

# Course 8

## Properties of Regular Languages



---

The structure and the content of the lecture is based on <http://www.eecs.wsu.edu/~ananth/CptS317/Lectures/index.htm>



# Topics

---

- 1) How to prove whether a given language is not regular?
- 2) Minimization of DFAs



# Some languages are *not* regular

---

When is a language is regular?

if we are able to construct one of the following: DFA *or* NFA *or*  $\epsilon$ -NFA *or* regular expression

When is it not?

If we can show that no FA can be built for a language



# How to prove languages are *not* regular?

---

What if we cannot come up with any FA?

- A) Can it be language that is not regular?
- B) Or is it that we tried wrong approaches?

How do we *decisively* prove that a language is not regular?

*“The hardest thing of all is to find a black cat in a dark room, especially if there is no cat!” -Confucius*



# Example of a non-regular language


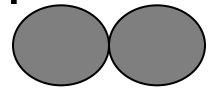
---

Let  $L = \{w \mid w \text{ is of the form } 0^n 1^n, \text{ for all } n \geq 0\}$

- Hypothesis:  $L$  is not regular
- Intuitive rationale: How do you keep track of a running count in an FA?
- A more formal rationale:
  - By contradiction, if  $L$  is regular then there should exist a DFA for  $L$ .
  - Let  $k = \text{number of states in that DFA}$ .
  - Consider the special word  $w = 0^k 1^k \Rightarrow w \in L$
  - DFA is in some state  $p_i$ , after consuming the first  $i$  symbols in  $w$



# Rationale...

- Let  $\{p_0, p_1, \dots, p_k\}$  be the sequence of states that the DFA should have visited after consuming the first  $k$  symbols in  $w$  which is  $0^k$
- But there are only  $k$  states in the DFA!
- $\implies$  at least one state should repeat somewhere along the path (by  +  Principle)
- $\implies$  Let the repeating state be  $p_i = p_j$  for  $i < j$
- $\implies$  We can fool the DFA by inputting  $0^{(k-(j-i))}1^k$  and still get it to accept (note:  $k-(j-i)$  is at most  $k-1$ ).
- $\implies$  DFA accepts strings  $w$  with unequal number of 0s and 1s, implying that the DFA is wrong!



# The Pumping Lemma for Regular Languages



---

## What it is?

The Pumping Lemma is a property of all regular languages.

## How is it used?

A technique that is used to show that a given language is **not** regular.

It can not be used to show that a given language is regular.



# Pumping Lemma for Regular Languages

Let  $L$  be a regular language

Then there exists some constant  $N$  such that for every string  $w \in L$  s.t.  $|w| \geq N$ , there exists a way to break  $w$  into three parts,  $w = xyz$ , such that:

1.  $y \neq \varepsilon$
2.  $|xy| \leq N$
3. For all  $k \geq 0$ , all strings of the form  $xy^kz \in L$

This property should hold for all regular languages.

**Definition:**  $N$  is called the “Pumping Lemma Constant”





# Pumping Lemma: Proof

- $L$  is regular  $\Rightarrow$  it should have a DFA.
  - Set  $N :=$  number of states in the DFA
- Any string  $w \in L$ , s.t.  $|w| \geq N$ , should have the form:  $w = a_1 a_2 \dots a_m$ , where  $m \geq N$
- Let the states traversed after reading the first  $N$  symbols be:  $\{p_0, p_1, \dots, p_N\}$ 
  - $\Rightarrow$  There are  $N+1$   $p$ -states, while there are only  $N$  DFA states
  - $\Rightarrow$  at least one state has to repeat  
i.e,  $p_i = p_j$  where  $0 \leq i < j \leq N$  (by PHP)

# Pumping Lemma: Proof...

- $\Rightarrow$  We should be able to break  $w = \mathbf{x} \mathbf{y} \mathbf{z}$  as follows:

- $\mathbf{x} = a_1 a_2 \dots a_i$ ;       $\mathbf{y} = a_{i+1} a_{i+2} \dots a_j$ ;       $\mathbf{z} = a_{j+1} a_{j+2} \dots a_m$
- $\mathbf{x}$ 's path will be  $p_0 \dots p_i$
- $\mathbf{y}$ 's path will be  $p_i p_{i+1} \dots p_j$  (but  $p_i = p_j$  implying a loop)
- $\mathbf{z}$ 's path will be  $p_j p_{j+1} \dots p_m$

- Now consider another string  $w_k = \mathbf{x} \mathbf{y}^k \mathbf{z}$ , where  $k \geq 0$

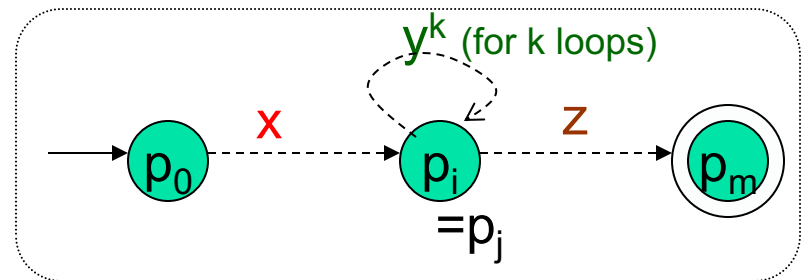
- Case  $k=0$

- DFA will reach the accept state  $p_m$

- Case  $k > 0$

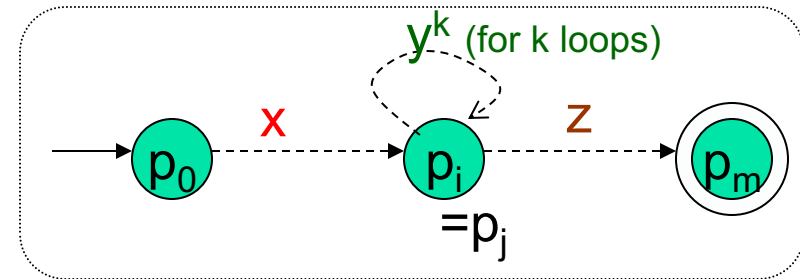
- DFA will loop for  $\mathbf{y}^k$ , and finally reach the accept state  $p_m$  for  $\mathbf{z}$

- In either case,  $w_k \in L$       This proves part (3) of the lemma



# Pumping Lemma: Proof...

- For part (1):
  - Since  $i < j$ ,  $y \neq \varepsilon$



- For part (2):
  - By PHP, the repetition of states has to occur within the first  $N$  symbols in  $w$
  - $\implies |xy| \leq N$

□



# Examples

---

Note: This N can be anything (need not necessarily be the #states in the DFA.)

# Example 1

**Claim  $L = \{w \mid w = 0^n 1^n, n \geq 1\}$  is not regular**

- Proof: By contradiction, assume L be regular.
  - Then, P/L constant should exist; let  $N =$  that P/L constant
  - Consider input  $w = 0^N 1^N$
  - By pumping lemma, we should be able to break  $w = xyz$ , such that:
    1.  $y \neq \varepsilon$
    2.  $|xy| \leq N$
    3. For all  $k \geq 0$ , the string  $xy^kz$  is also in L
  - $w$  can have one of the following shapes:

$$w = 0^N 1^N = 0 \dots 00 \dots 00 \dots 01 \dots 1 \implies \#0 > \#1$$

$\leftarrow x \rightarrow \leftarrow y \rightarrow \leftarrow x \rightarrow$

$$w = 0^N 1^N = 0 \dots 01 \dots 11 \dots 11 \dots 1 \implies \#0 < \#1$$

$\leftarrow x \rightarrow \leftarrow y \rightarrow \leftarrow x \rightarrow$

$$w = 0^N 1^N = 0 \dots 00 \dots 1 \dots 0 \dots 1 \dots 1 \dots 1 \implies w \text{ has not the required shape}$$

$\leftarrow x \rightarrow \leftarrow y \rightarrow \leftarrow x \rightarrow$



## Example 2

**Claim  $L_{eq} = \{w \mid w \text{ is a binary string with equal number of 1s and 0s}\}$  is not regular.**

Assume  $L_{eq}$  be regular. Then there exists  $N$  (P.L. constant) such that for every string  $w \in L$  s.t.  $|w| \geq N$ , there exists a way to break  $w$  into three parts,  $w=xyz$ , such that: (1)  $y \neq \varepsilon$ , (2)  $|xy| \leq N$  (3) For all  $k \geq 0$ , all strings of the form  $xy^kz \in L$ .

Take  $N=N^*$ , and  $w=0^{N^*}1^{N^*}$  ( $|w|=2N^* \geq N^*$ ),  $w \in L_{eq}$ .

Proof proceeds like in Example 1.



## Example 3

**Prove  $L = \{0^n 1 0^n \mid n \geq 1\}$  is not regular.**

Assume  $L_{eq}$  be regular. Then there exists  $N$  (P.L. constant) such that for every string  $w \in L$  s.t.  $|w| \geq N$ , there exists a way to break  $w$  into three parts,  $w = xyz$ , such that: (1)  $y \neq \varepsilon$ , (2)  $|xy| \leq N$  (3) For all  $k \geq 0$ , all strings of the form  $xy^kz \in L$ .

Take  $N = N^*$ , and  $w = 0^{N^*} 1 0^{N^*}$  ( $|w| = 2N^* + 1 \geq N^*$ ). Then  $w$  can be divided into 3 parts:  $x = 0 \dots 0$  (length  $N^* - 2$ ),  $y = 01$ , ( $|xy| = N^* - 2 + 2 \leq N^*$ ),  $z = 1 \dots 1$  (length  $N^*$ ). Then  $|xy| = N^* \leq N^*$ . For  $k = 0$  we have  $xz = 0 \dots 0$  (no 1). So not in  $L_{eq}$ .

We found a counterexample for which the PL does not hold. Hence  $L_{eq}$  is not regular.



## Example 4

---

**Prove  $L = \{1^n \mid n \text{ is prime}\}$  is not regular.**

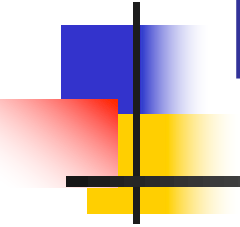
Assume  $L_{eq}$  be regular. Then there exists  $N$  (P.L. constant) such that for every string  $w \in L$  s.t.  $|w| \geq N$ , there exists a way to break  $w$  into three parts,  $w=xyz$ , such that: (1)  $y \neq \varepsilon$ , (2)  $|xy| \leq N$  (3) For all  $k \geq 0$ , all strings of the form  $xy^kz \in L$ .

Take  $N=p$ , and  $w=1^p$  ( $|w|=p \geq p$  and  $p$  - prime). Then  $w$  can be divided into 3 parts:  $|y|=l \geq 1$  (cond. (1) – is satisfied, and assume  $|xy| \leq p$  s.t. (2) is satisfied).

Trying to prove (3): Let  $k=p+1$ . We have  $|xy^{p+1}z| = |xyz| + |y^p| = p + p|y| = p(1+|y|)$  which is not always a prime number, e.g.  $p=3$ ,  $|y|=1$ ,  $|xy^{p+1}z| = 3(1+1) = 6$ .



# Equivalence & Minimization of DFAs





# Applications of interest

---

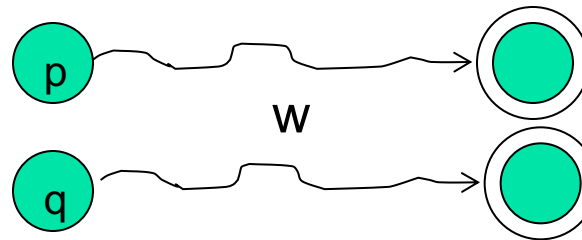
- Comparing two DFAs:
  - $L(\text{DFA}_1) == L(\text{DFA}_2)$ ?
  
- How to minimize a DFA?
  1. Remove unreachable states
  2. Identify & condense equivalent states into one

# When to call two states in a DFA “equivalent”?

Past doesn't matter - only future does!

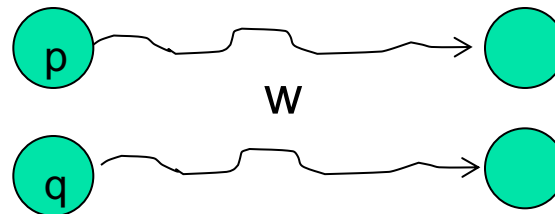
Two states  $p$  and  $q$  are said to be *equivalent* iff:

- i) Any string  $w$  accepted by starting at  $p$  is also accepted by starting at  $q$ ;



AND

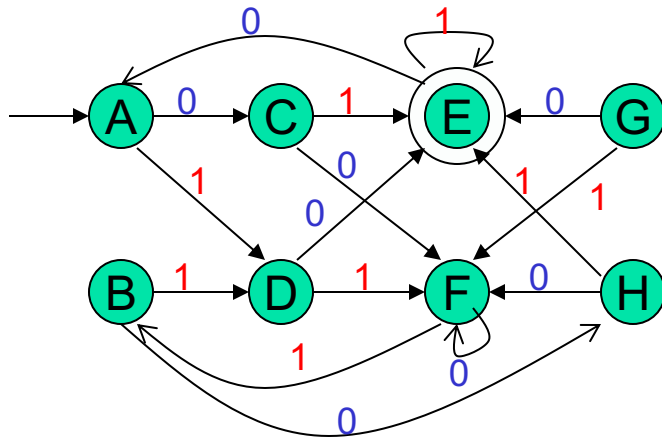
- i) Any string  $w$  rejected by starting at  $p$  is also rejected by starting at  $q$ .



→  $p \equiv q$

# Computing equivalent states in a DFA

## Table Filling Algorithm



Pass #0

1. Mark accepting states  $\neq$  non-accepting states

Pass #1

1. Compare every pair of states
2. Distinguish by one symbol transition
3. Mark = or  $\neq$  or blank (i.e. can not distinguish)

Pass #2

1. Compare every pair of states
2. Distinguish by up to two symbol transitions (until different or same or tbd)

....

(keep repeating until table complete) *How the table on the right was obtained?* **Table Filling Algorithm**

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

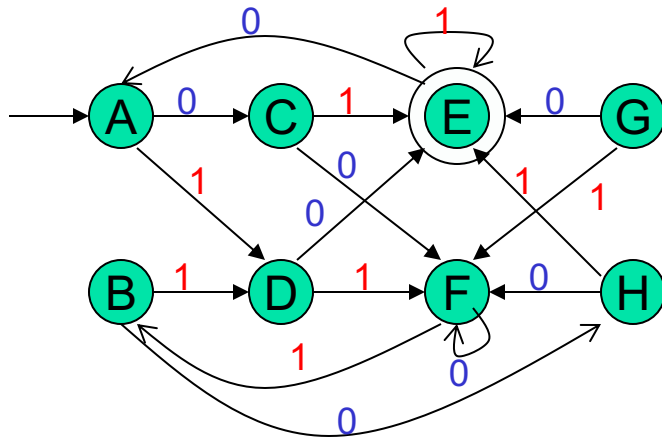


# Table Filling Algorithm (Myhill-Nerode Theorem)

---

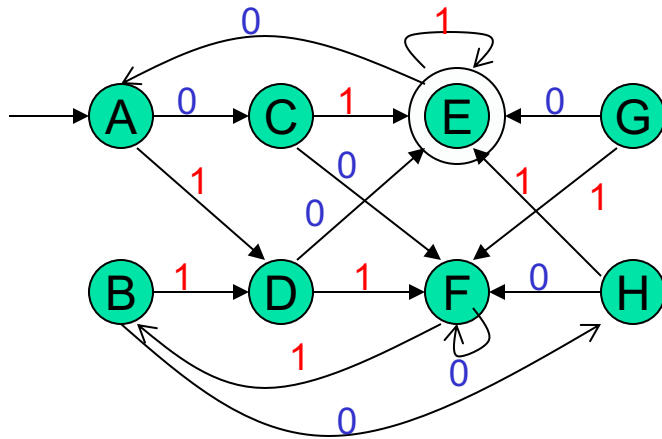
- Recursive discovery of distinguishable states in a DFA
  - **Base case:** If  $p$  is an accepting state and  $q$  is not accepting then the pair  $\{p, q\}$  is distinguishable.
  - **Induction:** Let  $p, q$  be states s.t. for some input symbol  $a$ ,  $r = \delta(p, a)$  and  $s = \delta(q, a)$  are known to be distinguishable. Then the pair  $\{p, q\}$  is distinguishable.

# Table Filling Algorithm - step by step



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

# Table Filling Algorithm - step by step

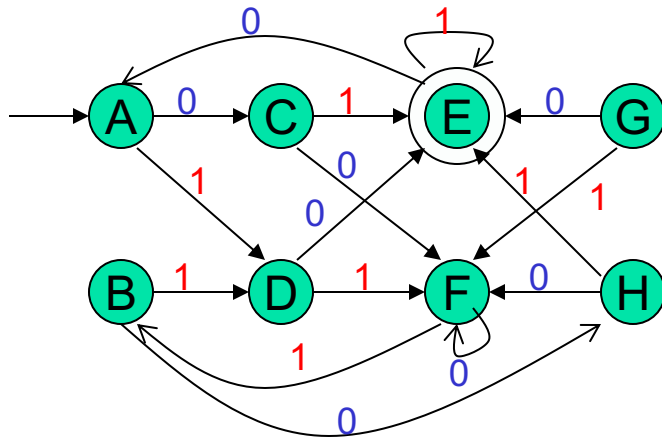


1. Mark X between accepting vs. non-accepting state



A	=							
B		=						
C			=					
D				=				
E	X	X	X	X	=			
F					X	=		
G					X		=	
H					X			=
	A	B	C	D	E	F	G	H

# Table Filling Algorithm - step by step



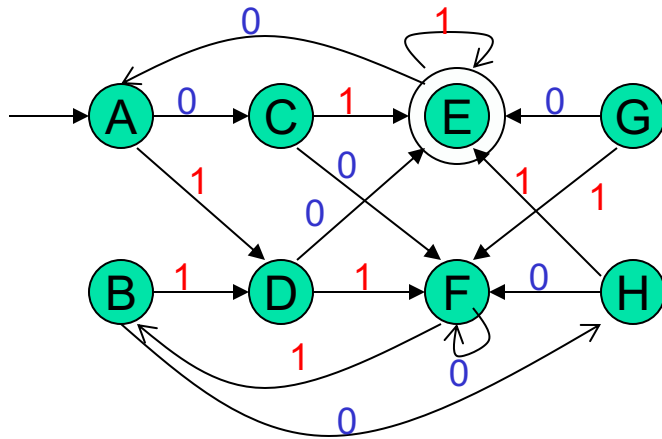
1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

A	=							
B		=						
C	X		=					
D	X			=				
E	X	X	X	X	=			
F					X	=		
G	X				X		=	
H	X				X			=
	A	B	C	D	E	F	G	H

↑



# Table Filling Algorithm - step by step

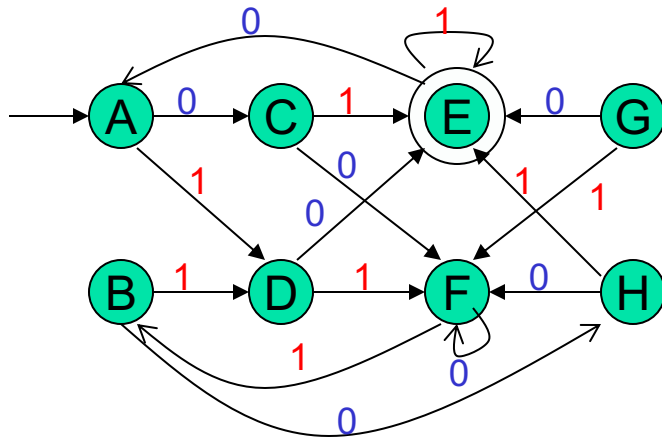


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

A	=							
B		=						
C	X	X	=					
D	X	X		=				
E	X	X	X	X	=			
F					X	=		
G	X	X			X		=	
H	X	X			X			=
	A	B	C	D	E	F	G	H



# Table Filling Algorithm - step by step

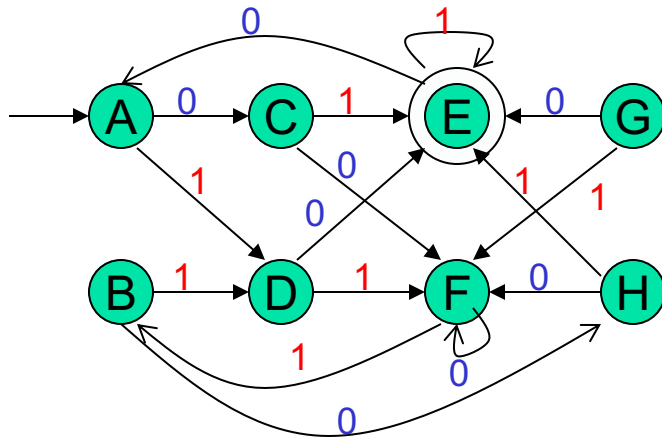


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

A	=							
B		=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F			X		X	=		
G	X	X	X		X		=	
H	X	X	=		X			=
	A	B	C	D	E	F	G	H

↑

# Table Filling Algorithm - step by step

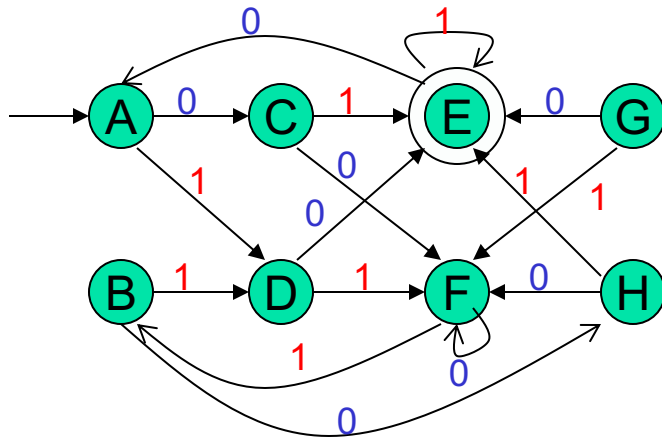


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

A	=							
B		=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F			X	X	X	=		
G	X	X	X	=	X		=	
H	X	X	=	X	X			=
	A	B	C	D	E	F	G	H

↑

# Table Filling Algorithm - step by step

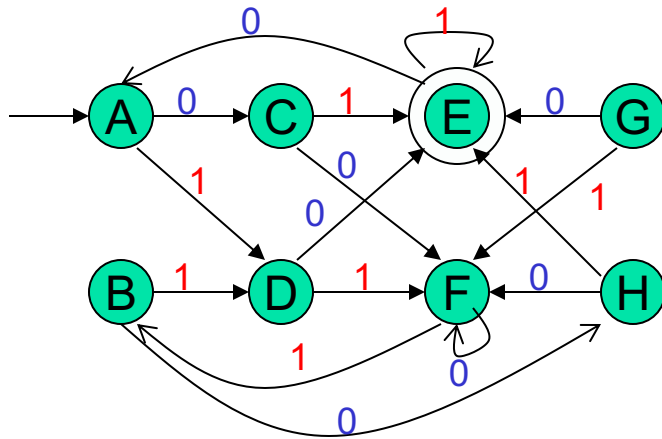


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

A	=							
B		=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F			X	X	X	=		
G	X	X	X	=	X	X	=	
H	X	X	=	X	X	X		=
	A	B	C	D	E	F	G	H



# Table Filling Algorithm - step by step

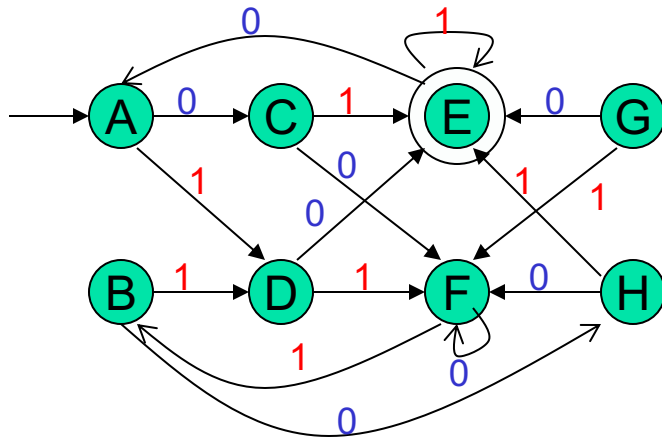


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

A	=							
B		=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F			X	X	X	=		
G	X	X	X	=	X	X	=	
H	X	X	=	X	X	X	X	=
	A	B	C	D	E	F	G	H



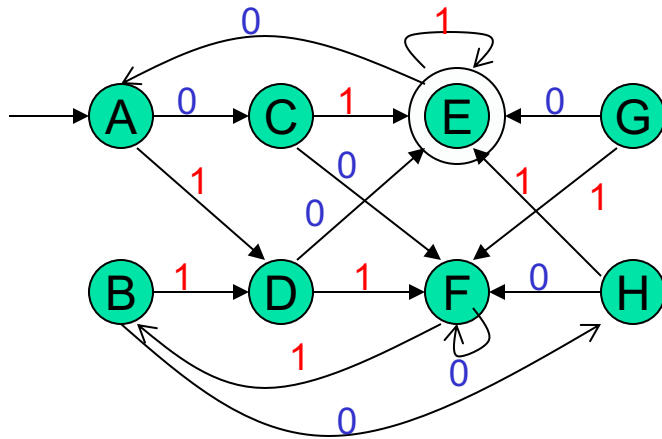
# Table Filling Algorithm - step by step



A	=							
B	=	=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F	X	X	X	X	X	=		
G	X	X	X	=	X	X	=	
H	X	X	=	X	X	X	X	=
	A	B	C	D	E	F	G	H

1. Mark **X** between accepting vs. non-accepting state
2. Pass 1:  
Look 1- hop away for distinguishing states or strings
3. Pass 2:  
Look 1-hop away again for distinguishing states or strings  
continue....

# Table Filling Algorithm - step by step



A	=							
B	=	=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F	X	X	X	X	X	=		
G	X	X	X	=	X	X	=	
H	X	X	=	X	X	X	X	=
	A	B	C	D	E	F	G	H

1. Mark **X** between accepting vs. non-accepting state

2. Pass 1:

Look 1- hop away for distinguishing states or strings

3. Pass 2:

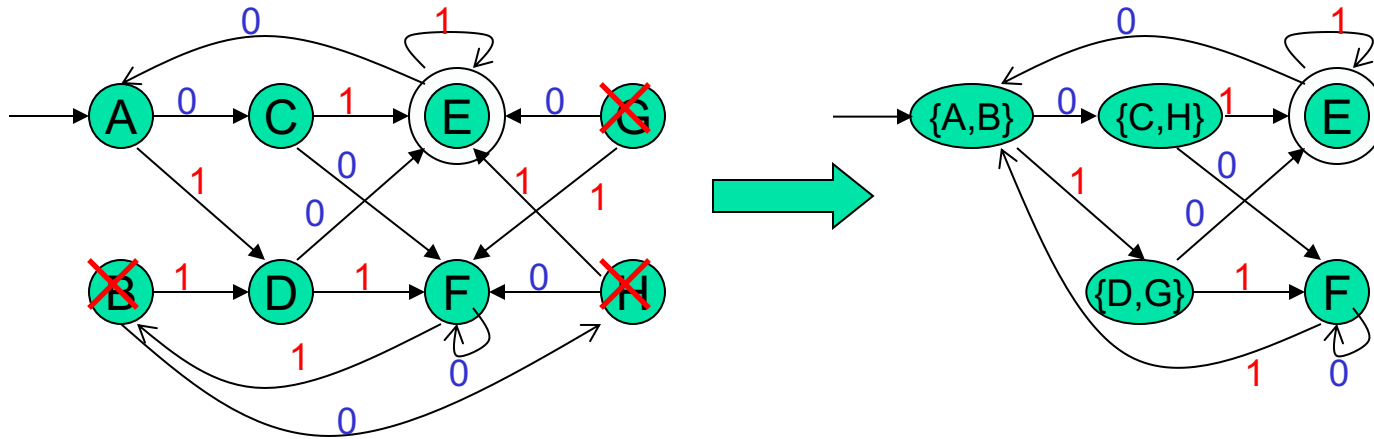
Look 1-hop away again for distinguishing states or strings

continue....

Equivalences:

- A=B
- C=H
- D=G

# Table Filling Algorithm - step by step



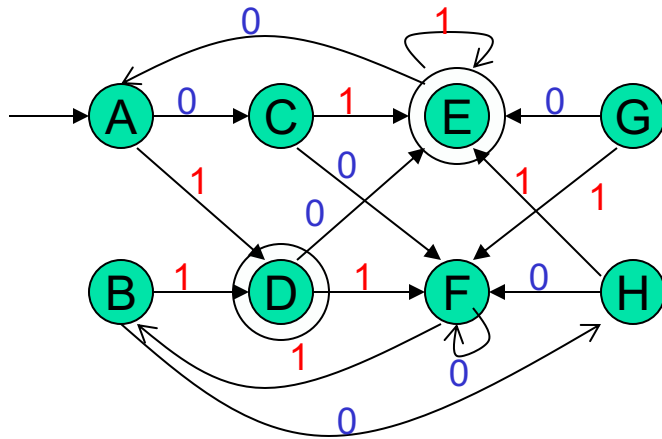
Retrain only one copy for each equivalence set of states

Equivalences:

- A=B
- C=H
- D=G



# Table Filling Algorithm – special case



A	=							
B		=						
C			=					
D				=				
E					?	=		
F							=	
G								=
H								=
	A	B	C	D	E	F	G	H

Q) What happens if the input DFA has more than one final state?  
 Can all final states initially be treated as equivalent to one another?

Putting it all together ...

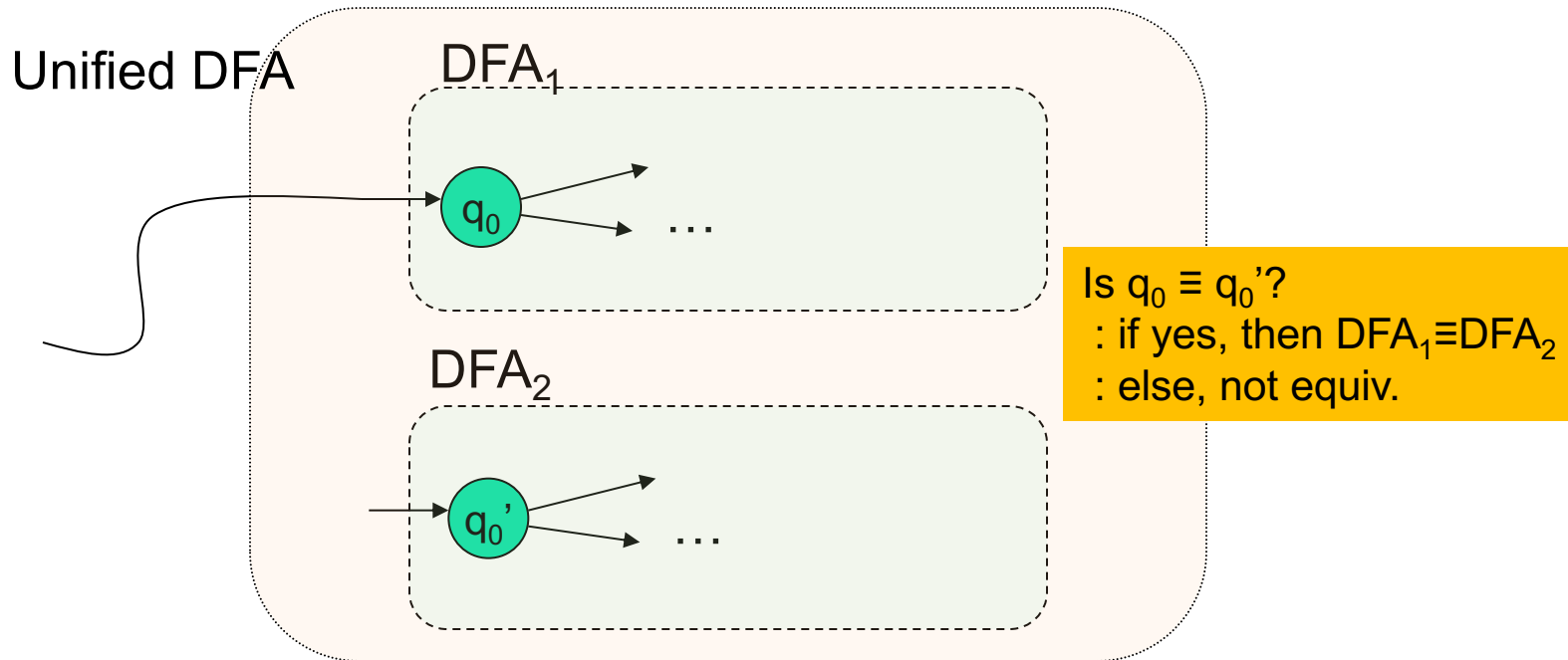
## How to minimize a DFA?

- Goal: Minimize the number of states in a DFA
- Algorithm:
  - 1. Eliminate states unreachable from the start state
  - 2. Identify and remove equivalent states
  - 3. Output the resultant DFA

Depth-first traversal from the start state

Table filling algorithm

# Are Two DFAs Equivalent?



1. Make a new dummy DFA by just putting together both DFAs
2. Run table-filling algorithm on the unified DFA
3. IF the start states of both DFAs are found to be equivalent,  
    *THEN:*  $DFA_1 \equiv DFA_2$   
    *ELSE:* different



# Summary

---

- How to prove languages are not regular?
  - Pumping lemma & its applications
- Simplification of DFAs
  - How to remove unreachable states?
  - How to identify and collapse equivalent states?
  - How to minimize a DFA?
  - How to tell whether two DFAs are equivalent?