Decision Properties of Regular Languages

- General Discussion of "Properties"
- The Pumping Lemma
- Membership, Emptiness, Etc.

Properties of Language Classes

- A *language class* is a set of languages.
  - We have one example: the regular languages.
  - We'll see many more in this class.
- Language classes have two important kinds of properties:
  1. Decision properties.
  2. Closure properties.

Representation of Languages

- Representations can be formal or informal.
- Example (formal): represent a language by a RE or DFA defining it.
- Example: (informal): a logical or prose statement about its strings:
  - $\{0^n1^n \mid n \text{ is a nonnegative integer}\}$
  - "The set of strings consisting of some number of 0's followed by the same number of 1's."

The Decision Properties

- A *decision property* for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.
- Example: Is language L empty?

Why Decision Properties?

- When we talked about protocols represented as DFA's, we noted that important properties of a good protocol were related to the language of the DFA.
- Example: "Does the protocol terminate?" = "Is the language finite?"
- Example: "Can the protocol fail?" = "Is the language nonempty?"

Why Decision Properties

- We might want a "smallest" representation for a language, e.g., a minimum-state DFA or a shortest RE.
- If you can't decide "Are these two languages the same?"
  - i.e., do two DFA's define the same language?

  You can't find a "smallest."

Closure Properties

- A *closure property* of a language class says that given languages in the class, an operator (e.g., union) produces another language in the same class.
- Example: the regular languages are obviously closed under union, concatenation, and (Kleene) closure.
  - Use the RE representation of languages.

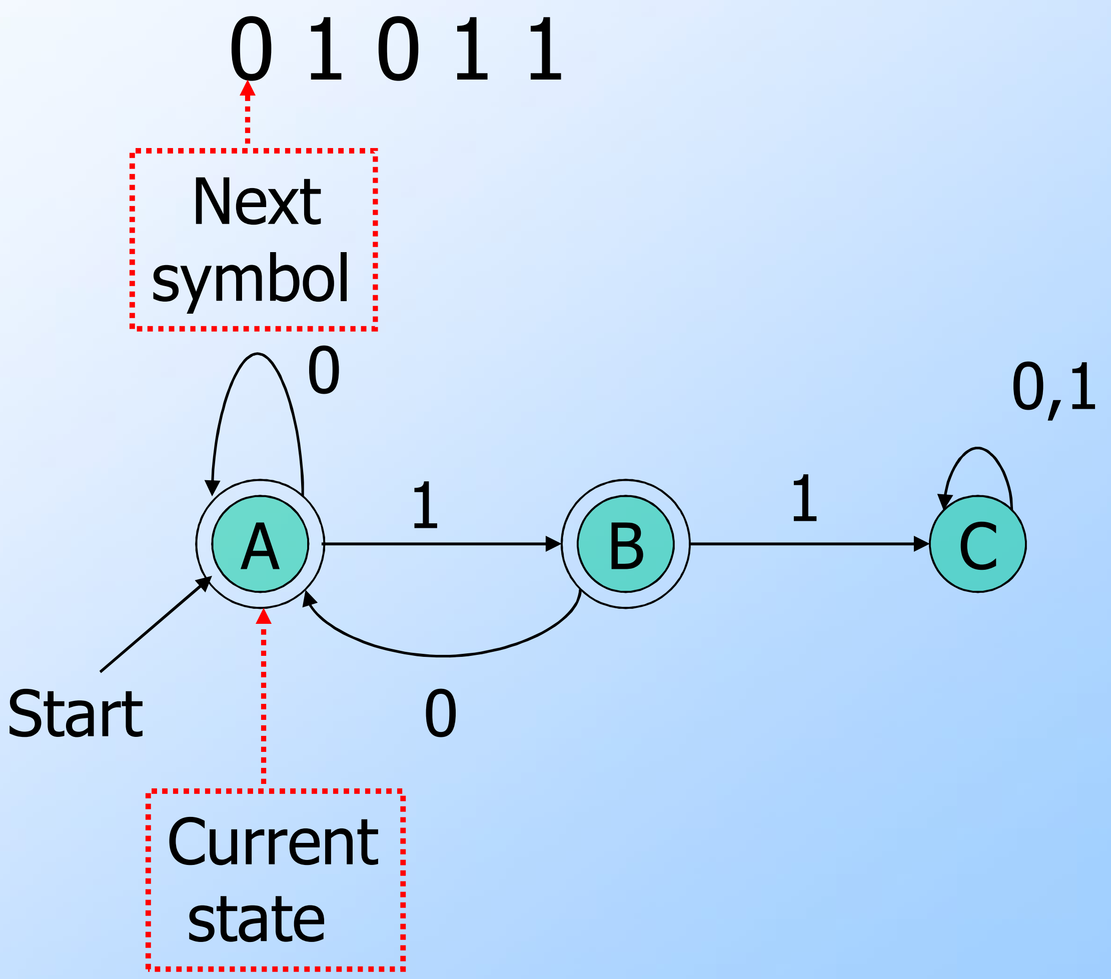
Why Closure Properties?

1. Helps construct representations of language. We can construct complex automata applying the closure properties of languages
2. Helps to show (informally described) languages not to be in the class , e.g Pumping Lemma – discussed later.

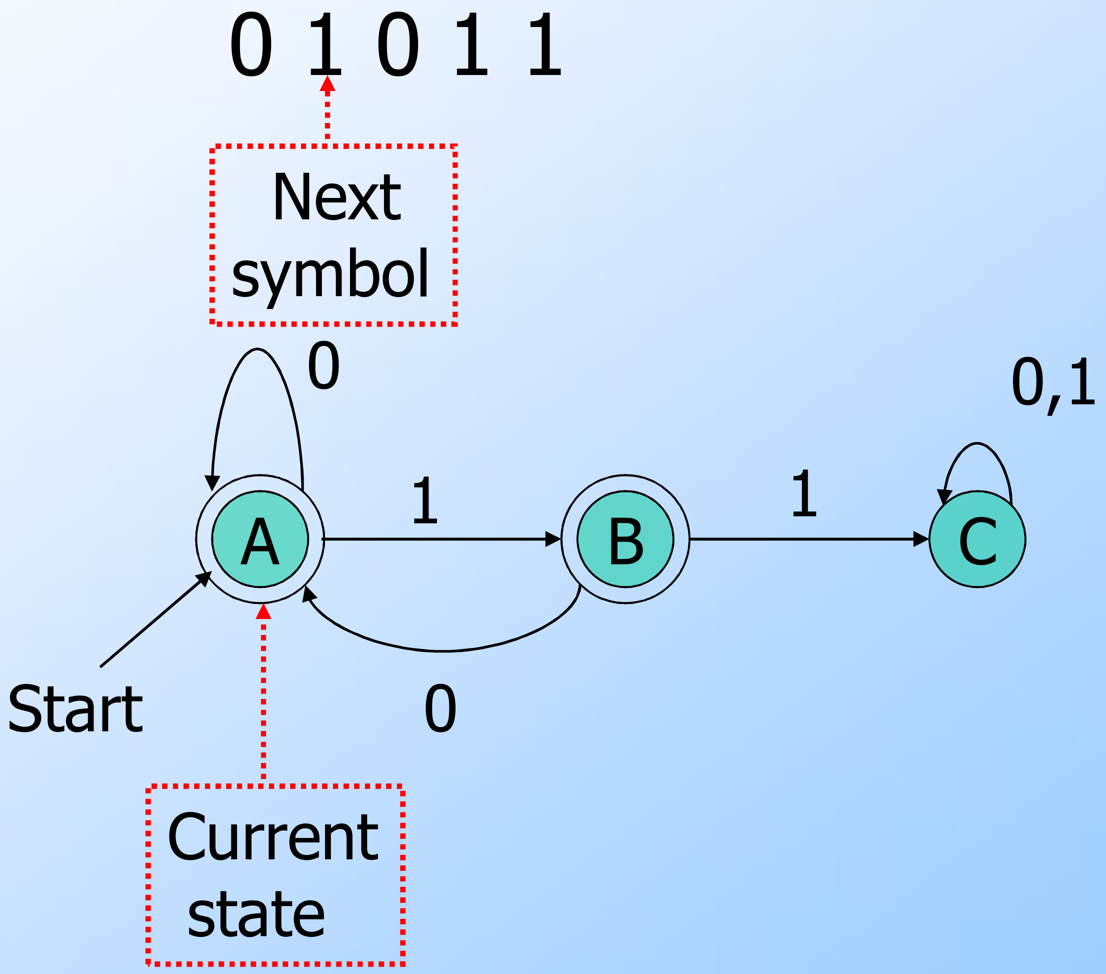
Decision Property: The Membership Question

- Our first decision property is the question: "Is string w in regular language L?"
- Assume L is represented by a DFA A.
- Simulate the action of A on the sequence of input symbols forming w.
- If DFA gives final state on input w, then the decision is Yes !

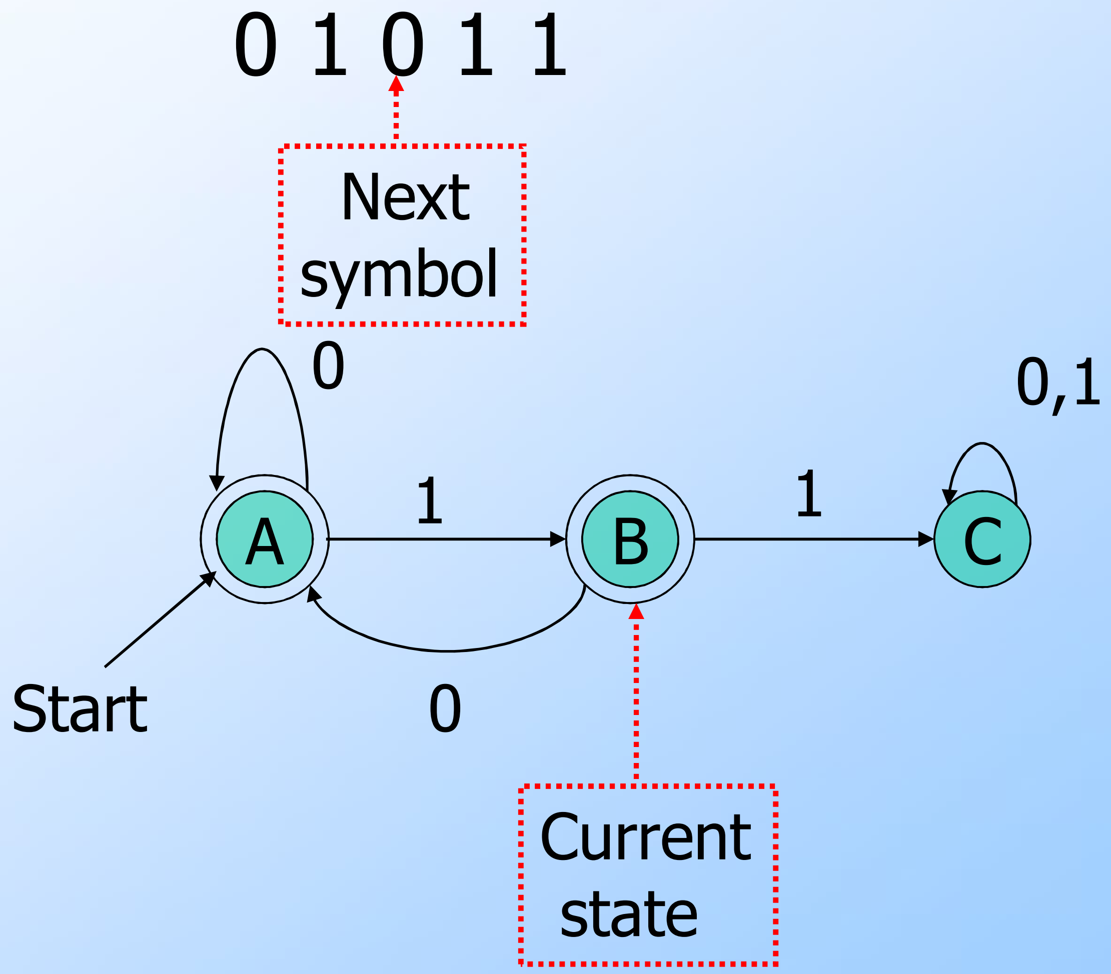
Example: Testing Membership



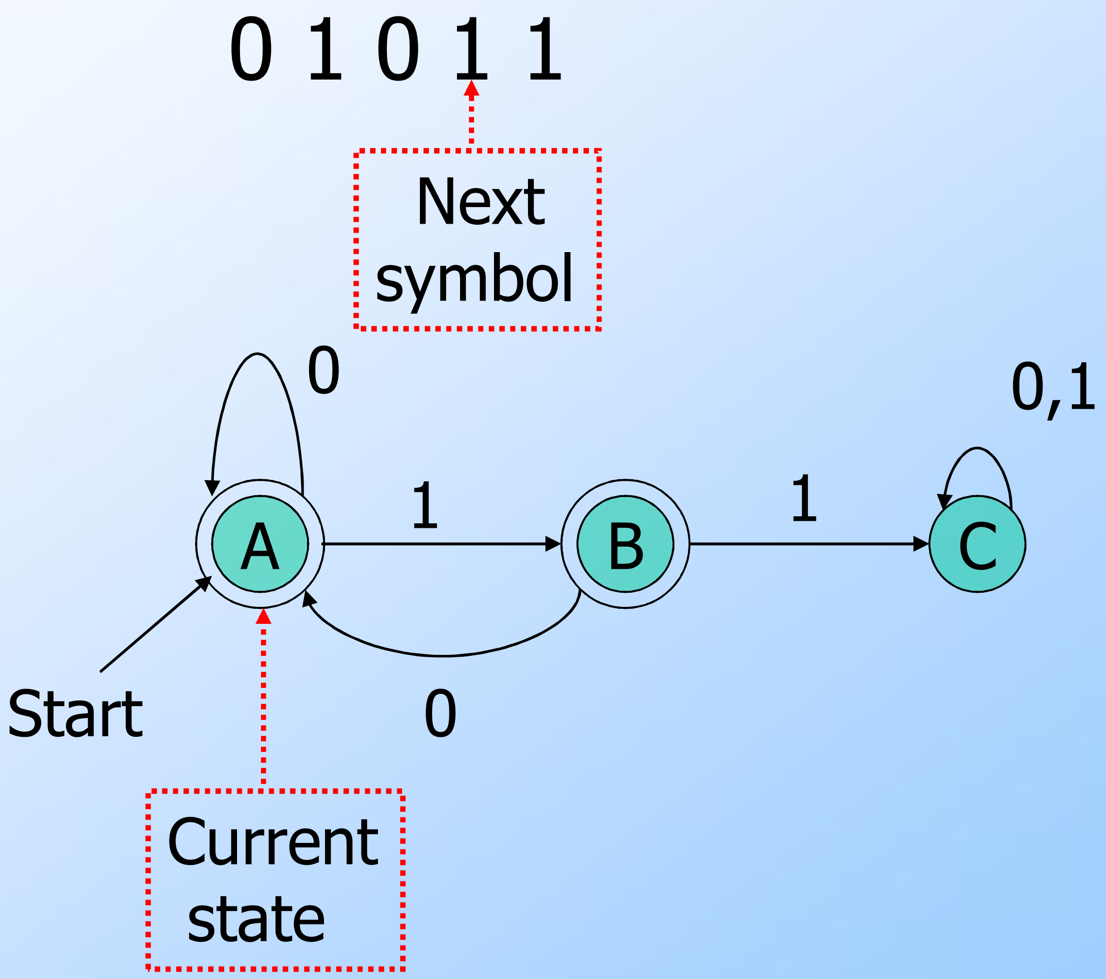
Example: Testing Membership



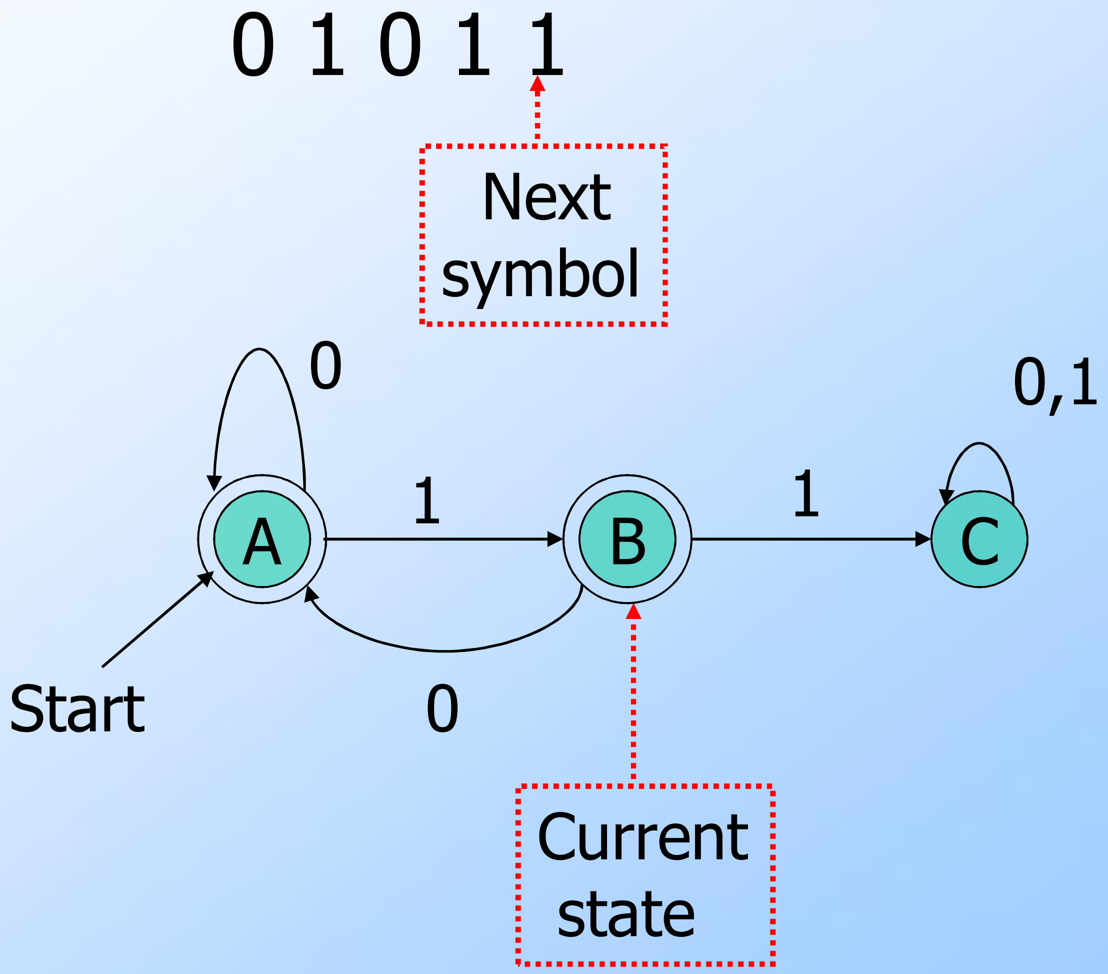
Example: Testing Membership



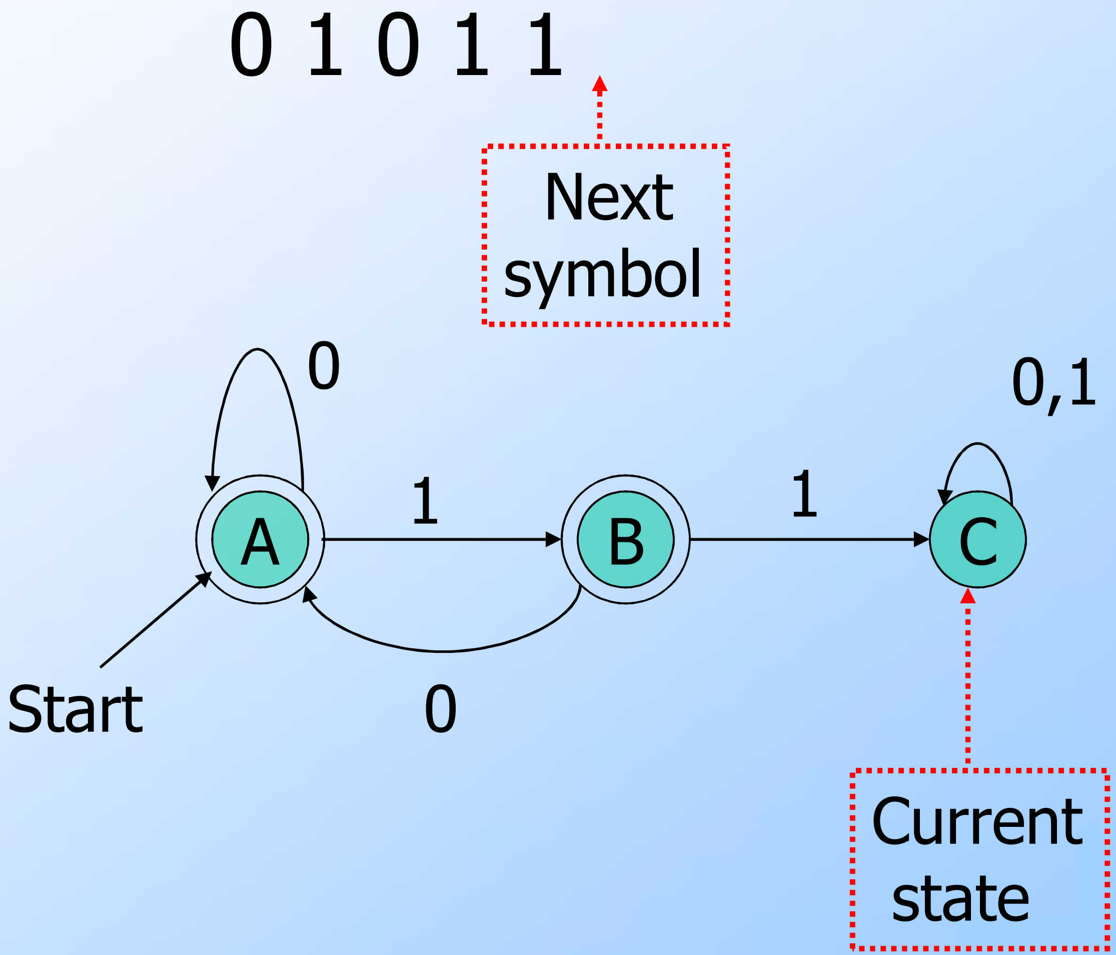
Example: Testing Membership



Example: Testing Membership



Example: Testing Membership



What if the Regular Language Is not Represented by a DFA?

- There is a circle of conversions from one form to another:

$$\begin{tikzcd}
& \boxed{\text{RE}} \arrow[dl, bend right] & \\
\boxed{\epsilon\text{-NFA}} \arrow[dr, bend right] & & \boxed{\text{DFA}} \arrow[ul, bend right] \\
& \boxed{\text{NFA}} \arrow[ur, bend right] &
\end{tikzcd}$$

Decision Property:The Emptiness Problem

- Given a regular language, does the language contain any string at all.
- Assume representation is DFA.
- Construct the transition graph.
- Compute the set of states reachable from the start state.
- If any final state is reachable, then yes(non-empty), else no(empty).

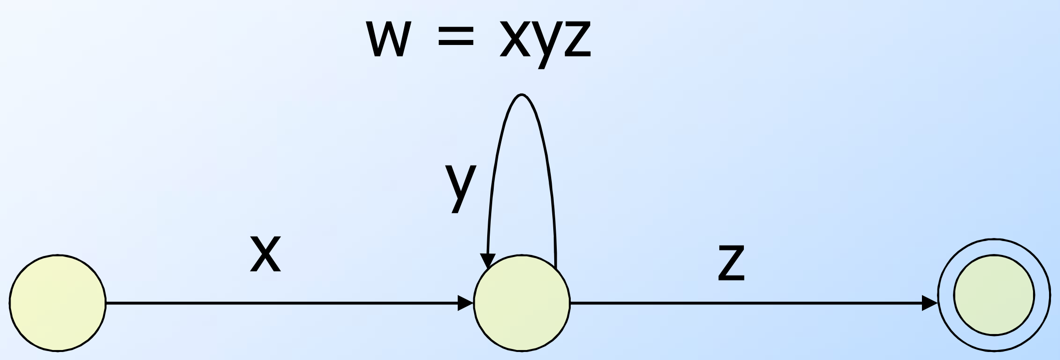
Decision Property: The Infiniteness Problem

- Is a given regular language infinite?
- Start with a DFA for the language.
- Key idea: if the DFA has n states, and the language contains any string of length n or more, then the language is infinite.
- Otherwise, the language is surely finite.
  - Limited to strings of length n or less.

Proof of Key Idea

- If an n-state DFA accepts a string w of length *n* or more, then there must be a state that appears twice on the path labeled w from the start state to a final state.
- Because there are at least n+1 states along the path.

Proof – (2)



Then xy^iz is in the language for all $i \geq 0$.

Since y is not ϵ, we see an infinite number of strings in L.

The Pumping Lemma

- The Pumping Lemma is quite useful for showing certain languages are not regular.

- Called the *pumping lemma for regular languages*.

Statement of the Pumping Lemma

For every regular language L

There is an integer n, such that

For every string w in L of length $\geq$ n

We can write w = xyz such that:

1. |xy| $\leq$ n.
2. |y| > 0.
3. For all k $\geq$ 0, xy^kz is in L.

Number of states of DFA for L

Labels along first cycle on path labeled w

Proof of Pumping Lemma

- Since L is regular language, it is accepted by a DFA D. Let us suppose D has n states.
- Since L is regular and w$\geq$ n is in L, then it is infinite.
- Let D accepts the string w of length n or greater. Let | w | = m$\geq$ n
- Let w=$a_1a_2a_3....a_m$ where each a_i in Σ be a string accepted by D.

Proof of Pumping Lemma

- Now Consider,
  - $\delta(q_0, w) = \delta(q_0, a_1a_2a_3....a_m)$
  - $= \quad \delta(q_1, a_2a_3....a_m)$
  - $= \delta(q_2, a_3....a_m)$
  - $= \dots \dots$
  - $= \delta(q_m, \epsilon)$
- where q_m will be final state and q_0 is initial state of D.

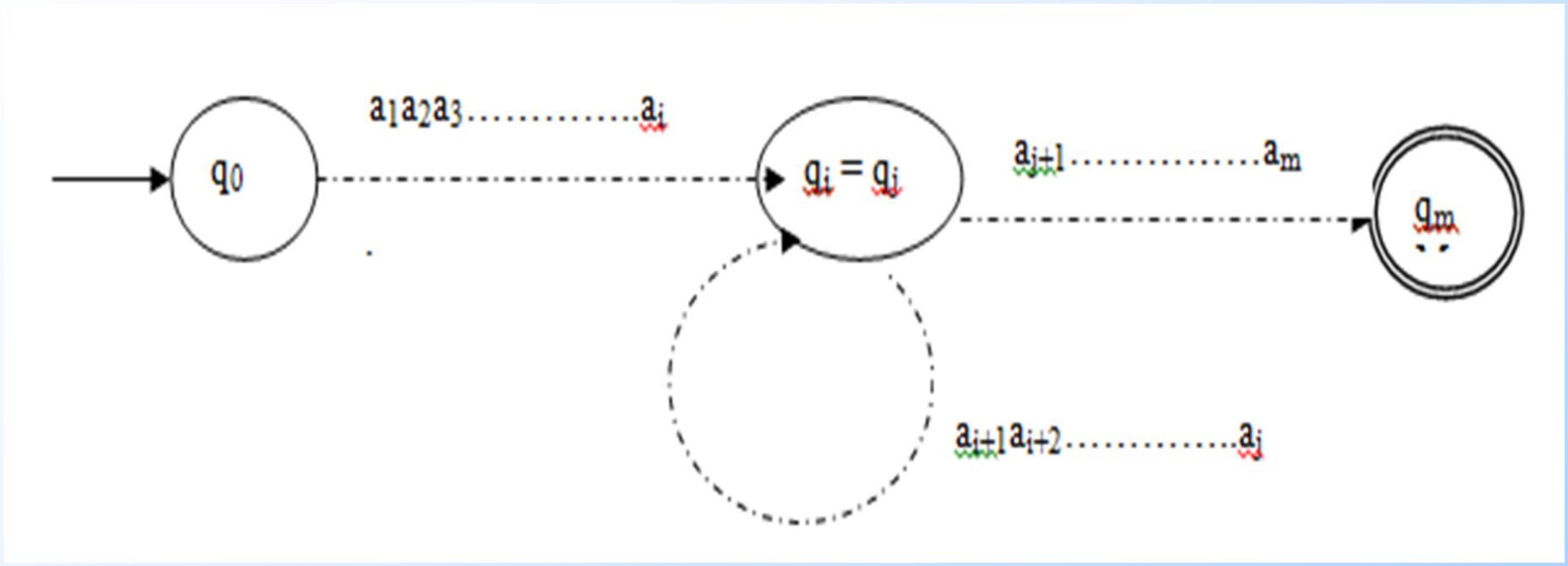
Proof of Pumping Lemma

- Since m $\geq$n and D has only n states, by pigeon hole principle there exist i and j , 0$\leq$ i<j$\leq$m such that $q_i = q_j$.
- Now we can break w = xyz as
  - $x= a_1a_2a_3.............a_i$
  - $y=a_{i+1}a_{i+2}.............a_j$
  - $z= a_{j+1}...............a_m$
  - i.e. String $a_{i+1}a_{i+2}.............a_j$ takes D from state q_i back to itself (since $q_i = q_j$)
- So we can say D accepts

$a_1a_2a_3...... a_i (a_{i+1}a_{i+2}........a_j)^k a_{j+1}...a_m$

for each k $\geq$0

Proof of Pumping Lemma


q_0
$a_1a_2a_3 \ldots\ldots\ldots a_i$
$q_i = q_j$
$a_{j+1} \ldots\ldots\ldots a_m$
q_m
$a_{i+1}a_{i+2} \ldots\ldots\ldots a_j$

- So we can say D accepts

$a_1a_2a_3\ldots\ldots a_i\,(a_{i+1}a_{i+2}\ldots\ldots\ldots a_j)^k\, a_{j+1}\ldots a_m$

for each $k \geq 0$

- **Hence xy^kz is in L , Proved**

Use of Pumping Lemma

- The main application of pumping lemma is to prove that a given language is not regular.
- It is a strong tool to do so.
- To show a language is not regular,
  - Assume the given language is regular
  - Based on the hypothesis, apply the pumping lemma for the string of language.
  - Surely , it will lead to a contradiction which will show that the language is not regular

Example: Use of Pumping Lemma

- Show that language $L = \{0^k 1^k \mid k \geq 1\}$ is not a regular language.

  - Suppose it is regular. Then there would be an associated n for the pumping lemma.
  - Let $w = 0^n 1^n$. We can write $w = xyz$, where $y \neq \epsilon$.
  - So by pumping lemma, xy^kz is in L

Example: Use of Pumping Lemma

<u>Case 1:</u> y consists of 0's only. Then

- $x = 0^p$, $y = 0^q$, $z = 0^r1^s$ where $p,r \geq 0$ and $q,s > 0$ such that $p+q+r = s$ Then,
- $xy^kz = 0^p(0^q)^k0^r1^s = 0^{p+kq+r}1^s$
- But only string in $0^{p+kq+r}1^s$ has equal number of 0's and 1's when k= 1, otherwise not. Hence y cannot contain 0's only.

Example: Use of Pumping Lemma

Case 2: y consists of 1's only. Then

- $x = 0^p 1^q$, $y = 1^r$, $z = 1^s$ where $q,s \geq 0$ and $p,r > 0$ such that $p=q+r+s$ Then,
- $xy^kz = 0^p1^q(1^r)^k1^s = 0^p1^{q+rk+s}$
- But only string in 0^p1^{q+rk+s} has equal number of 0's and 1's when k= 1, otherwise not. Hence y cannot contain 1's only.

Example: Use of Pumping Lemma

<u>Case 3:</u> y consists of 0’s and 1’s

- $x = 0^p$, $y = 0^q1^r$, $z = 1^s$ where q, r >0 such that p+q = r+s
- Now strung $xy^2z = 0^p0^q1^r0^q1^r1^s$ that is not in L, since 0 comes after 1
- In each case we got a contradiction.
- Hence L is not Regular.

Example-2

- **Example 2:** Show that language of palindrome over {0,1} is not regular.

Proof: Let L ={0^n10^n | n $\geq$0 } is palindrome language over {0,1}

- Suppose w = xyz such that y= 0^j (j>0) from the first 0^n part of w. Then,
  - x = 0^i , y=0^j and w = 10^n with n = i + j .
  - By pumping Lemma, we have xy^kz is in L
- If k = 0 , $xy^kz = 0^i (0^j)^k10^n = 0^{i+kj} 10^n = 0^i10^n$
- Since n=i+j and j>0 , 0^i10^n is not a member of L
- Only string in xy^kz is in L when k=1 otherwise not.
- String $0^{i+kj}10^n$ is not a member of L when k>1 also.
- Since L does not satisfies the pumping Lemma , L is not regular.

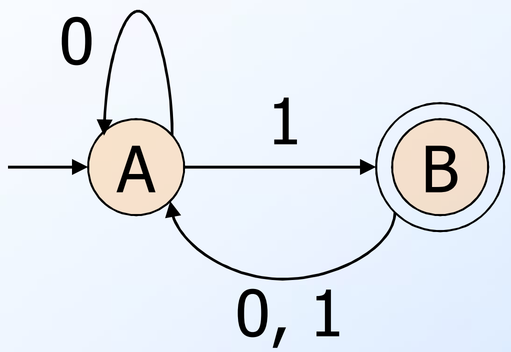
Decision Property: Equivalence

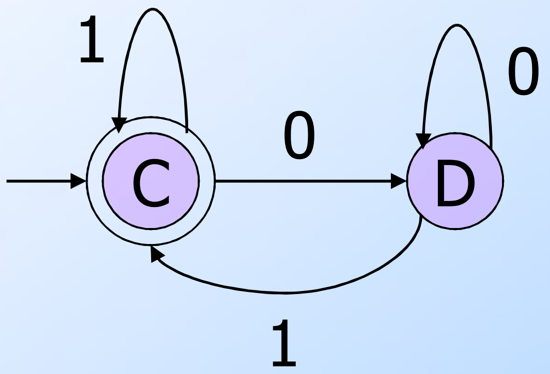
- Given regular languages L and M, is L = M?
- Algorithm involves constructing the *product DFA* from DFA's for L and M.
- Let these DFA's have sets of states Q and R, respectively.
- Product DFA has set of states Q × R.
  - I.e., pairs [q, r] with q in Q, r in R.

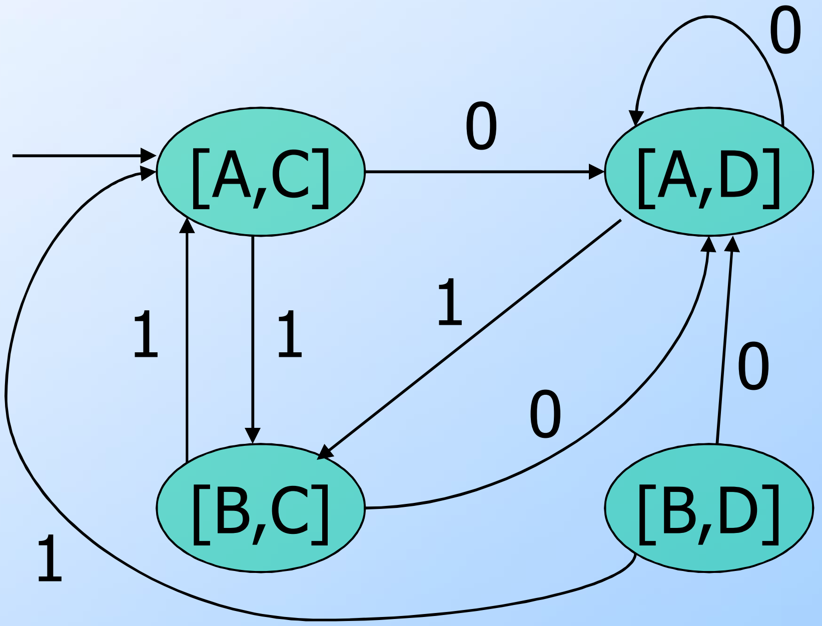
Product DFA – Continued

- Start state = $[q_0, r_0]$ (the start states of the DFA's for L, M).
- Transitions: $\delta([q,r], a) = [\delta_L(q,a), \delta_M(r,a)]$
  - δ_L, δ_M are the transition functions for the DFA's of L, M.
  - That is, we simulate the two DFA's in the two state components of the product DFA.

Example: Product DFA



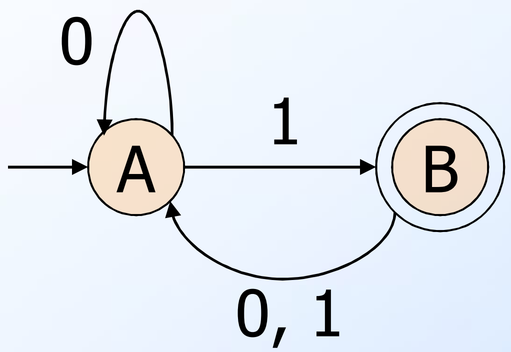


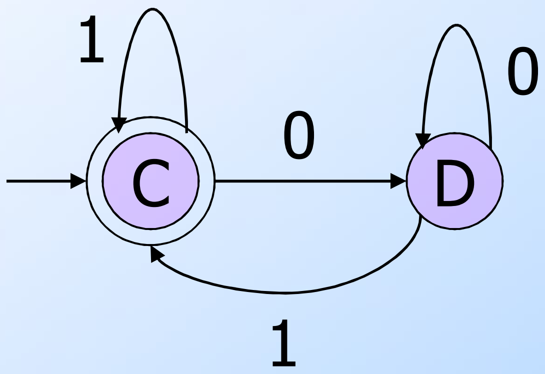


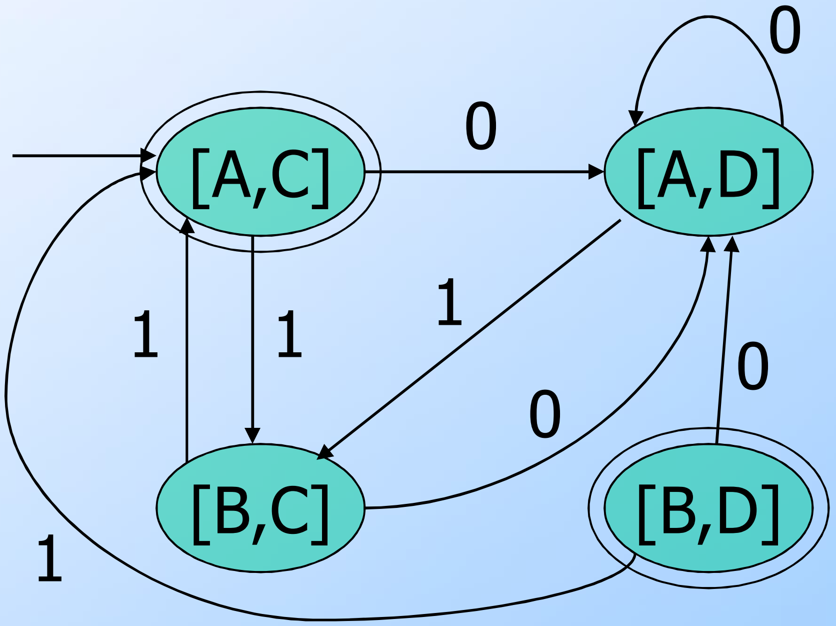
Equivalence Algorithm

- Make the final states of the product DFA be those states [q, r] such that exactly one of q and r is a final state of its own DFA.
- Thus, the product accepts w iff w is in exactly one of L and M.

Example: Equivalence







The product DFA's language is empty iff L = M.

The Minimum-State DFA for a Regular Language

- In principle, since we can test for equivalence of DFA's we can, given a DFA *A* find the DFA with the fewest states accepting L(A).
- Test all possible smaller DFA's for equivalence with *A*.
- But that's a terrible algorithm.

Efficient State Minimization: The Table Filling Algorithm

- Construct a table with all pairs of states.
- Find equivalence and distinguishable pairs in the table.
- If you find a string that *distinguishes* two states (takes exactly one to an accepting state and another non accepting state), mark that pair.
- Algorithm is a recursion on the length of the shortest distinguishing string.

DFA State Minimization -continue

- Basis: Initially, mark a pair if exactly one is a final state and another is not.
- Induction: mark [q, r] if there is some input symbol *a* such that [δ(q,a), δ(r,a)] is marked.
- After no more marks are possible, the unmarked pairs are equivalent and can be merged into one state.

Transitivity of "Indistinguishable"

- If state p is indistinguishable from q, and q is indistinguishable from r, then p is indistinguishable from r.
- Proof: The outcome (accept or don't) of p and q on input w is the same, and the outcome of q and r on w is the same, then likewise the outcome of p and r.

Constructing the Minimum-State DFA

- Suppose $q_1, \ldots, q_k$ are indistinguishable states.
- Replace them by one state q.
- Then $\delta(q_1, a), \ldots, \delta(q_k, a)$ are all indistinguishable states.
  - Key point: otherwise, we should have marked at least one more pair.
- Let $\delta(q, a)$ = the representative state for that group.

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

Here it is a DFA Representing in Transition Table

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B							
C	x	x					
D			x				
E			x				
F			x				
G			x				
H			x				
	A	**B**	**C**	**D**	**E**	**F**	**G**

Initially, mark the state pairs corresponding to Final state and non-final state: table above

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D			x				
E			x				
F			x				
G			x				
H			x				
	A	B	C	D	E	F	G

- **For pair (A,B)**
  - **on input 0 gives (B,G) not marked**
  - **On input 1, gives (F,C) already marked so mark (A,B)**

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D	x		x				
E			x				
F			x				
G			x				
H			x				
	A	**B**	**C**	**D**	**E**	**F**	**G**

- **For pair (A,D)**
  - **on input 0 gives (B,C) already marked so mark (A,D)**

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D	x		x				
E			x				
F			x				
G			x				
H			x				
	A	B	C	D	E	F	G

- **For pair (A,E)**
  - **on input 0 gives (B,H) not marked**
  - **On input 1 gives (F,F) undefined so (A,E) not marked now**

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D	x		x				
E			x				
F	x		x				
G			x				
H			x				
	A	B	C	D	E	F	G

- **For pair (A,F)**
  - **on input 0 gives (B,C) already marked so (A,F) is marked now**

Example: State Minimization

	0	1
→ A	B	F
B	G	C
* C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

B	x						
C	x	x					
D	x		x				
E			x				
F	x		x				
G			x				
H			x				
	A	B	C	D	E	F	G

- **For pair (A,G)**
  - **on input 0 gives (B,G) not marked yet.**
  - **On input 1 gives (F,E) undefined so (A,G) not marked now**

Example: State Minimization

	0	1
→ A	B	F
B	G	C
* C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

B	x						
C	x	x					
D	x		x				
E			x				
F	x		x				
G			x				
H	x		x				
	A	B	C	D	E	F	G

- **For pair (A,H)**
  - **on input 0 gives (B,G) not marked**
  - **On input 1 gives (F,C) Already marked so (A,H) is marked now**

Example: State Minimization

	0	1
→ A	B	F
B	G	C
* C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

B	x						
C	x	x					
D	x	x	x				
E		x	x				
F	x	x	x				
G		x	x				
H	x		x				
	A	B	C	D	E	F	G

- **Similarly Look at Pair (B,D), (B,E),(B,F),(B,G),(B,H)- mark (B,D) and (B,F) for input 0 and mark (B,E) and (B,G) for input 1**

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x				
G		x	x	x			
H	x		x	x			
	A	**B**	**C**	**D**	**E**	**F**	**G**

- **Similarly Look at Pair (D,E), (D,F),(D,G),(D,H)- mark (D,E), (D,G),(D,H) but not (D,F)**

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G		x	x	x	x		
H	x		x	x	x		
	A	**B**	**C**	**D**	**E**	**F**	**G**

- **Similarly Look at Pair (E,F),(E,G),(E,H)- mark ALL**

Example: State Minimization

	0	1
→ A	B	F
B	G	C
* C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G		x	x	x	x	x	
H	x		x	x	x	x	
	A	B	C	D	E	F	G

- **Similarly Look at Pair (F,G),(F,H)- mark ALL**

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G		x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

- **Similarly Look at Pair (G,H)- mark it, Now one cycle complete.**

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G		x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

- **Now 2nd Cycle,**
- **Similarly Look at Pair (A,E)- Not marked.**

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	**B**	**C**	**D**	**E**	**F**	**G**

- **Similarly Look at Pair (A,G)- marked for input 1.**

Example: State Minimization

	0	1
→ A	B	F
B	G	C
* C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

- **Similarly Look at Pair (B,H)- remains unmarked**
- **Similarly Look at pair (D,F) – remains unmarked**

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

Now cycle 3,

- **Look at Pair (A,E)- remains unmarked**
- **Similarly for Pair (B,H) – remains unmarked**
- **Similarly Look at pair (D,F) – remains unmarked**

Example: State Minimization

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

B	x						
C	x	x					
D	x	x	x				
E	🟩	x	x	x			
F	x	x	x	🟩	x		
G	x	x	x	x	x	x	
H	x	🟩	x	x	x	x	x
	A	B	C	D	E	F	G

No any remaining pair marked in this cycle. So the state pair unmarked till now are equivalent pairs.

Example: State Minimization

Hence we got A ≡ E , B ≡ H and D ≡ F.
Combining equivalent states, the minimized DFA is given below

		0	1
→	A	B	F
	B	G	C
*	C	A	C
	D	C	G
	E	H	F
	F	C	G
	G	G	E
	H	G	C

		0	1
→	[AE]	[BH]	[DF]
	[BH]	G	C
*	C	[AE]	C
	[DF]	C	G
	G	G	[AE]

Eliminating Unreachable States

- Unfortunately, combining equivalent states could leave us with unreachable states in the "minimum-state" DFA.
- Thus, before or after, remove states that are not reachable from the start state.