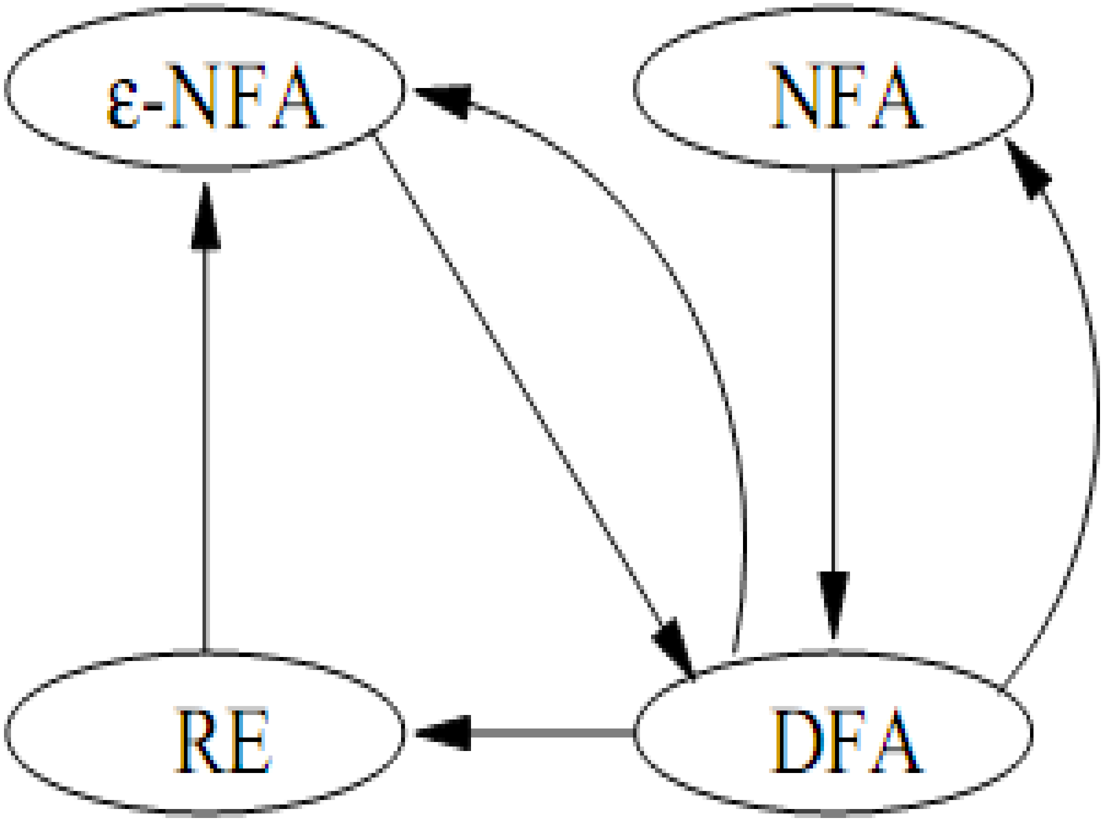
$$(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})(\mathbf{0}+\mathbf{1})+(\mathbf{0}+\mathbf{1})^*\mathbf{1}(\mathbf{0}+\mathbf{1})$$

	0	1
$\rightarrow *p$	s	p
q	p	s
r	r	q
s	q	r



Summary

- **Each of the three types of automata (DFA, NFA, ϵ-NFA) we discussed, and regular expressions as well, define exactly the same set of languages: the regular languages.**


ε-NFA
NFA
RE
DFA

Algebraic Laws for RE's

- **Union and concatenation behave sort of like addition and multiplication.**
  - + is commutative and associative; concatenation is associative.
  - Concatenation distributes over +.
  - Exception: Concatenation is not commutative.

Identities and Annihilators

- **$\emptyset$ is the identity for +.**
  - $R + \emptyset = R$.
- **ϵ is the identity for concatenation.**
  - $\epsilon R = R\epsilon = R$.
- **$\emptyset$ is the annihilator for concatenation.**
  - $\emptyset R = R\emptyset = \emptyset$.

- $L \cup M = M \cup L$.

Union is *commutative*.

- $(L \cup M) \cup N = L \cup (M \cup N)$.

Union is *associative*.

- $(LM)N = L(MN)$.

Concatenation is *associative*

Note: Concatenation is not commutative, *i.e.*, there are L and M such that $LM \neq ML$.

- $\emptyset \cup L = L \cup \emptyset = L$.

$\emptyset$ is *identity* for union.

- $\{\epsilon\}L = L\{\epsilon\} = L$.

$\{\epsilon\}$ is *left* and *right identity* for concatenation.

- $\emptyset L = L\emptyset = \emptyset$.

$\emptyset$ is *left* and *right annihilator* for concatenation.

- $L(M \cup N) = LM \cup LN$.

Concatenation is *left distributive* over union.

- $(M \cup N)L = ML \cup NL$.

Concatenation is *right distributive* over union.

- $L \cup L = L$.

Union is *idempotent*.

- $\emptyset^* = \{\epsilon\}$, $\{\epsilon\}^* = \{\epsilon\}$.

- $L^+ = LL^* = L^*L$, $L^* = L^+ \cup \{\epsilon\}$

- $(L^*)^* = L^*$. Closure is *idempotent*