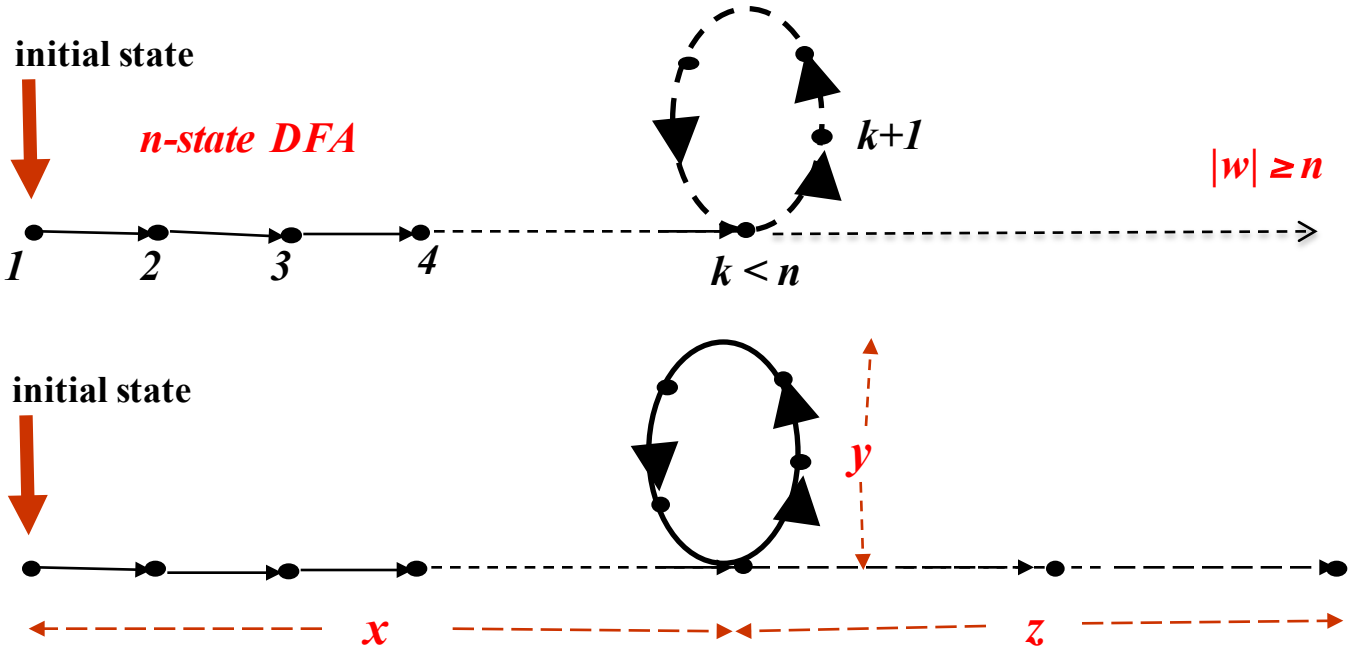
Pumping Lemma



Let ***L*** *be regular language then there exists an integer* ***n*** *(some* ***n****-state DFA must accept* ***L*** *!) such that for* ***any*** *string* ***w*** *with* $|w| \geq n$ *there are strings* ***x, y,*** *and* ***z*** *where* $w=xyz$ *with :*

(1) $|xy| \leq n$, (2) $|y| \geq 1$, (3) $xy^iz \in L \ \forall i \geq 0$

Proving languages non-regular !

Example 1

$L = (s \in \{0,1\}^* \mid s = 0^k.1^k \, , \, k \geq 0)$ *is non-regular!*

*Assume the contrary (L is regular !) and apply the pumping lemma (**PL**)*

Let n *(# states of a DFA accepting* L*) be as* ***given*** *in **PL** and*

choose $w = 0^n 1^n \in L$ *then* $|w| = 2n \geq n$ *as demanded by the **PL**.*

*Then by the **PL*** $w = 0^n 1^n = x.y.z \, ,$ *where* $|x.y| \leq n$ *and* $|y| > 0$

Hence $x.y = 0^p$ *and* $y = 0^q$, $x = 0^{p-q}$ *with* $p \leq n$, $q > 0$ *and* $z = 0^{n-p}. \, 1^n$

*But according to **PL*** $x.y^i.z \in L$ *for all* $i=0,1, \ldots ;$ *and for* $i=0$ *this implies* $x.z \in L$.

Yet this cannot be true since for $q > 0$, $x.z = 0^{p-q} . (0^{n-p}. 1^n) = 0^{n-q} .1^n \notin L$

Example 2

$L = (s \in \{0,1\}^* \mid s = u.u \text{ , } u \in \{0,1\}^*)$ *is non-regular!*

As before given n *choose* $w = 0^n.1.0^n.1$ *then* $w \in L$ *and* $|w| = 2n + 2 \geq n$.

Then by the **PL** $w = 0^n.1.0^n.1 = x.y.z$, *where* $|x.y| \leq n$ *and* $|y| > 0$

Hence $x.y = 0^p$, $y = 0^q$ *with* $p \leq n$, $q > 0$ *and* $x = 0^{p-q}$ *and* $z = 0^{n-p} .1.0^n .1$

Therefore $x.z = 0^{p-q}.0^{n-p}.1.0^n .1 = 0^{n-q}.1.0^n.1$

But according to **PL** $x.z \in L$ *(*$i=0$ *case);* *but there are two cases for obtaining* $xz = u.u$ *corresponding to cutting the dividing point among the first or second group of zeros, and both cases lead to the* ***impossibility*** *of such a division to yield* $xz = 0^{n-q}.1.0^n.1 = u.u$

Hence $x.z \notin L$, *a contradiction ! and thus* L *cannot be a regular language.*

Example 3

$L = (s \in 1^* \mid s = 1^k$, k *a prime number)* *is non-regular!*

As before given n *choose* $w = 1^m$ *where* m *is a prime number with* $m > n+1$.

then by **PL** $w = 1^m = x.y.z$, *where* $|x.y| = |x|+|y| \leq n$ *and* $|y| > 0$

Or setting $r := |x|$, $p := |y|$ *and* $q := |z|$ *it follows that* $m = |x.y.z| = r+p+q$, $r+p \leq n$, $p > 0$

But according to **PL** $x.y^i.z \in L$; *or* $r+p.i+q$ *is a prime number for all integers* i

Choosing $i := r+q$ *it follows that* $r+p.i+q = (r+q).(p+1)$ *must be a prime number !*

But $r+p \leq n$ *implies* $m \leq n+q$ *and since by choice* $m > n+1$ *we have*

$n+1 < m \leq n+q$ *and therefore* $q > 1$ *and thus* $r+q \geq 2$ *and since* $p+1 \geq 2$ *it follows that*

$r+p.i+q = (r+q).(p+1)$ *cannot be a prime number for* $i = r+q$, *a contradiction !*

Closure Properties of Regular Languages

(1) L,M** regular implies (i) **L ∪ M**, (ii) **L . M** and (iii) **L are regular – follows from the definition of **REs***

(2) L** regular implies **L^c** (complement of L) is regular – replace the final state set **F** of a DFA that accepts **L** by **Q - F ;** the resulting automaton accepts **L^c

***(3) L, M** regular implies **L ∩ M** is regular – short path :*

***L ∩ M = (L^c ∪ M ^{c}) ^{c} ,** which by (1) and (2) proves the result*

*- long path : If **A** and **B** are DFA that accept **L** and **M** then*

A × B** is a DFA that accepts **L ∩ M

The Product Automaton $A \times B$

$A = (Q, \Sigma, \delta_A, s, F_A)$; $B = (R, \Sigma, \delta_B, t, F_B)$

$A \times B := (Q \times R, \Sigma, \delta, (s,t), F_A \times F_B)$ where

$\delta((q,r), \sigma) := (\delta_A(q,\sigma), \delta_B(r,\sigma))$

Fact to be proved by induction on the length of u :

$\delta E((q,r),u) = (\delta_A E(q,u), \delta_B E(r,u))$

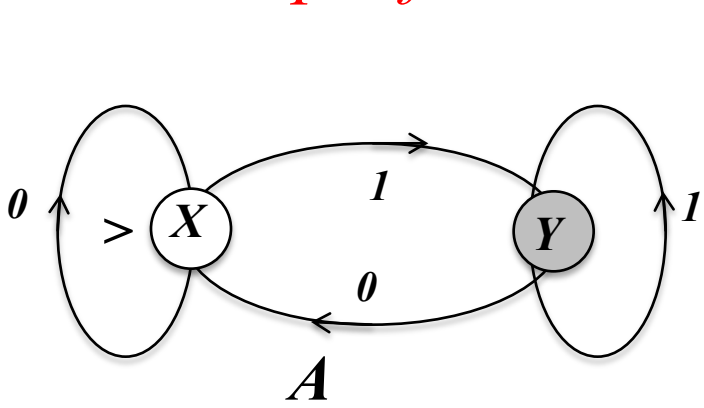
Proof of $L(AxB) = L(A) \cap L(B)$

$u \in L(A \times B) \Leftrightarrow \delta E((s,t),u) \in F_A \times F_B \Leftrightarrow (\delta_A E(s,u), \delta_B E(t,u)) \in F_A \times F_B$

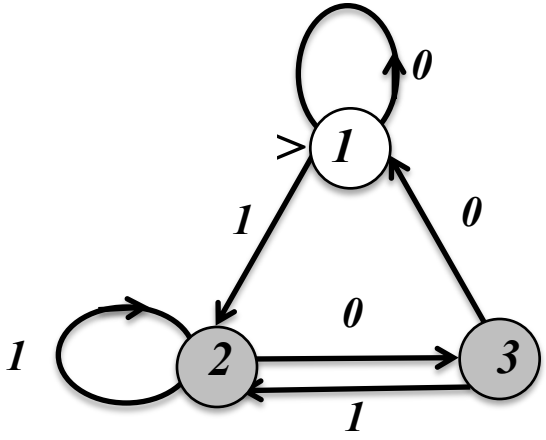
$\Leftrightarrow \delta_A E(s,u) \in F_A \wedge \delta_B E(t,u) \in F_B$

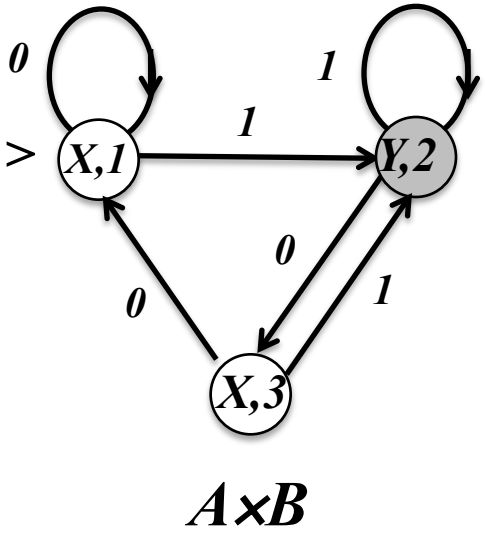
$\Leftrightarrow u \in L(A) \wedge u \in L(B) \Leftrightarrow u \in L(A) \cap L(B)$

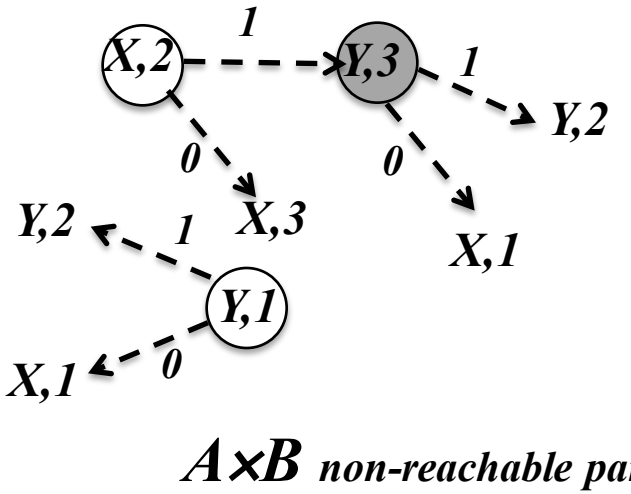
Example for the Product Automaton $A \times B$



×







(4) $\boldsymbol{L, M}$ *regular then* $\boldsymbol{L - M}$ *is regular ;* $\boldsymbol{L - M = L \cap M^c}$ *hence*

by previous results regularity follows

(5) $\boldsymbol{L}$ *regular then* $\boldsymbol{L^R := (u \in \Sigma^* \mid u^R \in L)}$ *where* $\boldsymbol{R}$ *denotes*

reversal of strings –

replace the DFA that accepts $\boldsymbol{L}$ *by the*

NFA obtained from the former by (i) interchanging initial and final

state sets ; and (ii) changing the direction of all transition arcs

Decision Problems for Regular Languages

Adjacency $n \times n$ *matrix* A *for a directed graph with* n *nodes*

$a_{ij} = 1$ *if there is an arc from node* i *to node* j

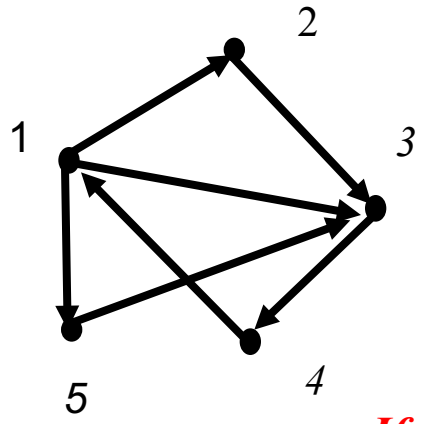
$= 0$ *otherwise*

The i-to-j reachability problem *: is there a path from node i to node j ?*

Let $w^k_{ij} = 1$ *, if there is a path from* i *to* j *using intermediate nodes* $1, \ldots, k$ *and* $= 0$ *otherwise. Answer to the reachability problem is* ***YES*** *iff* $w^n_{ij} = 1$. *Note that* $[w^0_{ij}] = [a_{ij}]$

Warshall's Algorithm *:* $w^k_{ij} = w^{k-1}_{ij} \vee (w^{k-1}_{ik} \wedge w^{k-1}_{kj}), \quad k = 1,..,n$

Warshall at Work !



1	1	1	0	1
0	1	1	0	0
0	0	1	1	0
1	0	0	1	0
0	0	1	0	1

1

1	1	1	0	1
0	1	1	0	0
0	0	1	1	0
1	1	1	1	1
0	0	1	0	1

1,2

If new entry node is i then add to row j≠i for which jith entry is 1 , row i

1	1	1	0	1
0	1	1	0	0
0	0	1	1	0
1	1	1	1	1
0	0	1	0	1

1,2,3

1	1	1	1	1
0	1	1	1	0
0	0	1	1	0
1	1	1	1	1
0	0	1	1	1

1,2,3,4

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

1,2,3,4,5

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Big O functions

Let $f(n)$ *be a positive increasing function of the positive integer* n.

A function $g(n)$ *is called a* **big O** *function of* $f(n)$ *shown by* $g(n) = O(f(n))$ **iff :**

there are constants $K_1, K_2 > 0$ *such that :*

$|g(n)| < K_1 f(n)$, *for all* $n > K_2$ *(*$g(n)$ *increases no faster than* $f(n)$ *asymptotically !)*

Example

The function $g(n) = n^3 + 3n^2 + 7n + 5 \log(n)$ *is an* $O(n^3)$ *function*

$|g(n)| = n^3 (1 + 1/n + 7/n^2 + 5 \log(n)/n^3) < n^3 (1 + 1 + 7 + 5) = 14 n^3$

Since for $n > 1$ *we have* $1 + 1/n + 7/n^2 + 5 \log(n)/n^3 < (1 + 1 + 7 + 5)$

using $\log(n) < n^3$ *for* $n > 1$ **;** *and therefore result follows with* $K_1 = 14$ *and* $K_2 = 1$

Complexity of Warshall's Algorithm

Two basic binary operations ('v' and 'λ') for each **i,j** *and for* **k = 1, …, n** *;*

hence total $2 * n^2 * n = O(n^3)$ *operations*

Notation *: for a Finite State Automaton (**DFA** or **NFA**) let* A_σ *denote the adjacency matrix corresponding to the directed graph after removing all transition arcs that are NOT labeled by* σ

Solution of the EPSILON-closure algorithm

(1) Apply ***Warshall's Algorithm*** *to* A_ε *to compute* $W_\varepsilon = [w_{ij}]$ → $O(n^3)$

(2) Epsilon closure of state (node) **i** *are the states (nodes) with value* **1** *in row* **i** *of* W_ε

(3) Epsilon closure of a set **S** *of nodes are the nodes with value* **1** *of the row obtained from the union ('v') of the rows corresponding to the nodes in* **S** → $O(n^2)$

hence total complexity is $O(n^3) + O(n^2) = O(n^3)$

Reachability from a given set of source nodes !
Reachability Algorithm

(1) ***Initialization*** *: Let the initial* ***LIST*** *consist of the row indices corresponding to the source nodes.*

(2) Process the current (unused) index in the ***LIST*** *and mark the column no.s with a* ***1*** *at the row corresponding to the current index and add those column indices that are not already in the* ***LIST*** *to the* ***LIST.***

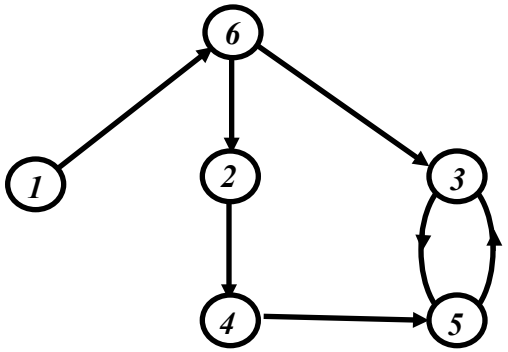
(3) Delete the row used and the marked columns and repeat (2) for the ***next*** *(row) index in the* ***LIST*** *; stop when all the indices in the* ***LIST*** *are used at step (2) !*

The vertices reached are members of the final LIST !

At step 2 the maximum no. of columns marked is ***n ;*** *max. total no. of iterations is* ***n***

Hence complexity = $O(n^2)$

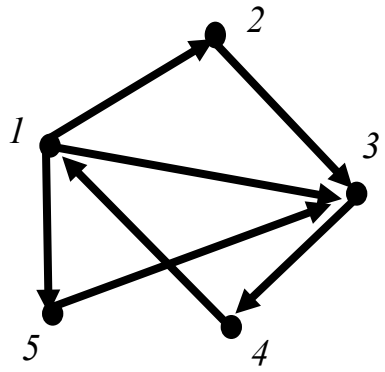
	1	2	3	4	5	6
1	1	0	0	0	0	1
2	0	1	0	1	0	0
3	0	0	1	0	1	0
4	0	0	0	1	1	0
5	0	0	1	0	1	0
6	0	1	1	0	0	1



Reachability from vertices 2 and 6

2,6 → 2,6,4 → 2,6,4,3 → 2,6,4,3,5 → 2,6,4,3,5 → 2,6,4,3,5

Example : Reachability from vertices 2 and 5



Initial ***LIST*** *= (2,5)*

	1	2	3	4	5
1	1	1	1	0	1
2	0	1	1	0	0
3	0	0	1	1	0
4	1	0	0	1	0
5	0	0	1	0	1

LIST *= (2,5,3)*

	1	4	5
1	1	0	1
3	0	1	0
4	1	1	0
5	0	0	1

LIST *= (2,5,3)*

	1	4
1	1	0
3	0	1
4	1	1

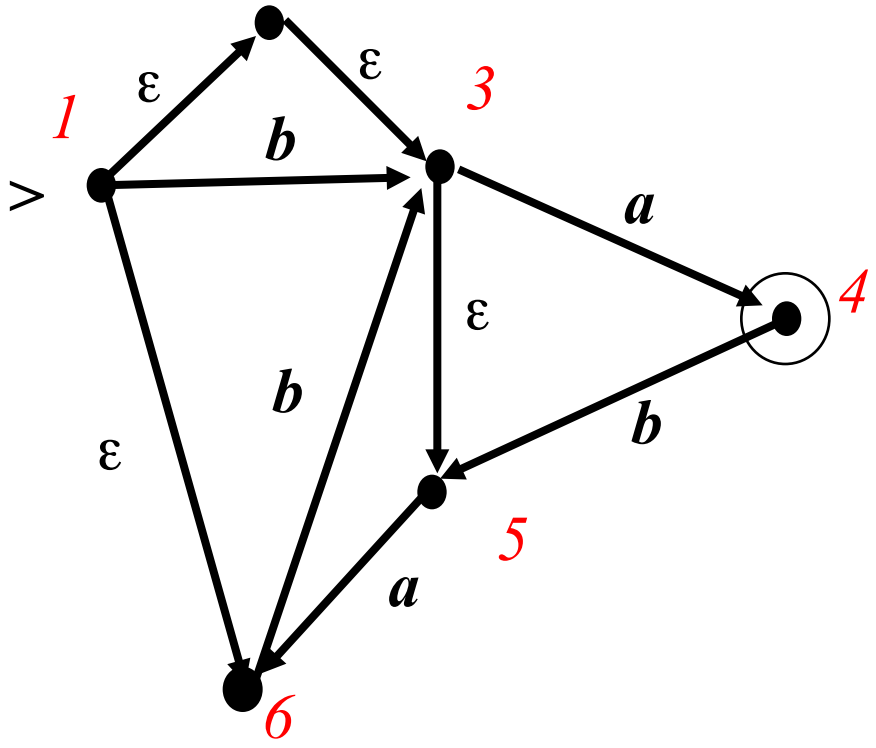
LIST *= (2,5,3,4)*

	1
1	1
4	1

LIST *= (2,5,3,4,1)*

empty matrix

reachability from vertices 2,5= all the vertices



$A_\varepsilon =$

	1	2	3	4	5	6
1	1	1	0	0	0	1
2	0	1	1	0	0	0
3	0	0	1	0	1	0
4	0	0	0	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

Warshall

$W_\varepsilon =$

	1	2	3	4	5	6
1	1	1	1	0	1	1
2	0	1	1	0	1	0
3	0	0	1	0	1	0
4	0	0	0	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

Example: ***From ε-NFA to NFA ,***
i.e. eliminate all ε-transitions

$A_a =$

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	0	0

$A_b =$

0	0	1	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	0	1	0	0	0

Non ε -NFA

$B_b =$

0	0	1	0	1	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	0
0	0	1	0	1	0

row 1

$W_\varepsilon \rightarrow (1,2,3,5,6)$

$A_b \rightarrow (3)$

$W_\varepsilon \rightarrow (3,5)$

Complexity of Regular Language Conversions

(1) ε - NFA to ***NFA*** $\rightarrow O(n^3)\ |\Sigma| + O(n^3) = O(n^3)\ |\Sigma|$

(2) NFA to ***DFA*** $\rightarrow O(n^2).|\Sigma|.\ 2^n = 2^{(n+\ log\ (O(n**2).|\Sigma|))} = 2^{O(n)}$

(Exponential!)

(3) RE to ***ε - NFA*** $\rightarrow$ ***O(n)*** (***n*** = # *basic operations in* ***RE***)

(4) ***DFA*** to ***RE*** $\rightarrow$ $O(|\Sigma|.n^2.4^n) = 2^{O(n)}$

(Exponential!)

Testing emptiness of (and membership in) a language L

(1) Emptiness Problem : *Is the language accepted by a DFA or an NFA empty ?*

Apply ***Reachability Algorithm*** *to* ***NFA*** *(or* ***DFA****) where all edges are included disregarding the label. If a final state is reached from any initial state,* ***L*** *is non-empty. Complexity is* $O(n^2)$ *both for* ***NFA*** *and* ***DFA***

For ***RE*** *complexity is* $O(n)$

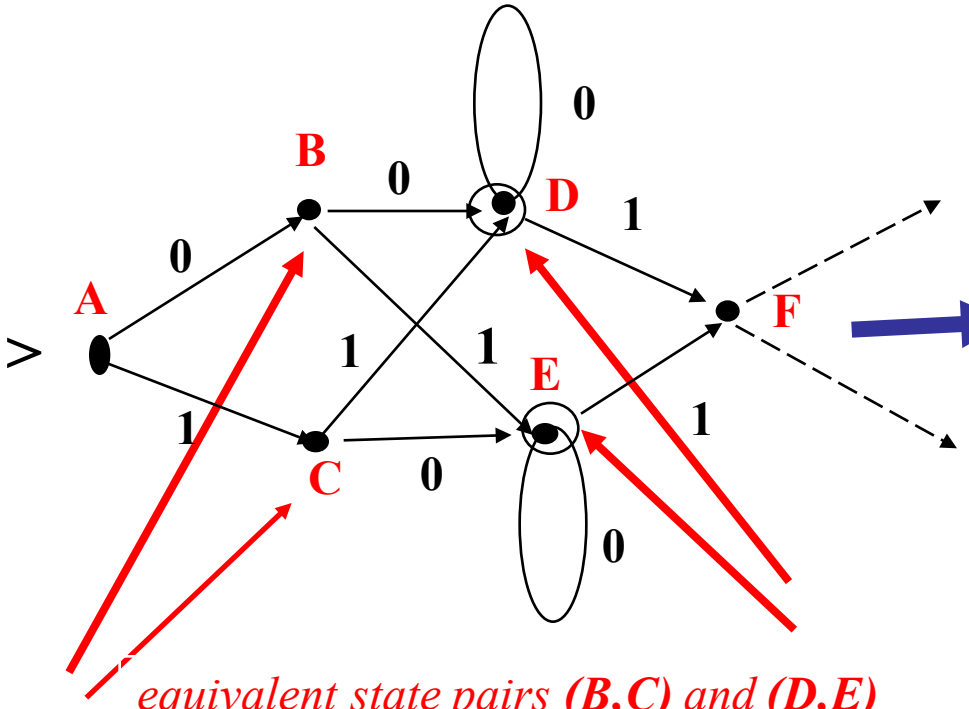
(2) Membership Problem : *does a* ***DFA*** *or an* ***NFA*** *accept the string* ***w ?***

$O(|w|+n)$ *for* ***DFA*** *representation.*

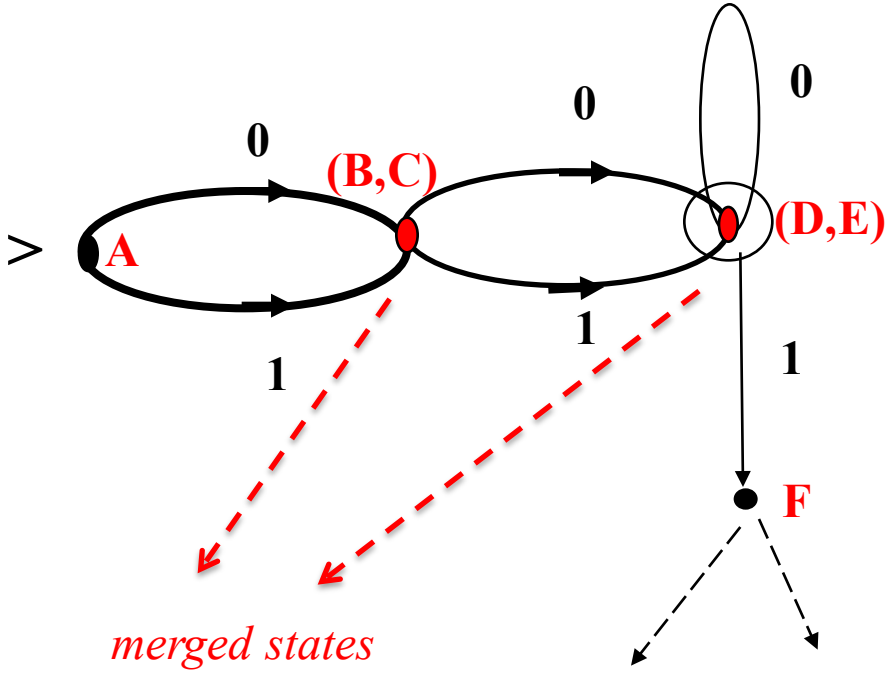
$O(|w|.n^2+n^2)$ *and* $O(n^3+|w|.n^2)$ *for* ***NFA*** *and* ***ε-NFA*** *representations respectively.*

$O(n^3+|w|.n^2)$ *for* ***RE*** *representations.*

An informal description of State Equivalence

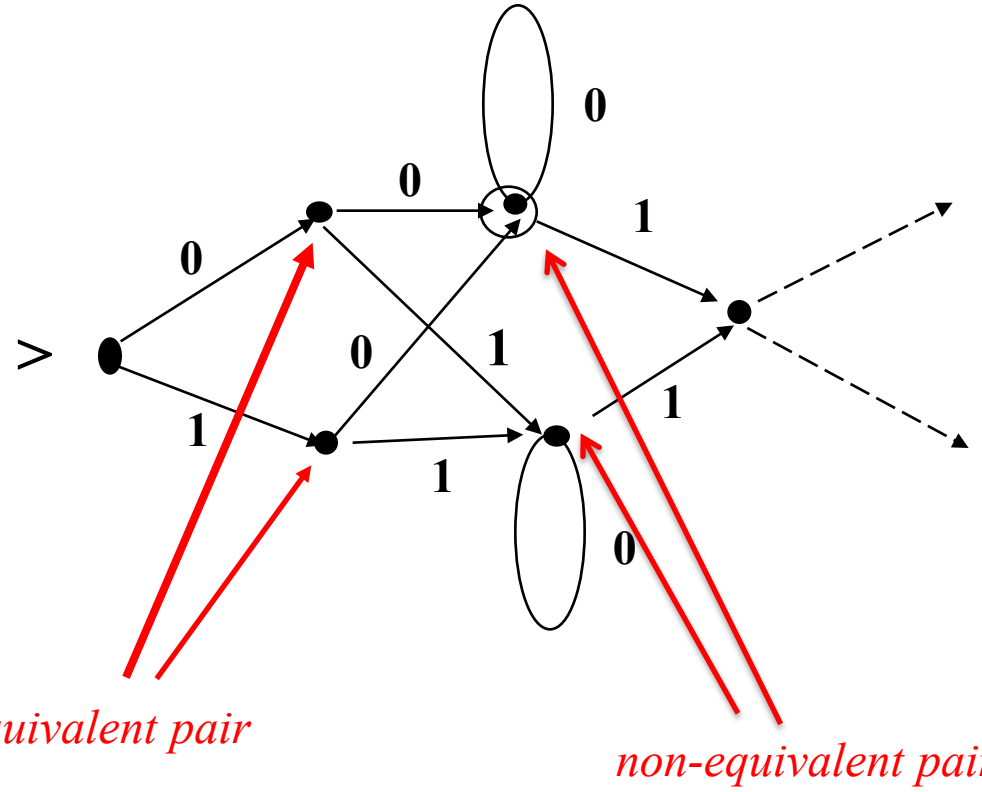


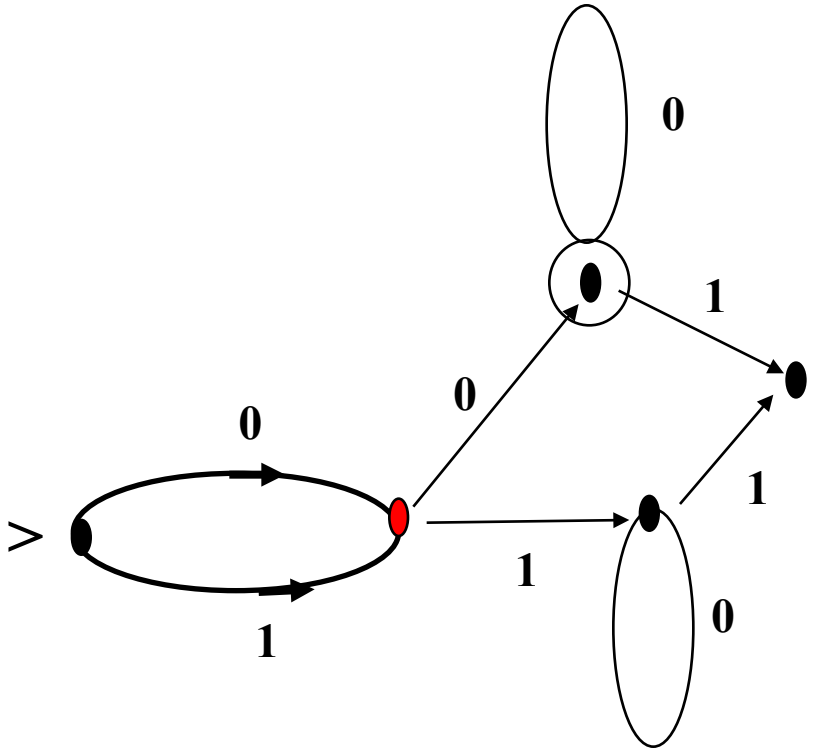
*equivalent state pairs **(B,C)** and **(D,E)***



merged states
(B,C)** and **(D,E)

The role of final states





State Equivalence for DFA

$A = (Q, \Sigma, \delta, s, F)$ be a *DFA* accepting the language $L(A)$

Definition $q, r \in Q$ *are said to be* ***equivalent*** *states of* A *if*

for all $w \in \Sigma^*$: $\delta E(q,w) \in F$ ***if and only if*** $\delta E(r,w) \in F$

States that are not ***equivalent*** *are called* ***distinguishable***

Note that (and prove !) if q *and* r *are equivalent and if*

$q' = \delta E(q,w)$ *and* $r' = \delta E(r,w)$ *then* q' *and* r' *are equivalent*

Hint : Use and prove the fact that $\delta E(r,u.v) = \delta E(\delta E(r,u),v)$

for all $u,v \in \Sigma^*$ *and* $r \in Q$ *(use induction on the length of* v*)*

Recursive computation of distinguishable state pairs

(Table Filling Algorithm)

Basis *: A pair* (r,q) *is* ***distinguishable*** *if one is* ***accepting***

(final) and the other is a ***non-accepting*** *(non-final) state.*

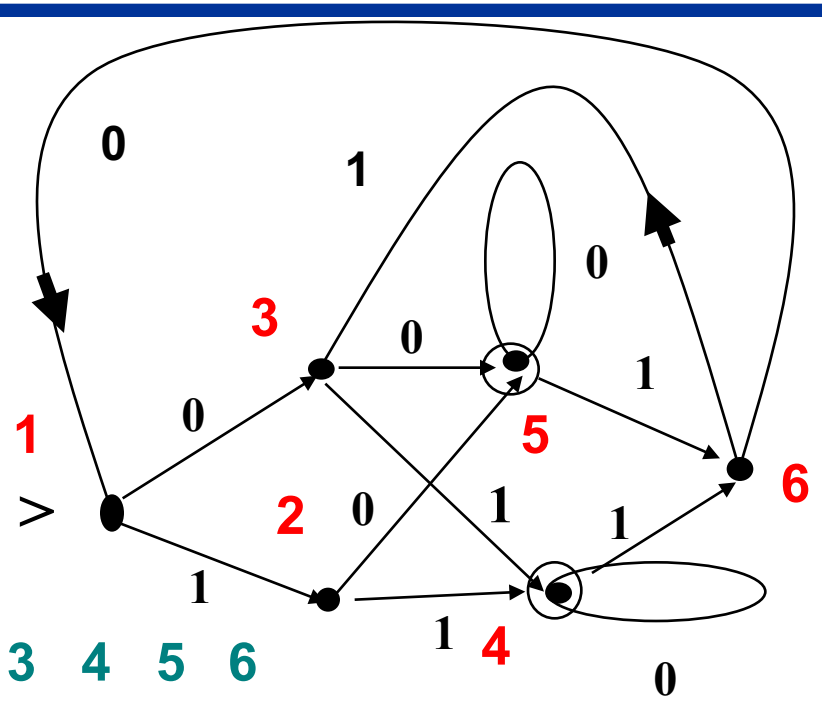
Induction *: If for some* $a \in \Sigma$, $r' = \delta(r, a)$ *and* $q' = \delta(q, a)$ *and* r' *and* q' *are* ***distinguishable*** *then* r *and* q *are also* ***distinguishable***

Proof *: Given that* r' *and* q' *are distinguishable (i.e.* ***NOT*** *equivalent), there exists an input string* u *such that* $\delta E(r', u) \in F$ *and* $\delta E(q', u) \notin F$ *(or vice-versa)* ***;***

But since $r' = \delta(r, a)$ *and* $q' = \delta(q, a)$ *we have*

$\delta E(r',u) = \delta E(\delta(r, a),u) = \delta E(r,a.u)$ *and* $\delta E(q',u) = \delta E(\delta(q, a),u) = \delta E(q,a.u)$

$\delta E(r, a.u) \in F$ *and* $\delta E(q, a.u) \notin F$ *and therefore* r *and* q *are* ***distinguished*** *by the input string* $a.u$



	1	2	3	4	5	6
1				x	x	
2				x	x	
3				x	x	
4						x
5						x
6						

	1	2	3	4	5	6
1		x	x	x	x	
2				x	x	x
3				x	x	x
4						x
5						x
6						

	1	2	3	4	5	6
1		x	x	x	x	x
2				x	x	x
3				x	x	x
4						x
5						x
6						

No more improvement

Table Filling Algorithm

Complexity of Table Filling Algorithm

Number of entries of the table = $\mathbf{n(n-1)/2 = O(n^2)}$

For every entry of the table inputs are checked whether the next state pair is already marked. Call this ***one*** *unit of computation.*

Since the table has $\mathbf{O(n^2)}$ *number of entries the complexity involved is* $\mathbf{O(n^2)}$ *for each run. But because each run guarantees minimum* ***one*** *improvement the total number of runs is* $\mathbf{O(n^2)}$ ***;*** *hence complexity is* $\mathbf{O(n^4)}$**.**

(the explanation on page 160 of the main textbook which claims that the complexity of the table filling algorithm is $\mathbf{O(n^2)}$ *is* ***unconvincing*** *as reformulated in the next 2 slides!)*

Table Filling (Economical ?)Algorithm

Step 1 *: **reverse** the direction of all the edges in **A** call the result* A^R

Step 2 *: Form the product automaton* $B=A^R \times A^R$ *where the states* **(x,y)** *and* **(y,x)** *are **merged** and the unreachable states* **(x,x)** *are omitted.*

Step 3 *: Define the initial state set of **B** as all pairs **(f,i)** where **f** is a final state and **i** is a non-final state.*

Step 4 : *Compute all the reachable states of **B** which are **all** the distinguishable pairs of* A ! **Complexity** $= O((n^2)^2) = O(n^4)$

Why A^R *??* → *If for some* $a \in \Sigma$, $r' = \delta(r, a)$ *and* $q' = \delta(q, a)$ *and* r' *and* q' *are **distinguishable** then **r** and **q** are also **distinguishable***

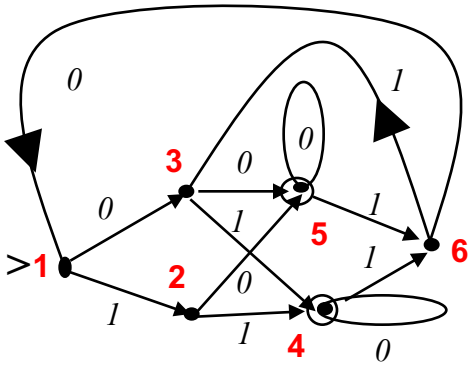
Note :

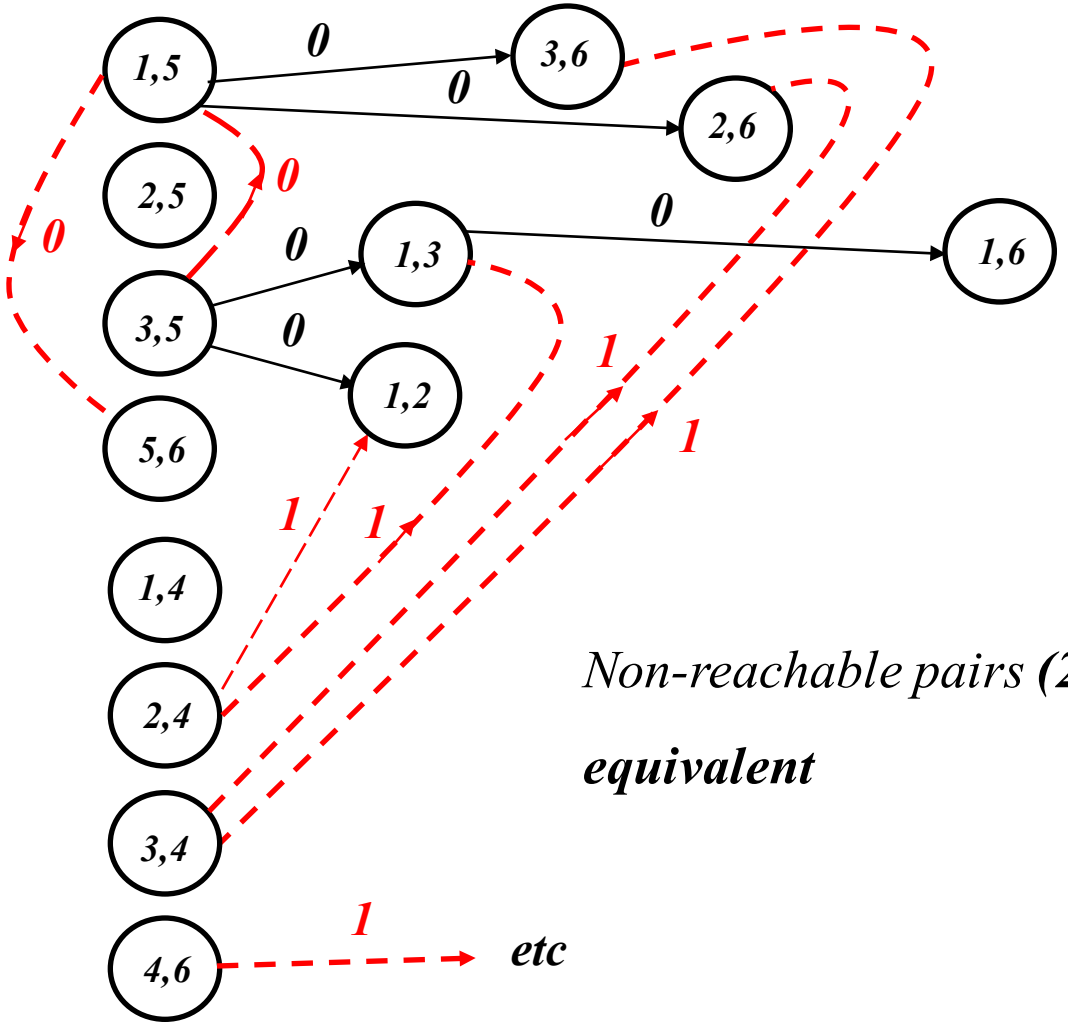
1 - *# of states of* $B = (n^2 - n)/2 = O(n^2)$

2 – *In the product automaton **B** there are no transitions to states **(x,x)** from any **(w.z)** with* $w \neq z$ *, which justifies their removal in **B** (this is because **A** is a **DFA**)*

Example

***B** has (36-6)/2 = **15** states*





*Non-reachable pairs **(2,3)** and **(4,5)** are*

equivalent

Definition

Let $A=(P,\Sigma,\delta_A,s,F_A)$ *and* $B=(Q,\Sigma,\delta_B,t,F_B)$ *be two given* **DFA**s .

Two states $p \in P$ *and* $q \in Q$ *are said to be* ***equivalent*** *(shown* $p \equiv q$ *) iff*

$\delta_A E(p,u) \in F_A$ ***if and only if*** $(\Leftrightarrow)$ $\delta_B E(q,u) \in F_B \quad \forall u \in \Sigma^*$.

A is said to be ***equivalent*** *to B (shown* $A \equiv B$*) iff* $s \equiv t$.

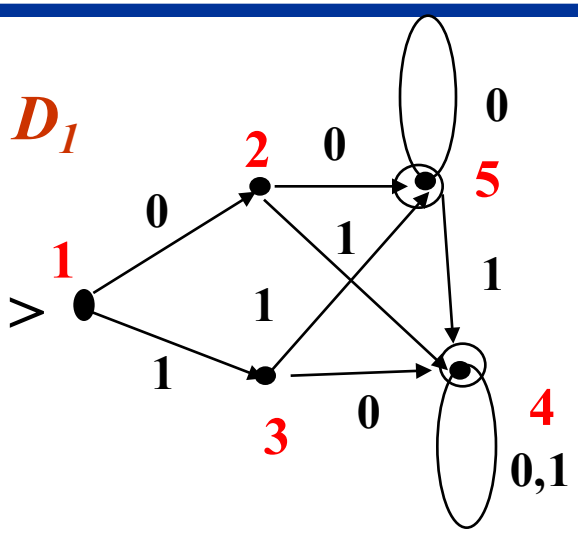
Corollary to the Definition

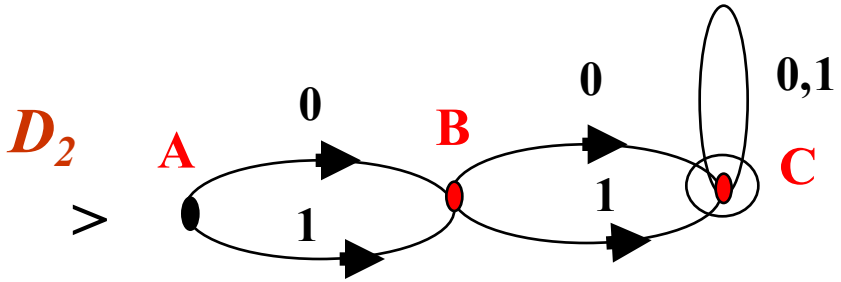
$A \equiv B$ ***if and only if*** $L(A) = L(B)$

Proof of Corollary

$A \equiv B \Leftrightarrow s \equiv t \Leftrightarrow \delta_A E(s,u) \in F_A (\Leftrightarrow) \delta_B E(t,u) \in F_B$

$\Leftrightarrow u \in L(A) (\Leftrightarrow) u \in L(B)$





	2	3	4	5	A	B	C
1	X	X	X	X		X	X
2			X	X	X		X
3			X	X	X		X
4					X	X	
5					X	X	
A						X	X
B							X

$1 \equiv A$

$D_1 \equiv D_2$

$L(D_1) = L(D_2)$

Theorem

Any pair of states that are not eventually filled by the ***Table Filling Algorithm*** *are* ***equivalent*** *states (either within a* ***DFA*** *or in two distinct* ***DFA****s)*

Proof : *Call a pair* (p,q) *unfilled by Algo a* ***bad*** *pair. Then* (p,q) *is an unreachable state of the automaton* $B := A^R \times A^R$ *in the previous slide.*

Suppose now that p *and* q *are distinguishable states. Then there is an input string* s *such that* $p' := \delta_A E(p,s) \in F_A$ *and* $q' := \delta_A E(q,s) \notin F_A$. *But this implies that* $(p,q) \in \delta_B E((p',q'), s^R)$ *where* s^R *is* s *reversed which further contradicts that* (p,q) *is unreachable in* B *in which* (p',q') *is an initial state.*

*A **minimum state machine** is obtained after all equivalent states are merged into common states thereby reducing the total number of states.*

Merging of equivalent states is justified by the proved fact that all the states on a common path from two equivalent states remain equivalent

*Can you beat a minimum state machine **M** that accepts **L** ?*

*Suppose you can ! then there is an L-accepting **DFA** , say **W,** with less states then that of **M** . Let $u_1, \ldots, u_n$ be strings that drive the initial state to the **n** distinct states of **M** . Apply these input strings to **W** and by pigeon hole at least 2 such strings will end up in an identical state : a contradiction ! Why ?*

(The auxiliary text : Lewis & Papadimitrou approach p (92-111))

The definition of the state equivalence '≡' is an equivalence relation

Definition $q \equiv_k p$ *if* $\delta E(q,u) \in F \Leftrightarrow \delta E(p,u) \in F$, $\forall u$ *with* $|u| \leq k$

Proposition $q \equiv_k p \Leftrightarrow q \equiv_{k-1} p \wedge \delta(q,\sigma) \equiv_{k-1} \delta(p,\sigma)\ \forall \sigma \in \Sigma$

Proof of Proposition

Assume $q \equiv_k p$ *then* $q \equiv_{k-1} p$ *since* $|u| \leq k-1 < k$

Also for any $\sigma.u$ *with* $|\sigma.u| \leq k$, $\delta E(q,\sigma.u) \in F \Leftrightarrow \delta E(p,\sigma.u) \in F$ *since* $q \equiv_k p$

But this implies $\delta(q,\sigma) \equiv_{k-1} \delta(p,\sigma)$ *since* $\delta E(\delta(q,\sigma), u) \in F \Leftrightarrow \delta E(\delta(p,\sigma), u) \in F$

Conversely assume $\delta(q,\sigma) \equiv_{k-1} \delta(p,\sigma)\ \forall \sigma \in \Sigma$ *then for any* $|w| \leq k$ *decompose*

$w = \sigma..u$ *and by assumption* $\delta E(\delta(q,\sigma), u) \in F \Leftrightarrow \delta E(\delta(p,\sigma), u) \in F$

Hence $\delta E(q, w) \in F \Leftrightarrow \delta E(p, w) \in F$ *and* $q \equiv_k p$

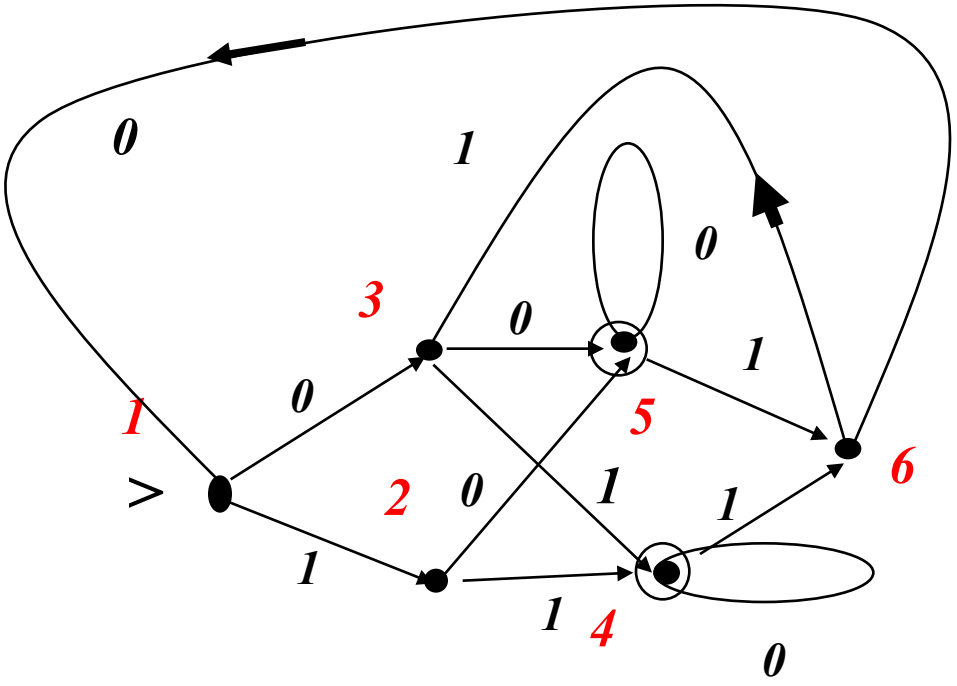
Algorithm (P&L)

Suppose the state set Q is already partitioned according to ' $\equiv_k$ *'*

with partition $P_k = (Q(1), \ldots, Q(m_k))$

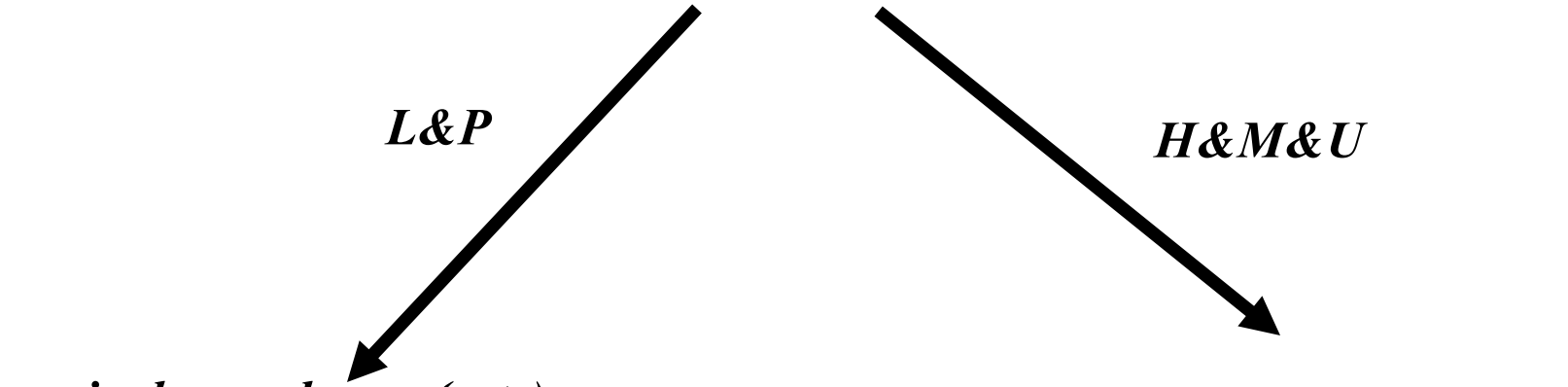
then each $Q(i)$ *is split further by '* $\equiv_{k+1}$ *' where* q *and* p *are in the same component if and only for all* $\sigma \in \Sigma$ *both* $\delta(q,\sigma)$ *and* $\delta(p,\sigma)$ *are in the same* $Q(j)$, *for some* j .

Algorithm halts if no further splitting occurs



Step 1	Step 2	Step 3	Step 4
(1236)	(16)	(1)	No Change!
(45)	(23)	(6)	
	(45)	(23)	
		(45)	

Two ways of representing equivalence

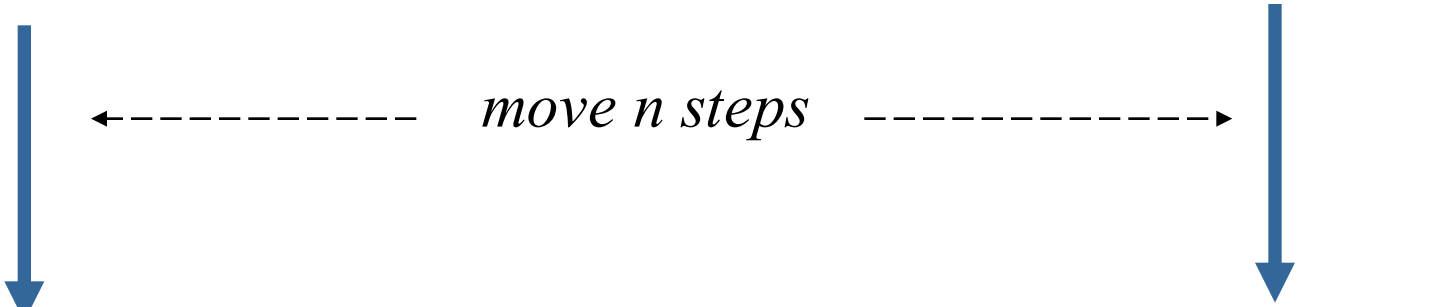


equivalence classes (sets)

Start k=0 {F, Q-F}

distinguishable pair set

Start k=0 : F×(Q-F)



k=n : final partition

k=n : D ⊆ Q×Q

Note : if $p \equiv q$ *then* $\delta(p, a) \equiv \delta(q,a)$ *for all* $a \in \Sigma$

AND…

(notation for NOT in logic)

if $\delta(p,a)$ $(\neg \equiv)$ $\delta(q,a)$ *(distinguishable) for some* $a \in \Sigma$ *then*
p $(\neg \equiv)$ q

(Logic : $A \Rightarrow B$ *is true iff* $\neg B \Rightarrow \neg A$ *)*

The Abstract Theory

Let Σ *be an alphabet and* $L \subseteq \Sigma^*$ *be a language*

For $u,v \in \Sigma^*$ *: define* $u \equiv v$ *if* $(u.s \in L \Leftrightarrow v.s \in L)\ \forall s \in \Sigma^*$

***L** is called a **regular** iff there are a finite number of equivalence classes.*

*Define (abstractly !) a **DFA** **M** as follows :*

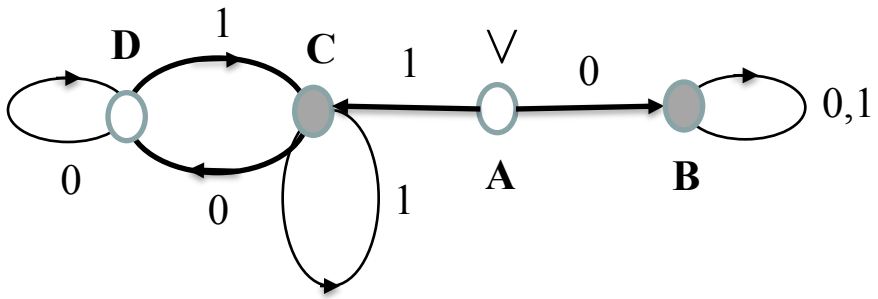
***Q** = set of equivalence classes ;* $\delta([u],a) := [u.a]$; $s = [e]$

$F = ([u] \in Q \mid u \in L)$

*Every **minimum** state **DFA** that accepts **L** is **isomorphic** to **M***

(Myhill-Nerode theory)

Example



States as language equivalence classes

[e] = e** for **A

[0] = 0.(0+1)* for B

[1] = 1.1*(0.0*.1.1*)* = $1^+(0^+.1^+)^*$ for C

[10] = $1^+(0^+.1^+)^*.0^+$ for D

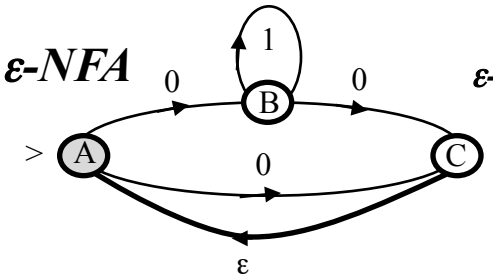
$L = L_B + L_C$ = [0]+[1] *where* ***L_B*** *and* ***L_C*** *are disjoint sets*

Example

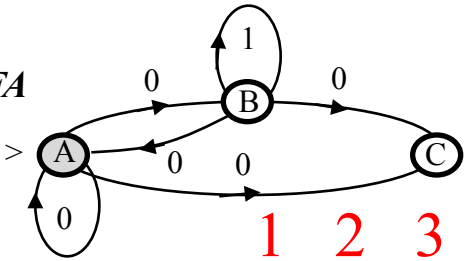
$E = (0.1^*.0+0)^*$

A (1)	0	ABC
A	1	∅ (5)
ABC(2)	0	ABC
ABC	1	B
B(3)	0	AC
B	1	B
AC(4)	0	ABC
AC	1	∅

ε-NFA

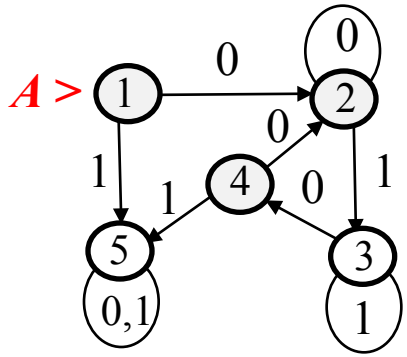


ε-NFA to NFA

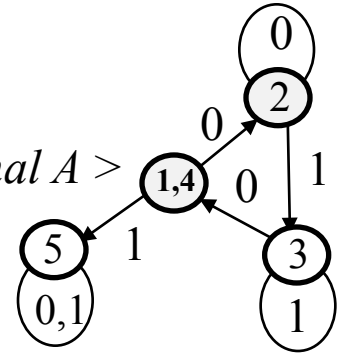


	1	2	3	4	5
1		3	1	E	1
2			1	3	1
3				1	2
4					1
5					

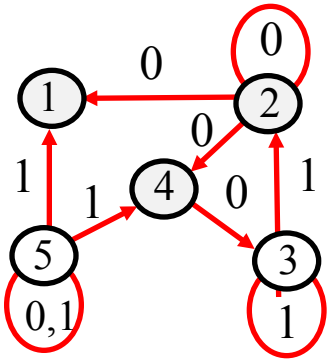
NFA to the DFA A



Minimal A >



A^R



$A^R \times A^R$