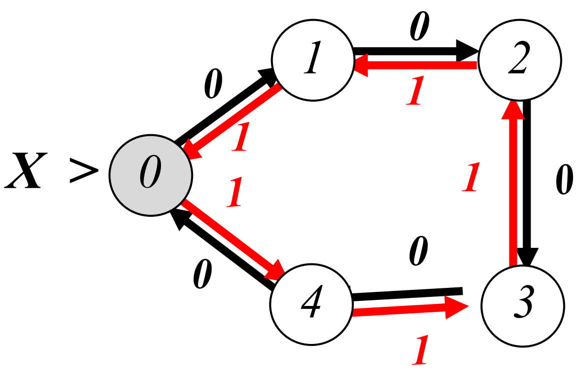
How expressive is finite state linguistics ?

$L(N) := (s \in \{0,1\}^* \mid (\#0\text{'s in } s) \bmod N = (\#1\text{'s in } s) \bmod N)$

$= (s \in \{0,1\}^* \mid \#0\text{'s in } s =_{\bmod N} \#1\text{'s in } s)$

x mod N := remainder after dividing x by N

*DFA **X** to accept **L(5)** must have **5** states*



M mod +∞ = M

hence ' $=_{\bmod +\infty}$ ' is same as ' = '

*DFA to accept **L(N)** must have **N** states !*

*DFA to accept **L(+∞)** must have +∞ states ! ? But what is **L(+∞)** ?*

$L(+\infty) := (s \in \{0,1\}^* \mid \#0\text{'s in } s =_{\bmod +\infty} \#1\text{'s in } s)$

$= (s \in \{0,1\}^* \mid \#0\text{'s in } s = \#1\text{'s in } s)$

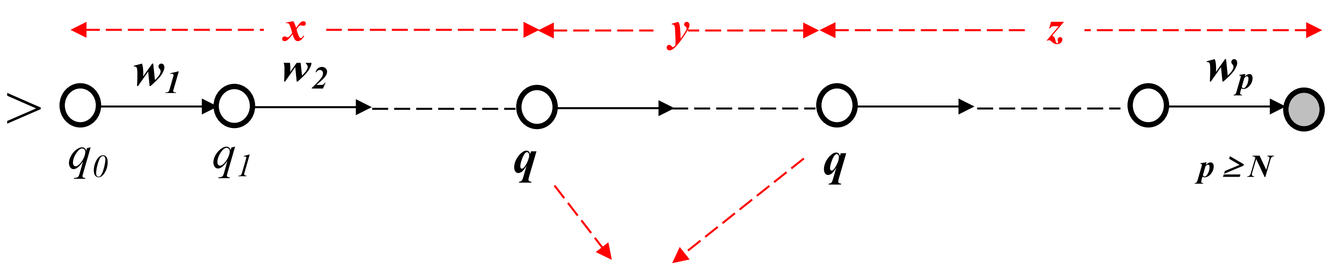
The previous slide is an example of a simple language that is ***intuitively*** *shown to be* ***unacceptable*** *by a finite state automaton !*

We make this intuition mathematically precise by the ***Pumping Lemma***

Consider a DFA ***X*** *with* ***N*** *states that accepts a language* ***L***

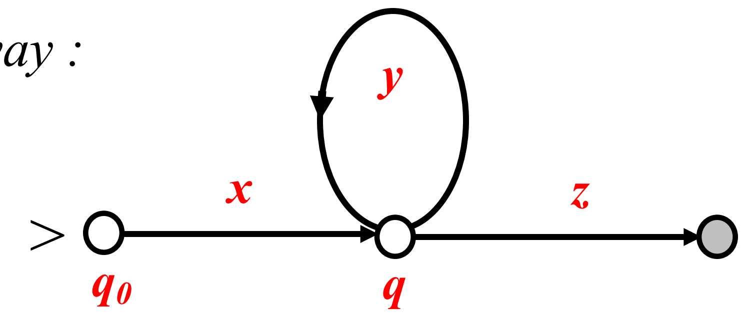
Let $\mathbf{w} \in \mathbf{L}$ *with the property that* $|\mathbf{w}| \geq N$

Apply ***w*** *to* ***X*** *starting at the initial state and plot the state transitions as follows :*



*since the no. of states visited before the arrival at the final state is > **N** ; some state, say **q**, is repeated on the way . We assume that **q** is the **first** instance of a state repeating a past instance of itself ; hence the states before the second **q** are visited are all distinct and are ≤ **N**).*

The string moving through the states in the previous slide can be plotted in the following way :



Hence : $w = x.y.z$

where : (i) $|x.y| \leq N$ *(by the assumption that* q *is the first instance of a state repetition) ;*

(ii) $|y| > 0$ *(at least one transition from* q *to* q*)*

Finally we make the key observation : we can loop from q *to* q *as many times (including 0 times !) as we want and still end up at the accepting final state !*

Hence we conclude that $x.y^j.z$ *is accepted by* X *for all* $j = 0, 1, 2, \ldots$

Pumping Lemma *(Explanatory Version)*

Let $\boldsymbol{L}$ *be a language accepted by a DFA* $\boldsymbol{X}$ *with* $\boldsymbol{N}$ *states.*

Let $\boldsymbol{w} \in \boldsymbol{L}$ *with* $|\boldsymbol{w}| \geq N$.

Then there exists strings $\boldsymbol{x,y}$ *and* $\boldsymbol{z}$ *such that* $\boldsymbol{w}$ *can be decomposed as :*

$\boldsymbol{w = x.y.z}$ *where (i)* $|\boldsymbol{x.y}| \leq N$ *and ; (ii)* $|\boldsymbol{y}| > \boldsymbol{0}$.

The DFA ***X*** *that accepts* ***w must necessarily accept*** *an infinite collection of strings given by :*

$\boldsymbol{x.y^j.z \in L}$ *for all* $j=0,1,2,\ldots$

Pumping Lemma (Elegant Version)

Let $\mathbf{L}$ *be a* ***regular*** *language then there exists an integer* $\mathbf{N > 0}$ *such that*

For any $\mathbf{w} \in \mathbf{L}$ *with* $|\mathbf{w}| \geq N$ *there exists strings* $\mathbf{x,y}$ *and* $\mathbf{z}$ *such that* $\mathbf{w}$

can be decomposed as $\mathbf{w = x.y.z}$ *where*

(i) $|\mathbf{x.y}| \leq N$;

(ii) $|\mathbf{y}| > \mathbf{0}$;

(iii) $\mathbf{x.y^j.z} \in \mathbf{L}$ *for all* $\mathbf{j=0,1,2,\ldots}$

Pumping Lemma *can be used to prove that a language* ***L*** *is* ***not regular*** *;*

Or equivalently that ***L cannot be accepted by a DFA***

Caution (dikkat!)*:Pumping Lemma* ***CANNOT*** *be used to prove that a language is regular :* ***don't try!!*** *Pumping Lemma shows that a given language* **L is NOT regular** *by* ***contradiction*** *as explained by the following steps :*

Step 1 . *Assume that* ***L*** *is accepted by some DFA with* **N** *states (to be contradicted at the end!) .*

Step 2 . Choose *a* ***specific*** *and* ***suitable*** *(simple?)* ***w*** *in* ***L*** *with* $|w| \geq N$ *.*

Step 3 . *Using the decomposition* $w = x.y.z$ *and the facts* $|x.y| \leq N$ *and* $|y| > 0$ *, express separately* ***x,y*** *and* ***z*** *according to your choice of* ***w*** *.*

Step 4 . *Show that according to* ***x,y*** *and* ***z*** *as computed in* ***Step 3*** *the string* $x.y^j z$ *is* ***NOT*** *in* ***L*** *for some* $j \geq 0$ *(usually* $j=0$ *) contradicting the Pumping Lemma .*

Proving languages non-regular !

Example 1

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

*Assume the contrary (**L** is regular !) and apply the steps in the previous slide*

Step 1 *Let **N** be the no. of states of a DFA that accepts **L***

Step 2 *Choose* $w = 0^N 1^N$ *which is in **L** ; then* $|w| = 2N \geq N$ *as demanded by the **PL**.*

Step 3 *Then* $w = 0^N 1^N = x.y.z$ *, where* $|x.y| \leq N$ *and* $|y| > 0$

Hence $x.y = 0^p$; $y = 0^q$ *, and thus* $x = 0^{p-q}$ *where* $p \leq N$, $q > 0$ *and so* $z = 0^{N-p}.\ 1^N$

Step 4 *But according to **PL*** $x.y^j.z \in L$ *for all* $j = 0, 1, \ldots$; *and for* $j=0$ *this implies that*

$x.z \in L$. *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

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

Step 1 *Let* **N** *be the no. of states of a DFA that accepts* **L**

Step 2 *Choose* $w = 0^N.1.0^N.1 = u.u$ *with* $u = 0^N.1$ *; then* $w \in L$ *and* $|w| = 2N + 2 \geq N$.

Step 3 *Then since* $w = 0^N.1.0^N.1 = x.y.z$ *, where* $|x.y| \leq N$ *and* $|y| > 0$ *it follows that*

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

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

Step 4 *According to the* **PL** *for* $j=0$ *, we must have* $x.z \in L$.

But if $x.z \in L$ *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!*

add q *to both sides*

Step 1 *Let N be the no. of states of a DFA that accepts* L

Step 2 *Choose* $w = 1^m$ *where* m *is a prime number with* $m > N+1$.

Step 3 $w = 1^m = x.y.z$, *where* $|x.y| \le N$ *and* $|y| > 0$

Setting $r{:}=|x|$, $p{:}=|y|$ *and* $q{:}=|z|$ *it follows that* $m=|x.y.z|= r+p+q$, $r+p \le N$, $p > 0$

Step 4 *But according to* **PL** $x.y^j.z \in L$; *or* $r+p.j+q$ *must be a prime number for all integers* j. *Choosing* $j{:}=r+q$ *it follows that* $r+p.j+q = (r+q).(p+1)$ *must be a prime number !* *But* $r+p \le N$ *implies* $m \le N+q$ *and since by choice* $m > N+1$ *we have :* $N+1 < m \le N+q$ *and therefore* $q \ge 2$ *and thus* $r+q \ge 2$; *and since* $p+1 \ge 2$ *it follows that* $r+p.j+q = (r+q).(p+1)$ *cannot be a prime number for* $j=r+q$, *a contradiction !*

Example 4

$L = (s \in 1^* \mid s = 1^n, n = k^2, k > 1)$; **is non-regular!**

Step 1 *Let $N > 1$ be the no. of states of a DFA that accepts L_Q*

Step 2 *Choose $w = 1^m$ where $m = N^2$ hence $|w| = m = N^2 > N$.*

for $j=1$, $p+qj+r = m = N^2 = k_1^2$
hence $k_1 = N$

Step 3 $w = 1^m = x.y.z$, *where* $|x.y| \leq N$ *and* $|y| > 0$

Setting $p := |x|$, $q := |y|$ *and* $r := |z|$ *it follows that* $m = |x.y.z| = p+q+r = N^2$, $p+q \leq N$, $q > 0$

Step 4 *According to* **PL** $x.y^j.z \in L$; *or* $p+q.j+r = k_j^2$ *for each* $j > 1$ *where necessarily*

$N = k_1 < k_2 < \ldots$ *so that for any* $j > 1$, $k_j < k_{j+1} = k_j + s_j$ *for some* $s_j > 0$.

But $(p+q.(j+1)+r) - (p+q.j+r) = q = k_{j+1}^2 - k_j^2 = (k_j+s_j)^2 - k_j^2$

$(k_j+s_j)^2 - k_j^2 = (k_j^2 + 2k_j s_j + s_j^2) - k_j^2 = 2\underbrace{k_j}_{>N} \underbrace{s_j}_{>0} + s_j^2 > N$

whereas $LHS = q \leq N$ *contradicting the* **PL**.

Example 5 *(credits TA Alperen Doğan)*

$L = (s \in \{0,1\}^* \mid s=0^n 1^m;\ n \neq m)$ *is nonregular.*

Step 1 *Let N be the no. of states of a DFA that accepts L*

Note : $q \leq N$ implies $(N!/q)$ is an integer

Step 2 *Choose* $w = 0^N 1^{N+N!}$ *hence* $w \in L$ *and* $|w| > N$

Step 3 *By the PL* $w = x.y.z$, *where* $|x.y| \leq N$ *and* $y = 0^q$ *with* $q>0$

Step 4 *By the PL* $x.y^j.z \in L$ *for all* $j=0,1,\ldots$, *in particular for* $j = (N!/q) + 1 > 0$

So $x = 0^p$ *and* $z = 0^{N-p-q}\,1^{N+N!}$, $p+q \leq N$, *and*

$x\,y^j\,z = 0^p\,0^{jq}\,0^{N-p-q}\,1^{N+N!} = 0^{N+q(j-1)}\,1^{N+N!}$ *and substituting* $j = (N!/q) + 1$

$xy^jz = 0^{N+N!}\,1^{N+N!} \notin L$ *and PL is contradicted.*

Closure Properties of Regular Languages

(1) ***L,M*** *regular implies (i)* $L \cup M$*, (ii)* $L \,.\, M$ *and (iii)* L^* *are regular –*

follows from the definition of ***REs***

E+E ***E.E*** ***E****

(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* $L \cap M$ *is regular*

- *short path :* $L \cap M = (L^c \cup M^c)^c$*, which by (1) and (2) proves the result*
- *long path : If* ***A*** *and* ***B*** *are DFA that accept* ***L*** *and* ***M*** *then* $A \times B$ *is a DFA that accepts* $L \cap M$

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

$\delta((q,r), \sigma) := (\delta_A(q,\sigma), \delta_B(r,\sigma))$ $\quad \delta : (Q \times R) \times \Sigma \rightarrow 2^{(Q \times R)}$

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

initial state sets of A and B

$\delta E((Q_1 \times R_1), u) = (\delta_A E(Q_1, u) \times \delta_B E(R_1, u))$

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

$u \in L(A \times B) \Leftrightarrow \delta E((Q_0 \times R_0), u) \cap (F_A \times F_B) \neq \emptyset$

$\Leftrightarrow (\delta_A E(Q_0, u) \times \delta_B E(R_0, u)) \cap (F_A \times F_B) \neq \emptyset$

$\Leftrightarrow \delta_A E(Q_0, u) \cap F_A \neq \emptyset \wedge \delta_B E(R_0, u) \cap F_B \neq \emptyset$

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

The proof by induction on the length of ***u*** **:**

Basis $\delta E((Q_1 \times R_1), e) = Q_1 \times R_1 = (\delta_A E(Q_1, e) \times \delta_B E(R_1, e))$

Induction $\delta E((Q_1 \times R_1), u.a) = \cup_{(x,y) \in Z} (\delta_A(x,a) \times \delta_B(y,a))$

where $Z = \delta E((Q_1 \times R_1), u)$

but $\delta E((Q_1 \times R_1), u) = (\delta_A E(Q_1, u) \times \delta_B E(R_1, u))$ *by induction hypothesis*

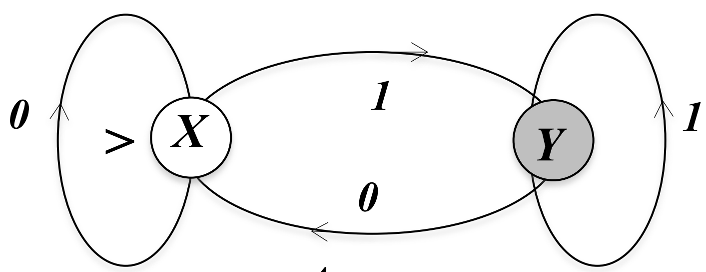
hence $Z = (\delta_A E(Q_1, u) \times \delta_B E(R_1, u))$

and so $\delta E((Q_1 \times R_1), u.a) =$

$\cup_{x \in \delta\_A\, E(Q\_1, u)} \delta_A(x, a) \times \cup_{y \in \delta\_B\, E(R\_1, u)} \delta_B(y, a)$

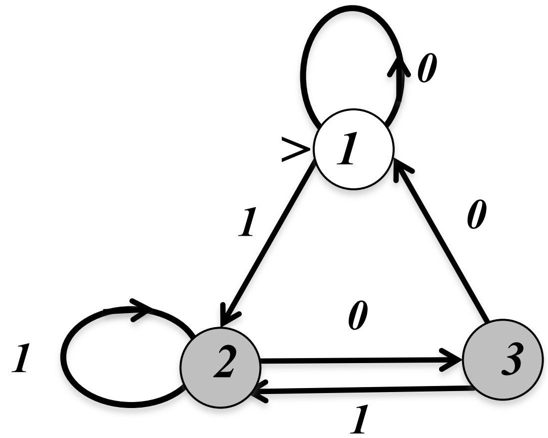
$= (\delta_A E(Q_1, u.a) \times \delta_B E(R_1, u.a))$

Example for the Product Automaton $A \times B$

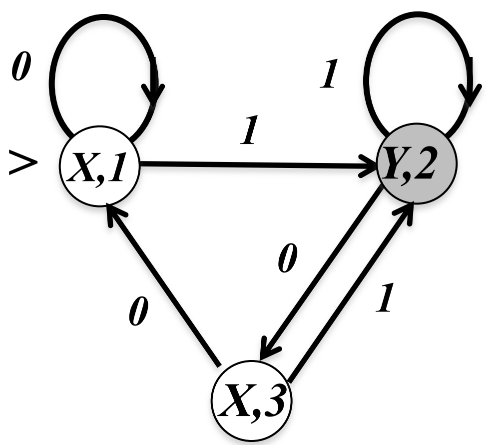


A

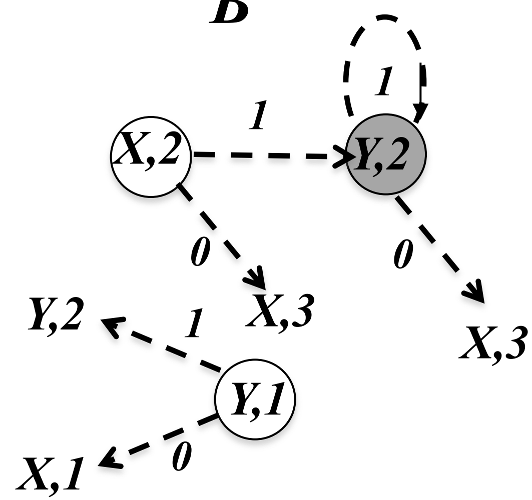
×



B



$A \times B$



$A \times B$ ***non-reachable …***

Typical Use of the Product Automata

(Simultaneously satisfy multiple constraints)

$\boldsymbol{L}$ *satisfies* $\textbf{Property}_1 \wedge \textbf{Property}_2 \wedge \ldots \wedge \textbf{Property}_N$

If $\boldsymbol{Y_j}$ *accepts the language satisfying* ***Property j*** *then the product automaton*

$\boldsymbol{Y = Y_1 \times Y_2 \times \ldots \times Y_N}$ *accepts the language* $\boldsymbol{L}$

Example *: Construct a DFA that accepts strings where*

(#0's - #1's) mod k = 0 *, for* ***k =3,5,7***

(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

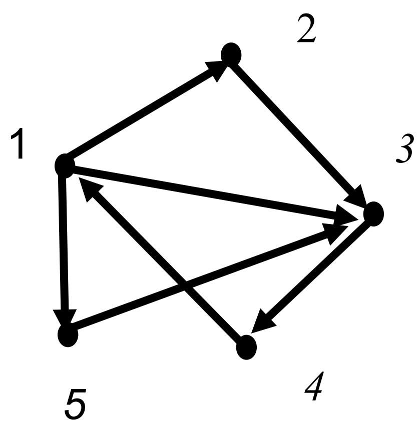
$= 0$ *otherwise*

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 : $w^k_{ij} = w^{k-1}_{ij} \vee (w^{k-1}_{ik} \wedge w^{k-1}_{kj})$, $k = 1,..,n$

Warshall at Work !



	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

add

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 row i to row j≠i for which (ji)th entry is 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

1,2,3

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*

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

Since for $n > 1$ *we have* $1 + 3/n + 7/n^2 + 5 \log(n)/n^3 < (1 + 3 + 7 + 5) = 16$

it follows using $\log(n) < n$ *for* $n > 1$ *that :* $g(n) < n^3 (1 + 3 + 7 + 5) = 16 n^3$

and therefore result follows with $K_1 = 16$ *and* $K_2 = 1$

Complexity of Warshall's Algorithm : $w^{k}_{ij} = w^{k-1}_{ij} \lor (w^{k-1}_{ik} \land w^{k-1}_{kj})$, $k = 1,..,n$

Two basic binary operations ('$\lor$' and '$\land$') for each $\mathbf{i,j}$ *and for* $\mathbf{k = 1, \ldots, n}$;

hence total $\mathbf{2 \times n^2 \times n = O(n^3)}$ *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* $\boldsymbol{\sigma}$

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) $\mathbf{i}$ *are the states (nodes) with value* $\mathbf{1}$ *in row* $\mathbf{i}$ *of* W_ε

(3) Epsilon closure of a set $\mathbf{S}$ *of nodes are the nodes with value* $\mathbf{1}$ *of the row obtained from the union ('$\lor$' = mod 2 addition) of the rows corresponding to the nodes in* $\mathbf{S}$ → $O(n^2)$

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.*

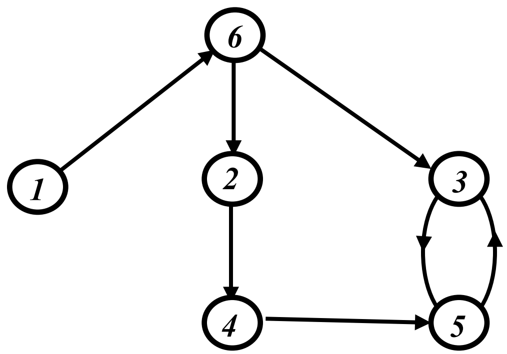
(2) Process the current (unused) index in the ***LIST*** *and mark the columns with a* ***1*** *at the row corresponding to the current index and ; add those column indices that are not already in the* ***LIST*** *to the* ***LIST.***

(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 = $O(n^2)$



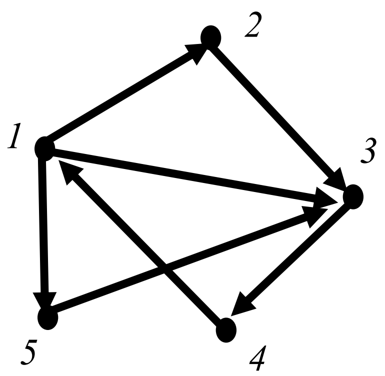
	1	2	3	4	5	6
1	1	0	0	0	0	1
2	0	1	0	1	0	0
3	0	0	1	0	1	0
4	0	0	0	1	1	0
5	0	0	1	0	1	0
6	0	1	1	0	0	1

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***

Hence reachable nodes from vertices ***2*** *and* ***6*** *is* ***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)

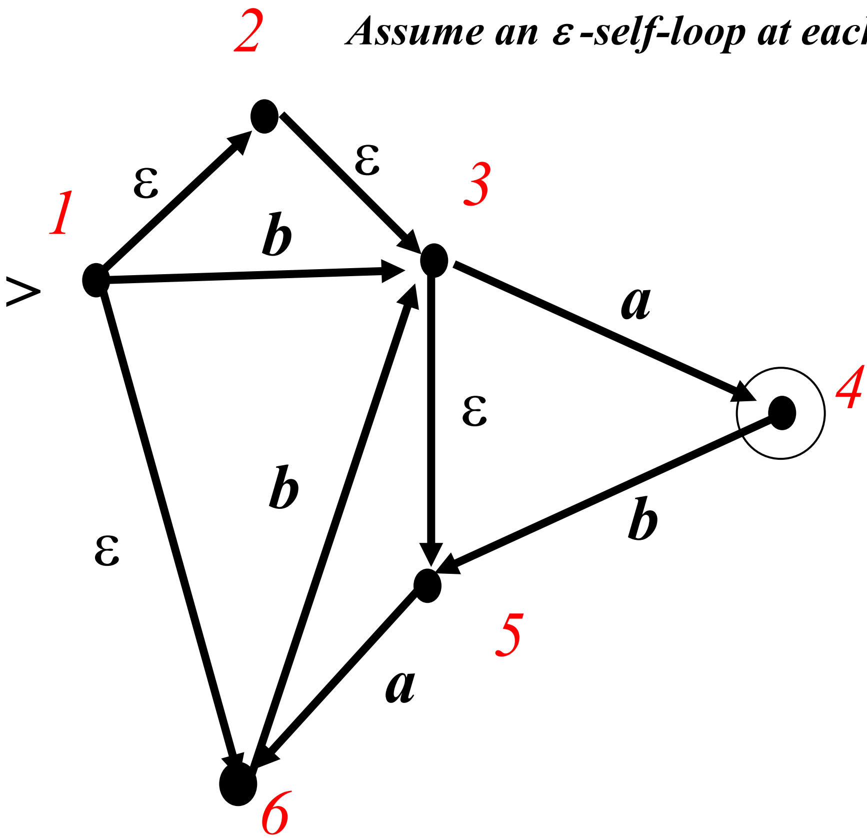
	1
1	1
4	1

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

empty matrix

reachability from vertices 2,5 = all the vertices

Assume an ε-self-loop at each node !



$A_{\varepsilon} =$

	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: *Transform* ***ε-NFA*** *to* ***equivalent 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

ε-closure of state 1

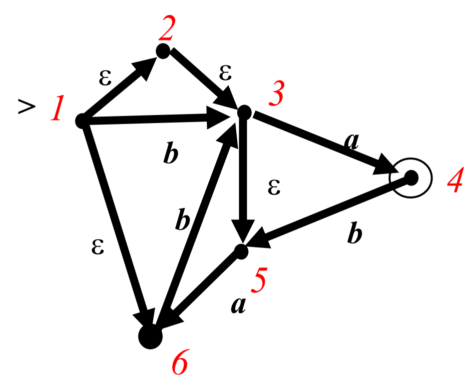
$W_\varepsilon \longrightarrow (1,2,3,5,6)$

after b-transition

$A_b \longrightarrow (3)$

ε-closure of state 3

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



Complexity of Regular Language Conversions

1- Add rows of A_σ according to initial epsilon-transitions and
2- add rows of Warshall for post-σ epsilon transitions

(1) ε**- NFA** to **NFA** $\rightarrow$ *First compute* $W_\varepsilon \rightarrow O(n^3)$

For each state and input: $O(n^2)+O(n^2)= O(n^2)$

Hence total effort = $n.|\Sigma|.O(n^2)+O(n^3) = O(|\Sigma|.n^3)$

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

(Exponential!)

complexity of RE = length of RE

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

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

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

$|R_{ij}^0| \rightarrow O(|\Sigma|)$; $R_{ij}^1 \rightarrow 4*O(|\Sigma|)$

$|R_{ij}^k| \rightarrow 4* |R_{ij}^{k-1}|$

(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 some final state is reached from the initial state set, **L** is non-empty. Complexity is* $O(n^2)$ *both for **NFA** and **DFA***

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

at each step

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

final state check

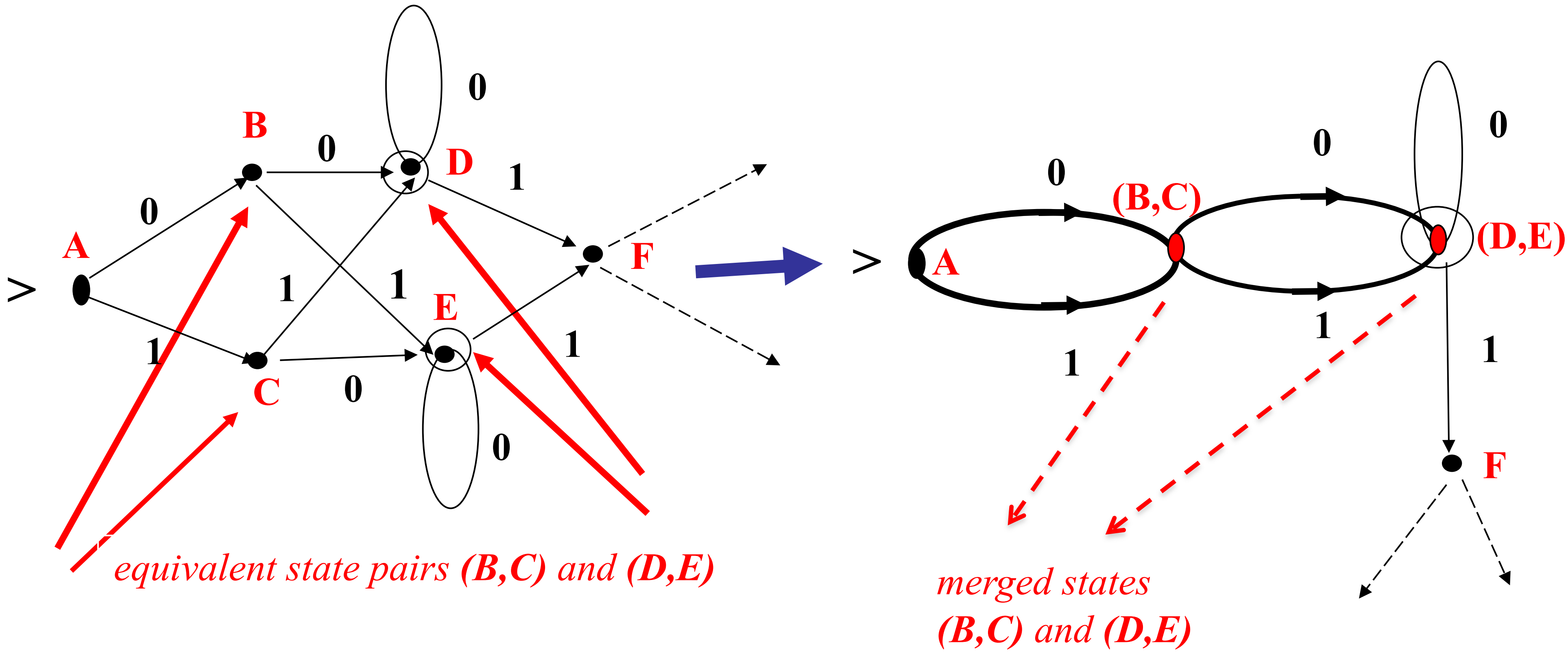
$O(|w|+n)$ *for **DFA** representation.*

Warshall

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

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

An informal description of State Equivalence



State Equivalence for DFA

$q \equiv q$ **identity**
$q \equiv r \Leftrightarrow r \equiv q$ **commutativity**
$q \equiv t \wedge t \equiv r \Rightarrow q \equiv r$ **transitivity**

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

Definition $q, r \in Q$ *are said to be* **equivalent** *states of* A*, shown by* $q \equiv r$ *, if* **for all** *inputs* $w \in \Sigma^* : \delta E(q,w) \in F$ **iff** $\delta E(r,w) \in F$

Exercise : Show !

States that are **NOT** **equivalent** *are called* **distinguishable**

More precisely $q, r \in Q$ *are said to be* **distinguishable** *states of* A *if* **there exists** *an input* $w \in \Sigma^* : \delta E(q,w) \in F$ **and** $\delta E(r,w) \notin F$

Definition $q, r \in Q$ *are said to be* **k-equivalent** *states of* A*, shown by* $q \equiv_k r$ *,*
if **for all** *inputs* $w \in \Sigma^*$ *with* $|w| \leq k : \delta E(q,w) \in F$ **iff** $\delta E(r,w) \in F$

Note that if $q, r \in Q$ *are* **NOT** *a* **k-equivalent** *pair of states they are* **NOT** **equivalent** *;*
moreover there is an input w *with* $|w| \leq k$ *such that* $\delta E(q,w) \in F$ **and** $\delta E(r,w) \notin F$
That is : they can be distinguished by an input of length no more than k *!*

Theorem *Given two states* $\boldsymbol{q}$ *and* $\boldsymbol{r}$ *of a DFA and* $\boldsymbol{k \geq 1 : q \equiv_k r} \Leftrightarrow$

for any $\boldsymbol{\sigma \in \Sigma}$, $\boldsymbol{\delta(q,\sigma) \equiv_{k-1} \delta(r,\sigma)}$

Proof of Theorem : $\Leftarrow$ *(if)*

For any $\boldsymbol{w}$ *with* $|\boldsymbol{w}| \leq \boldsymbol{k}$ *write it as* $\boldsymbol{w = \sigma . u}$ *where* $|\boldsymbol{u}| \leq \boldsymbol{k\text{-}1}$.

Then $\boldsymbol{\delta E(q, \sigma . u) = \delta E(\delta(q,\sigma),u)}$ *and* $\boldsymbol{\delta E(r, \sigma . u) = \delta E(\delta(r,\sigma),u)}$

and since $\boldsymbol{\delta(q,\sigma) \equiv_{k-1} \delta(r,\sigma)}$ **:** $\boldsymbol{\delta E(\delta(q,\sigma),u) \in F}$ ***iff*** $\boldsymbol{\delta E(\delta(r,\sigma),u) \in F}$

hence $\boldsymbol{\delta E(q, w) \in F}$ ***iff*** $\boldsymbol{\delta E(r,w) \in F}$ *where* $\boldsymbol{w=\sigma . u}$ **;** *so* $\boldsymbol{q \equiv_k r}$

$\Rightarrow$ ***(only if)***

For any $\boldsymbol{u}$ *and* $\boldsymbol{\sigma}$ *with* $|\boldsymbol{u}| \leq \boldsymbol{k\text{-}1}$ *set* $\boldsymbol{w := \sigma . u}$ *so that* $|\boldsymbol{w}| \leq \boldsymbol{k}$

Then by assumption $\boldsymbol{\delta E(q, w) \in F}$ ***iff*** $\boldsymbol{\delta E(r,w) \in F}$ *which implies*

$\boldsymbol{\delta E(\delta(q,\sigma),u) \in F}$ ***iff*** $\boldsymbol{\delta E(\delta(r,\sigma),u) \in F}$ *and so* $\boldsymbol{\delta(q,\sigma) \equiv_{k-1} \delta(r,\sigma)}$

Recursive computation of distinguishable state pairs

(Table Filling Algorithm)

Basis *: A pair* ***(r,q)*** *is distinguished by a string of* ***0*** *length if one is* ***accepting*** *(final) and the other is a* ***non-accepting*** *(non-final) state.*

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

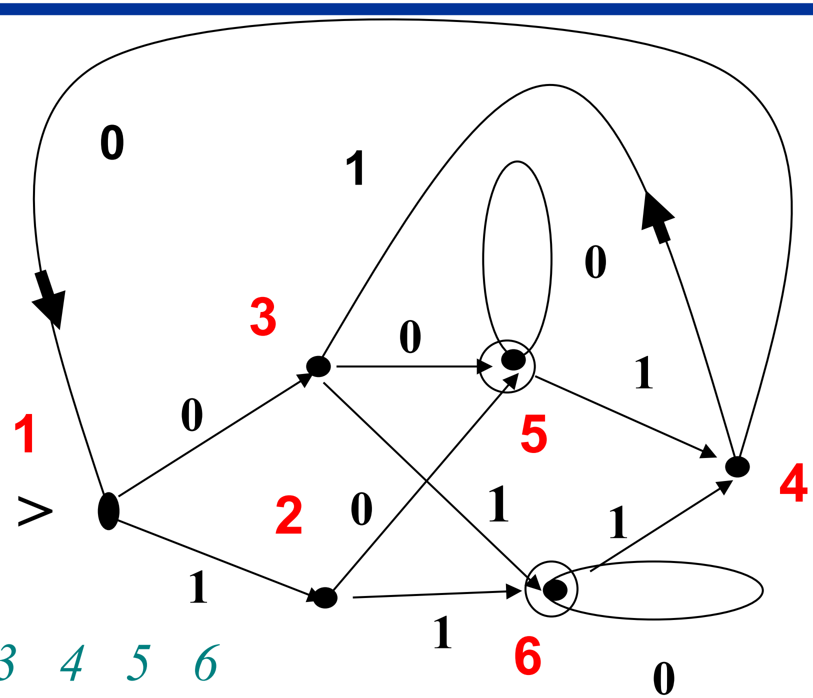
Proof *: Given that* $\boldsymbol{r'}$ *and* $\boldsymbol{q'}$ *are distinguishable (i.e.* ***NOT*** *equivalent), there exists an input string* ***u*** *such that* $\boldsymbol{\delta E(r', u) \in F}$ *and* $\boldsymbol{\delta E(q', u) \notin F}$ *(or vice-versa)* ***;***

But since (see ***Theorem*** *above)* $\boldsymbol{r' = \delta(r, a)}$ *and* $\boldsymbol{q' = \delta(q, a)}$ *we have*

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

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

Table Filling Algorithm



apply input 0

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

Run 0

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

Run 1

apply input 0

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

Run 2

apply input 0

No more improvement

Interpretation of the Table Filling Algorithm

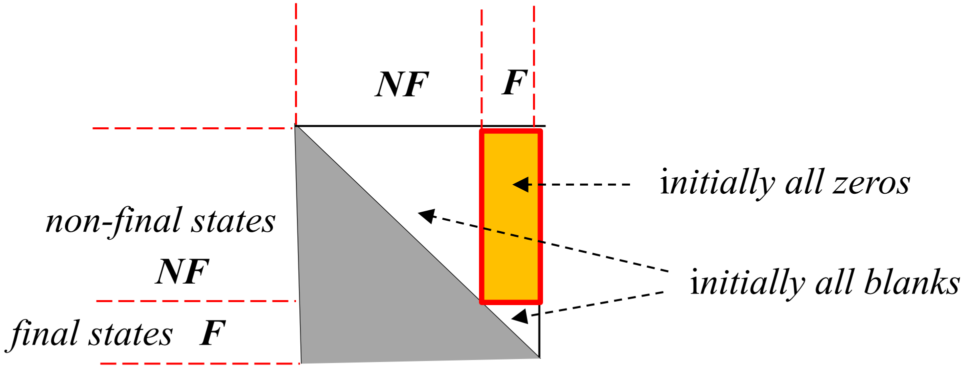
The table starts with squares ***{i,j}*** *marked with a* ***0*** *where states* $\boldsymbol{i \in Q\text{-}F}$ and $\boldsymbol{j \in F}$ *or vice-versa. This corresponds to* ***Run 0*** *where marked squares correspond to state pairs that are distinguished with the input of* ***zero*** *length, namely the empty string* ***e****.*

The unmarked squares correspond to state pairs that are ***0-equivalent*** **.**

The next run, ***Run 1****, searches for all unmarked state pairs* ***{k,m}*** *that can be moved to a* ***marked*** *state pair* ***{i, j}*** *of* ***Run 0*** *by some input. If this is possible for some input, then the unmarked pair* ***{k,m}*** *is marked with a* ***1*** *implying that the pair* ***{k,m}*** *is distinguishable by an input of length* ***1****. The pairs that remain unmarked in* ***Run 1*** *are clearly* ***1-equivalent*** *states. Generalizing : if at* ***Run J*** *an input is found that drives a blank square pair* ***{k,m}*** *to a marked square of* ***Run J-1*** *then it is marked with the integer* ***J*** *showing that pair* ***{k,m}*** *is distinguishable by an input of length* ***J.***

Clearly squares that remain blank at ***Run J*** *correspond to* ***J-equivalent*** *pairs*

Interpretation of the Table Filling Algorithm (Cont')



$i_{ms} \equiv_J m \wedge m \equiv_J i_{mt} \Rightarrow i_{ms} \equiv_J i_{mt}$

*1- If at **Run J** the corresponding columns $\mathbf{i_{m1}, i_{m2}}, \ldots, \mathbf{i_{mk}}$ at row **m** are **blank***

*then all the **state pairs** in the set $\{\mathbf{m, i_{m1}, i_{m2}, \ldots, i_{mk}}\}$ are **J-equivalent** .*

*2- For two rows **m** and **p** the sets $\{\mathbf{m, i_{m1}, i_{m2}, \ldots, i_{mk}}\}$ and $\{\mathbf{p, i_{p1}, i_{p2}, \ldots, i_{pl}}\}$ are either **disjoint** ; or restricted to blanks they are **identical** if the darkened regions of the table are also included.*

*3- The maximum value for the number of **runs** before the algorithm halts is no more than **K-1** where **K** = the total number of states.*

J^{th} RUN

Non – Final   Final

		k	m			
i			*n*		*0*	*0*
			x		*0*	*0*
p			*?*		*0*	*0*
	0	*0*	*0*	*0*		
	0	*0*	*0*	*0*		

Non – Final   Final

All Blank entries correspond to J-equivalent pairs

*number **n** at **ij^{th}** slot signify that states **i** and **j** can be distinguished in **n** steps*

*all diagonal entries are **blank** since **$q \equiv_J q$***

Fact : blank entries of rows are identical or disjoint

Suppose ik and pk are both blank yet im non-blank and pm blank → not possible !

$i \equiv_J p$ and $p \equiv_J m$ but $\neg(i \equiv_J m)$ violates transitivity !

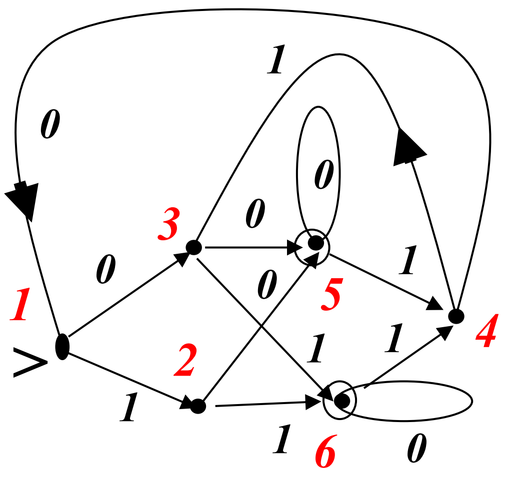
Interpretation of the Table Filling Algorithm (Cont')

1- Justification of statement 2 →

*2- For two rows **m** and **p** the sets $\{m, i_{m1}, i_{m2}, \ldots, i_{mk}\}$ and $\{p, i_{p1}, i_{p2}, \ldots, i_{pl}\}$ are either **disjoint** ; or restricted to blanks they are i**dentical** if the darkened regions of the table are also included.*

***Fact** : Suppose that $R_1 = \{m, i_{m1}, i_{m2}, \ldots, i_{mk}\}$ and $R_2 = \{p, i_{p1}, i_{p2}, \ldots, i_{pl}\}$ are two row sets of the table which are **not** disjoint.*

*Then R_1 and R_2 possess a common element **x** , which implies that **all** the blank entries of the rows **m** and **p** (including those in the darkened region) are identical and are **J-equivalent** to **x**. Finally at each run the state set is partitioned into a disjoint union of state sets some of which (those that are only equivalent to themselves) are singletones ; and at each run at least one blank square is filled and hence at least one set is splitted. Since the total number of splittings cannot exceed the total no.of states **-1** , it follows that total number of runs is smaller than **K-1** .*



use 0 or 1

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

Run 0

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

Run 1

By the ***identity law*** all diagonal entries are blank

use 0 or 1

use 0

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

Run 2

Run 0 : (1,2,3,4) ; (5,6)

Run 1 : (1,4) ; (2,3) ; (5,6)

*Run 2 : **(2,3)** ; **(5,6)** ; (1) ; (4)*

equivalent states

Full Table Pictures

	1	2	3	4	5	6	7
1						0	0
2						0	0
3						0	0
4						0	0
5						0	0
6	0	0	0	0	0		
7	0	0	0	0	0		

Run 0

	1	2	3	4	5	6	7
1				1	1	0	0
2				1	1	0	0
3				1	1	0	0
4	1	1	1			0	0
5	1	1	1			0	0
6	0	0	0	0	0		
7	0	0	0	0	0		

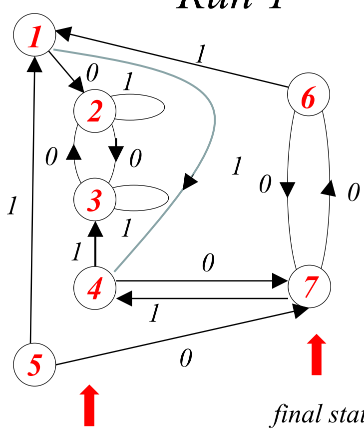
Run 1

	1	2	3	4	5	6	7
1		2	2	1	1	0	0
2	2			1	1	0	0
3	2			1	1	0	0
4	1	1	1			0	0
5	1	1	1			0	0
6	0	0	0	0	0		2
7	0	0	0	0	0	2	

Run 2

	1	2	3	4	5	6	7
1		2	2	1	1	0	0
2	2			1	1	0	0
3	2			1	1	0	0
4	1	1	1		3	0	0
5	1	1	1	3		0	0
6	0	0	0	0	0		2
7	0	0	0	0	0	2	

Run 3



non-final states

final states

Example

Run 0 : (1,2,3,4,5) ; (6,7)

Run 1 : (1,2,3) ; (4,5) ; (6,7)

Run 2 : (1);(2,3) ;(4,5);(6);(7)

Run 3 : (1);(2,3);(4);(5);(6);(7)

2 and 3 are equivalent states

$4,5 \xrightarrow{1} 3,1$

$3,1 \xrightarrow{1} 3,4$

$3,4 \xrightarrow{0} 2,7$

$4,5 \xrightarrow{110} 2,7$

Explanation of Table Filling Algorithm (Full Table Version)

*1- At **Run J** and row **m** of table the columns with **blank slots** correspond to **J-equivalent states .***

*2 - At all runs two **rows a** and **b** have **blank slots** that are (i) either **disjoint** ; or (ii) **identical** , in column numbers. If they are not disjoint they have a common column number which corresponds to a state that is **equivalent to all the remaining blank slots** in rows **a** and **b** and by **commutativity** and **transitivity** of an equivalence relation. Hence blank slots of rows **a** and **b** , if not disjoint are **identical**.*

*3 – At least one **J-equivalent** group at **Run J** shall split into **2** or more **J+1-equivalent** groups in **Run J+1** . This split reveals itself as different disjoint blank blocks in **Run J+1** whereas they were identical in **Run J** .*

***4 –** If all blank slots in **Run J+1** are same as those in **Run J ,** the algorithm halts at **Run J** and **J-equivalent** states corresponding to blank entries in **Run J** are in fact **equivalent states** .*

***5 –** At **Run 0** the states are split into **two 0-equivalence** groups : **final** and **non-final** states. At **Run 1** the number of equivalence groups are larger or equal to **3.** If the algorithm has not terminated then at **Run J** , number of **J-equivalent** groups ≥ **J +2** . Since the total number of splits cannot exceed the total number of states after **Run n-2** algorithm must halt.*

***6 –** If the data of **input used** (e.g. **0** or **1**) at each slot $(i_1 j_1)$ with the target $(i_2 j_2)$ are recorded at each slot then a minimum length distinguishing sequence for every **nonequivalent pair** can readily be computed from the algo.*

Complexity of the Table Filling Algorithm

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

*In **each run** and for **every** entry of the table inputs are searched whether a pair is already marked in the previous run . Call this **one** unit of computation. Since the table has* $\boldsymbol{O(n^2)}$ *number of entries the complexity involved is* $\boldsymbol{O(|\Sigma|\, n^2)}$ *for each run.*

*But because total no. of of runs is at most **n-1**, which is* $\boldsymbol{O(n)}$ *, complexity is* $\boldsymbol{O(|\Sigma|\, n^3)}$ **.**

*at most **n-1***

Run 0 = 2 pieces** ; **Run 1 ≥ 3 pieces** . . . **Run J ≥ J+2 pieces

Definition

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

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* $s \equiv t$.

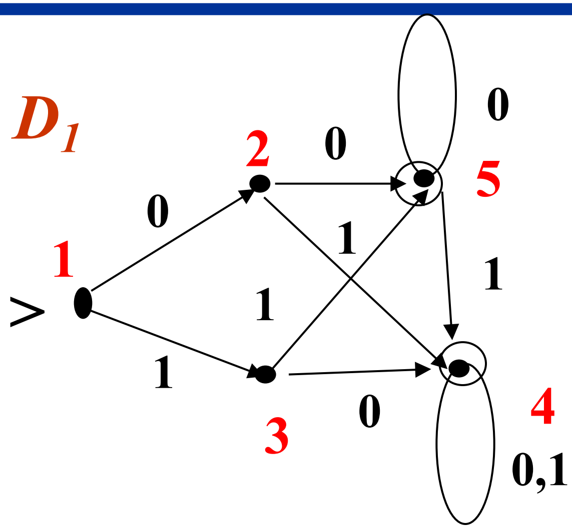
Corollary to the Definition

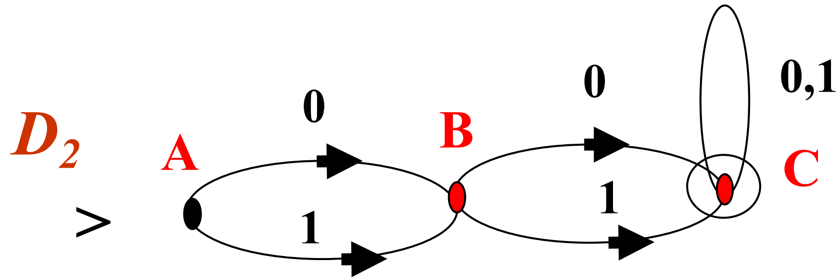
$A \equiv B$ ***if and only if*** $L(A) = L(B)$

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	A	B	4	5	C
1	1	1		1	0	0	0
2			1		0	0	0
3			1		0	0	0
A				1	0	0	0
B					0	0	0
4							
5							

(1,A),(2,3,B),(4,5,C)

Run 1

$1 \equiv A$

$D_1 \equiv D_2$

$L(D_1) = L(D_2)$

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 **DFAs**)*

Proof : *Call a pair **(p,q)** unfilled by Algo. a **bad** pair ; i.e. there is a shortest distinguishing input string **s** such that $\boldsymbol{\delta E(p,s)} \in \boldsymbol{F}$ and $\boldsymbol{\delta E(q,s)} \notin \boldsymbol{F}$ although Algo. terminated with **(p,q)** a blank square! Let $\boldsymbol{s = a_1 . a_2 \ldots a_J}$ and since it is the **shortest** distinguishing sequence for the **(p,q)** pair it follows that at **Run J-1** the **(p,q)** slot was a **blank** and $\boldsymbol{a_J}$ was the input that moved the **(p,q)** pair to a marked entry of **Run J-1** and thus make **s** a distinguishing sequence. Hence **(p,q)** at **Run J** is marked .*

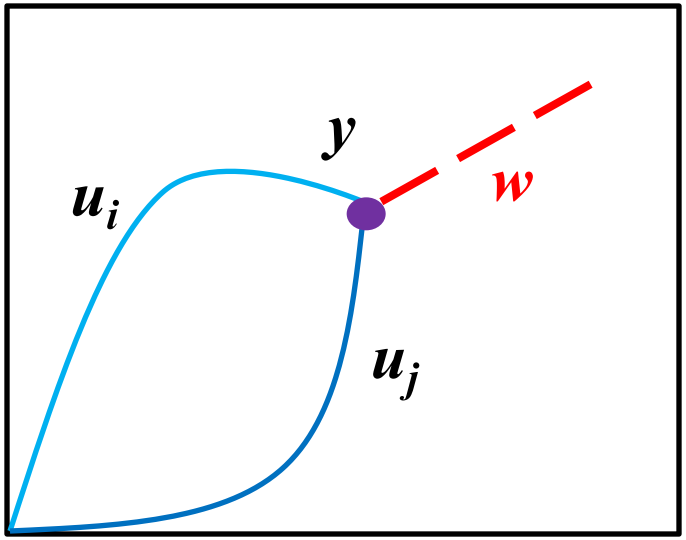
*A **minimum state machine** is obtained after all equivalent states are merged into common states thereby reducing the total number of states.*

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** ?*

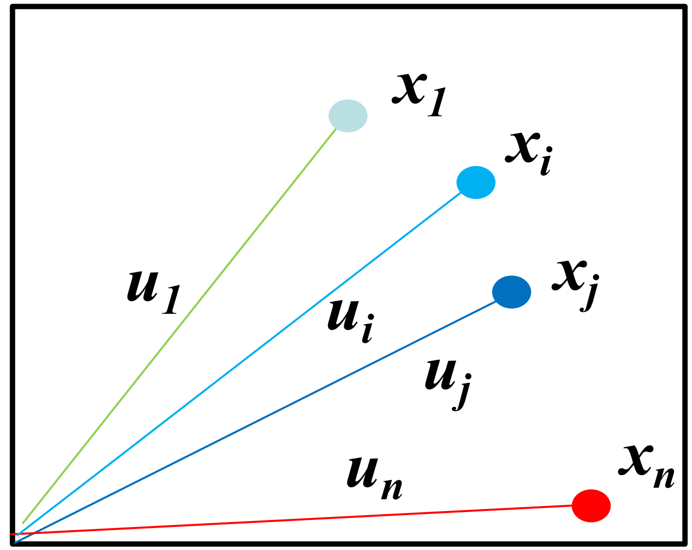
*Suppose you can ! then there is an **L**-accepting **DFA** , say **W**, with less states than that of **M** . Let u_1, …, u_n be strings that drive the initial state to the **n** distinct states of **M** . Apply these input strings to **W** and by pigeon hole at least 2 such strings will end up in an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

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



W = accepts L_M

#states of W < #states of M



M = minimal state machine

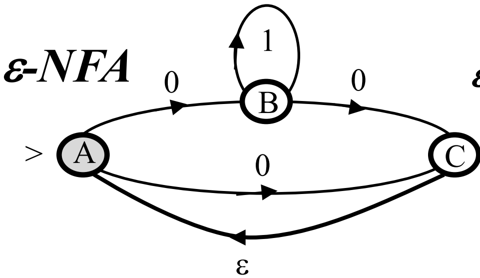
Apply $s = u_i.w$ *and* $t = u_j.w$ *to both* *Conclusion :* $x_i \equiv x_j$

Example

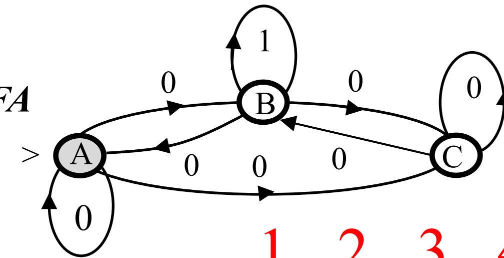
$E = (0.1^*.0+0)^*$

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

ε-NFA

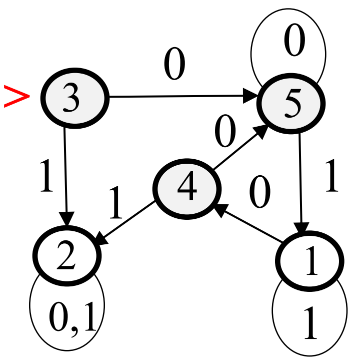


ε-NFA to NFA



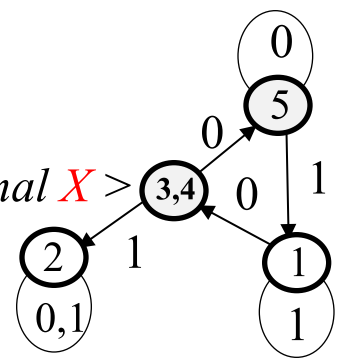
NFA to the DFA X

X >



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

Minimal X >



The Abstract Theory

Let $\boldsymbol{\Sigma}$ *be an alphabet and* $\boldsymbol{L} \subseteq \boldsymbol{\Sigma}^*$ *be a language*

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

$[u] := \{w \in \Sigma^* \mid w \equiv u\}$ *is called an* ***equivalence class*** *with* ***u*** *a* ***representative***

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

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

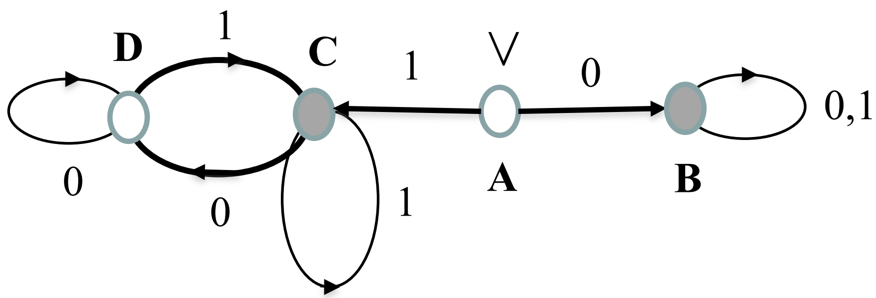
$\boldsymbol{Q}$ *= set of equivalence classes ;* $\boldsymbol{\delta([u],a) := [u.a]}$ ***; s = [e] = initial state***

$\boldsymbol{F = ([u] \in Q \mid u \in L\,)}$

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

(Myhill-Nerode theory)

Example 1 *States as language equivalence classes*



	A	D	B	C
A		1	0	0
D	1		0	0
B	0	0		1
C	0	0	1	

Minimal state DFA

$[e] = e$ *for* $A = L_A$

sets of strings terminating at states ***A,B,C*** *and* ***D***

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

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

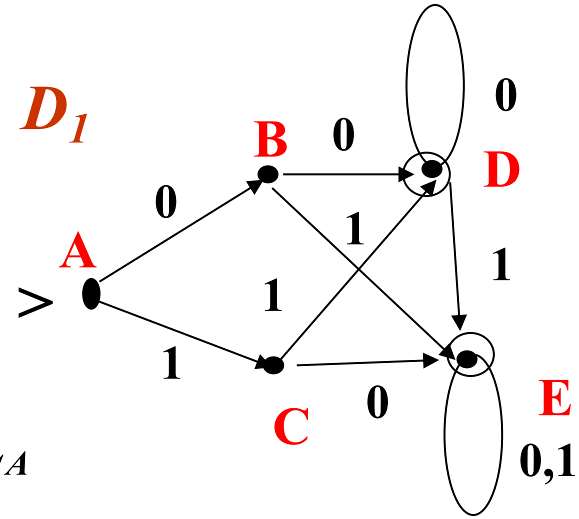
$[1.0] = 1^+(0^+.1^+)^*.0^+$ *for* $D = L_D$

L_A, L_B, L_C, L_D *are always* ***disjoint*** *sets (languages)* *Obvious !!*

Only in ***minimal state machines*** *they coincide with the* ***equivalence classes relative to one language!*** ***Namely :*** $L = [0]+[1] = L_B + L_C$

Example 2

D_1



	C	*B*	*D*	*E*
A	*1*	*1*	*0*	*0*
C			*0*	*0*
B			*0*	*0*
D				

B≡C** and **D≡E

$[e] = e$ *for* $A \rightarrow L_A$

strings terminating at $B = 0 = L_B$

strings terminating at $C = 1 = L_C$

strings terminating at $D = 0.0.0^* + 1.1.0^* = L_D$

strings terminating at $E = 1.0.(1+0)^* + 0.1.(1+0)^* + (0.0.0^* + 1.1.0^*).1.(1+0)^* = L_E$

$[0] = [1] = L_B + L_C$; $[0.0] = [1.1] = [0.1] = [1.0] = L_D + L_E$

L_A, $L_B + L_C$ *and* $L_D + L_E$ *are the* ***equivalent classes*** *relative to* $L_D + L_E$