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* ε -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?

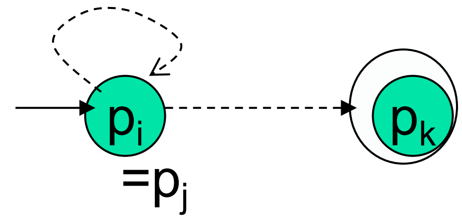
> *"The hardest thing of all is to find a black cat in a dark room, especially if there is no cat!"* *-Confucius*

Example of a non-regular language

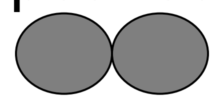
Let L = {w | w is of the form 0^n1^n , for all n≥0}

- *Hypothesis:* *L is not regular*
- Intuitive rationale: How do you keep track of a running count in an FA?
- A more formal rationale:
  - By contradition, if L is regular then there should exist a DFA for L.
  - Let k = number of states in that DFA.
  - Consider the special word w= 0^k1^k => w $\in$ L
  - DFA is in some state p_i, after consuming the first i symbols in w

Uses Pigeon Hole Principle



Rationale…

- Let $\{p_0, p_1, \ldots\ p_k\}$ 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!
- ==> at least one state should repeat somewhere along the path (by  +  Principle)
- ==> Let the repeating state be $p_i = p_j$ for $i < j$
- ==> We can fool the DFA by inputing $0^{(k-(j-i))}1^k$ and still get it to accept (note: k-(j-i) is at most k-1).
- ==> DFA accepts strings w/ 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*** x, y, z, $w=xyz$, 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.

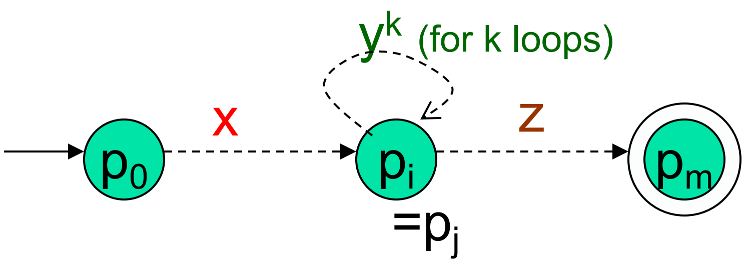
Definition: *N* is called the "Pumping Lemma Constant"

Pumping Lemma: Proof

- L is regular => 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: $w = a_1 a_2 \ldots a_m$, where $m \geq N$
- Let the states traversed after reading the first N symbols be: $\{p_0, p_1, \ldots p_N\}$
  - ==> There are N+1 p-states, while there are only N DFA states
  - ==> 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…

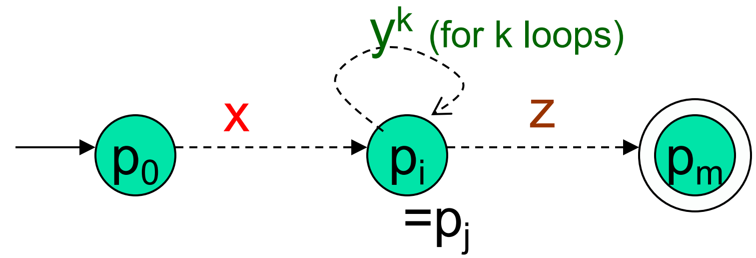
- => We should be able to break w=xyz as follows:
  - $x=a_1a_2..a_i$; $y=a_{i+1}a_{i+2}..a_J$; $z=a_{J+1}a_{J+2}..a_m$
  - x's path will be $p_0..p_i$
  - y's path will be $p_i\, p_{i+1}..p_j$ (but $p_i=p_j$ implying a loop)
  - z's path will be $p_jp_{J+1}..p_m$
- Now consider another string $w_k=xy^kz$, where $k\geq 0$



- Case k=0
  - DFA will reach the accept state p_m
- Case k>0
  - DFA will loop for y^k, and finally reach the accept state p_m for 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
  - ==> $|xy| \leq N$

$\square$

Using P.L. to prove non-regularity

1. L regular $\Rightarrow$ L satisfies P.L.
2. L non-regular $\Rightarrow$?
3. L non-regular $\Longleftarrow$ L does not satisfy P.L.: (1) & (2) & not (3) from P.L. formulation

Forall **N** exists $w \in L$ and $|w| \geq N$, forall x, y, z, $w = xyz$, such that:

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

Then L non-regular.

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

Example 1

Claim L = {w | $w = 0^n 1^n, n \geq 1$} is non-regular.

- Proof: Show that P.L. does not hold. (If P.L. holds this does not mean regularity)
- By contradiction, assume L is regular and derive a contradiction
  - Then, there exists N s.t. for all $w \in L$. Let $w = 0^N 1^N$
  - There exists x, y, z, 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 satisfies (1) and (2) above but:

$$w = 0^N 1^N = \underbrace{0\ldots0}_{x}\underbrace{0\ldots0}_{y}\underbrace{0\ldots01\ldots1}_{z} \implies \#0 > \#1$$

(3) does not hold

$$w = 0^N 1^N = \underbrace{0\ldots0}_{x}\underbrace{1\ldots1}_{y}\underbrace{1\ldots1\ldots1}_{z} \implies \#0 < \#1$$

$$w = 0^N 1^N = \underbrace{0\ldots0}_{x}\underbrace{0..1\ldots0\ldots1\ldots}_{y}\underbrace{1\ldots1}_{z}$$ ==> w has not the required shape

Example 2

Claim L_{eq} = {w | w is a binary string with equal number of 1s and 0s} is non-regular.

Assume L_{eq} be non-regular. Then there exists N such that for every string $w \in L$ s.t. $|w| \geq N$, there exists x, y, z with w=xyz, such that: (1) $y \neq \varepsilon$, (2) $|xy| \leq N$ (3) Exists $k \geq 0$ and $xy^kz \in L$.

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 $L = \{0^n10^n \mid n \geq 1\}$ is not regular.

Assume L_{eq} be regular and derive a contradiction. Then there exists N such that for every string $w \in L$ s.t. $|w| \geq N$, there exists x, y, z with 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 $w=0^{N^*}10^{N^*}$ ($|w|=2N^*+1 \geq N^*$). Then w can be divided into 3 parts: x=0…0 (length N*-2), y=01,($|xy|= N^*-2 +2 \leq N^*$), z=0…0 (length N*). Then $|xy|=N^* \leq N^*$. For k=0 we have xz=0…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 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 $|xy| <= p$ 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:
  - $L(DFA_1) == L(DFA_2)$?

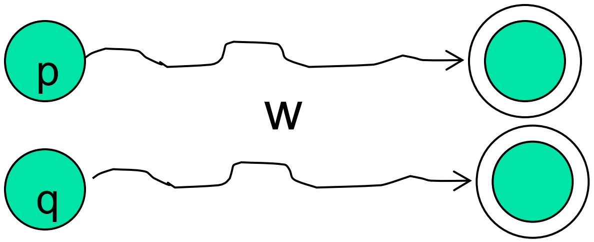
- How to minimize a DFA?
  1. Remove unreachable states
  2. Identify & condense equivalent states into one

When to call two states in a DFA "equivalent"?

Past doesn't matter - only future does!

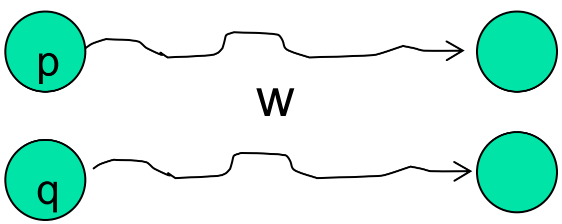
Two states p and q are said to be *equivalent* iff:

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



AND

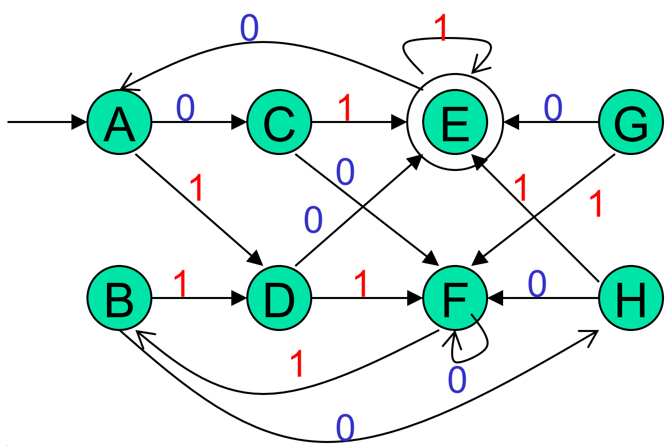
i) Any string w rejected by starting at p is also rejected by starting at q.



➔ p≡q

Computing equivalent states in a DFA

<u>Table Filling Algorithm</u>



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

<u>Pass #0</u>

1. Mark accepting states ≠ non-accepting states

<u>Pass #1</u>

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

<u>Pass #2</u>

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

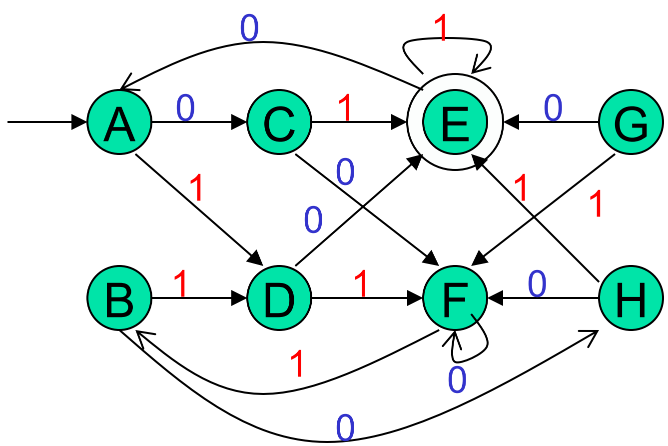
….

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

Table Filling Algorithm

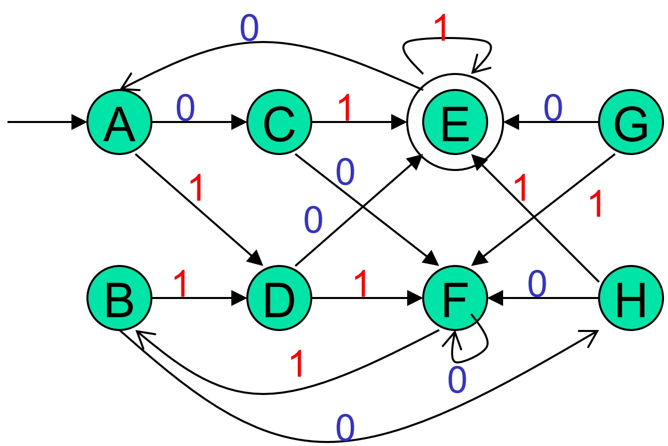
- Recursive discovery of distinguishable states in a DFA
  - **Base case**: If p is an accepting state and q is not accepting then the pair {p,q} is distinguishable.
  - **Induction**: Let p, q be states s.t. for some input symbol a, r = δ (p, a) and s = δ (q, a) are known to be distinguishable. Then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

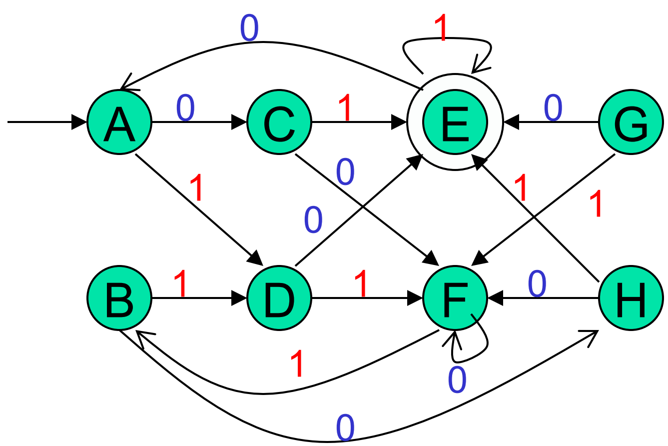
Table Filling Algorithm - step by step



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

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

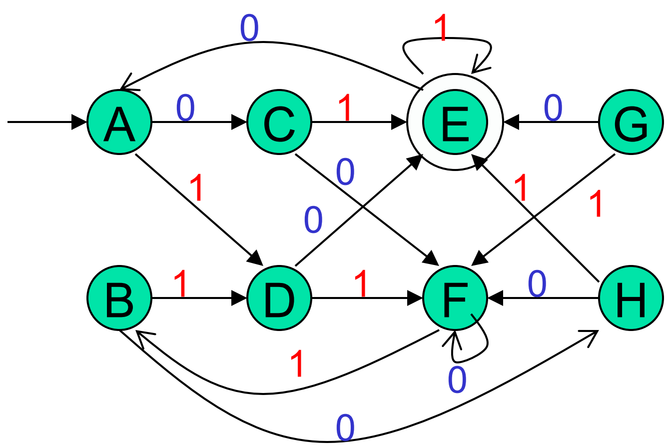
Table Filling Algorithm - step by step



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

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

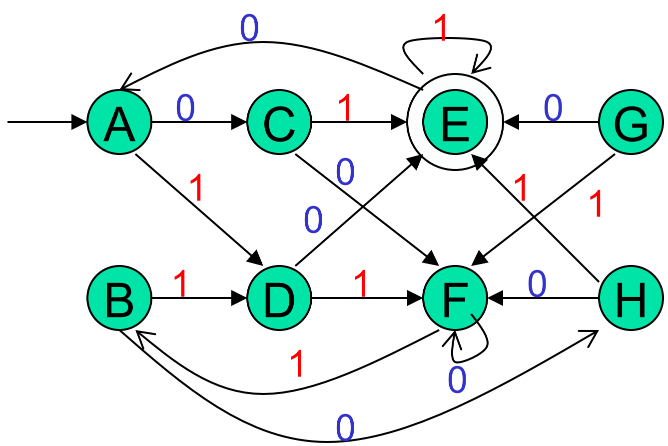
Table Filling Algorithm - step by step



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

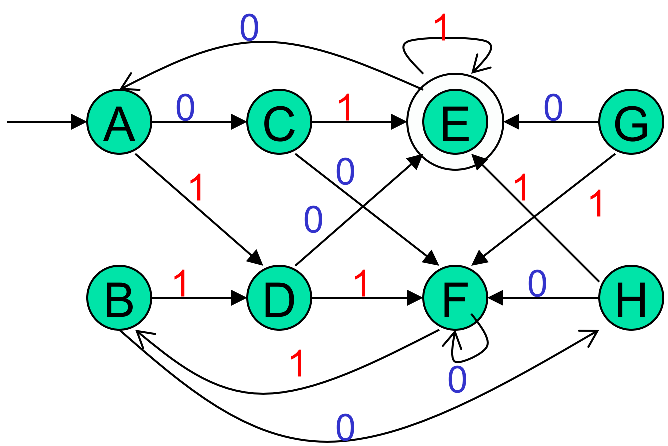
Table Filling Algorithm - step by step



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

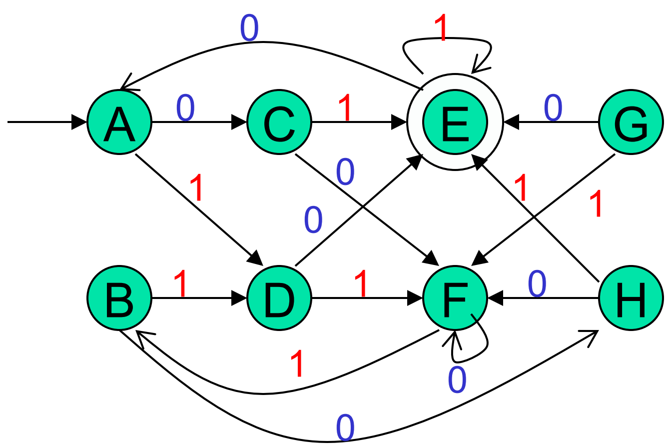
Table Filling Algorithm - step by step



	A	B	C	D	E	F	G	H
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			=

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

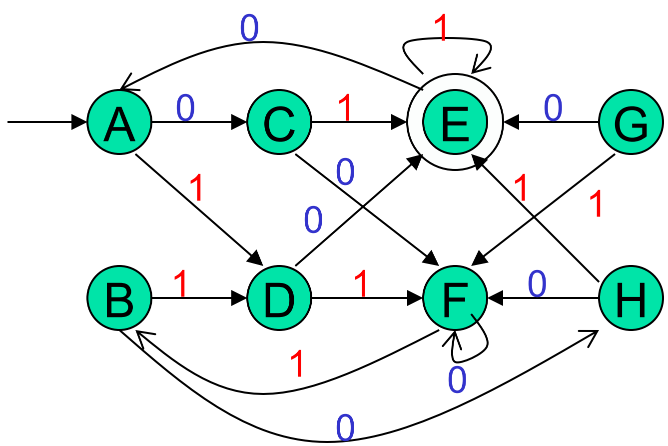
Table Filling Algorithm - step by step



	A	B	C	D	E	F	G	H
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		=

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

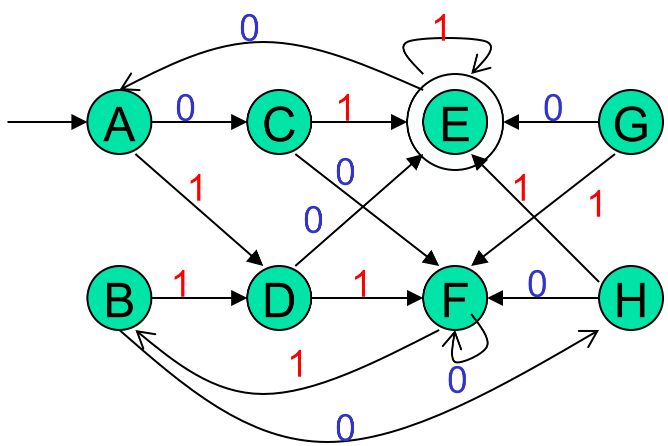
Table Filling Algorithm - step by step



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

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

Table Filling Algorithm - step by step

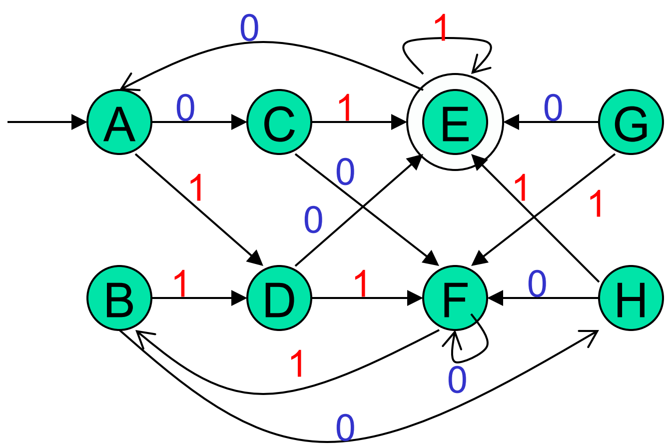


A	=							
B	**=**	=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F	**X**	**X**	X	X	X	=		
G	X	X	X	**=**	X	X	=	
H	X	X	**=**	X	X	X	X	=
	A	B	C	D	E	F	G	H

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

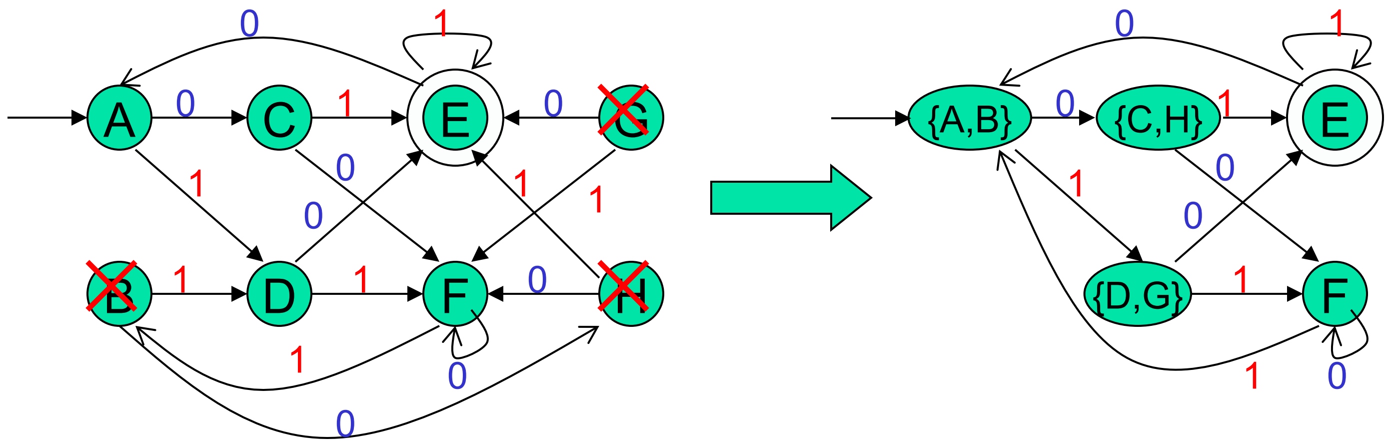


	A	B	C	D	E	F	G	H
A	=							
B	=	=						
C	X	X	=					
D	X	X	X	=				
E	X	X	X	X	=			
F	X	X	X	X	X	=		
G	X	X	X	=	X	X	=	
H	X	X	=	X	X	X	X	=

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

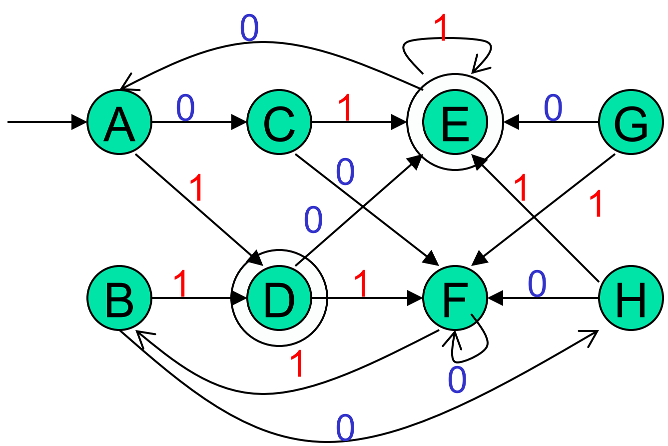
Table Filling Algorithm - step by step



Retrain only one copy for
each equivalence set of states

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

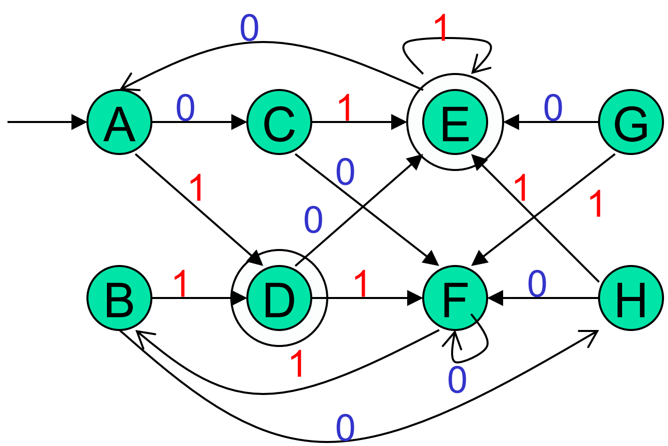
Table Filling Algorithm – special case



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?

DFA Minimization by state equivalence method



0 – Equivalence: $\{A, B, C, F, G, H\}$ $\{D, E\}$

1 – Equivalence $\{A, B, C, H\}$ $\{F\}$ $\{G\}$ $\{D\}$ $\{E\}$

$\delta(A,0) = C$
$\delta(B,0) = H$

$\delta(A,1) = D$
$\delta(B,1) = D$

$\delta(C,0) = F$ $\delta(F,0) = F$ $\delta(H,0) = F = \delta(C,0)$
$\delta(C,1) = E$ $\delta(F,1) = B$ $\delta(H,1) = E = \delta(C,1)$
$\delta(G,0) = E$ $\delta(G,1) = F$

$\delta(D,0) = E$
$\delta(E,0) = A$

2 – Equivalence

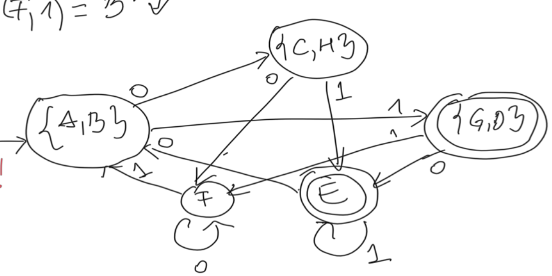
$\{A, B\}$ $\{C, H\}$ $\{F\}$ $\{G, D\}$ $\{E\}$

$\delta(A,0) = C$
$\delta(H,0) = F$
$\delta(C,0) = F$
$\delta(F,0) = F$
$\delta(G,0) = E$

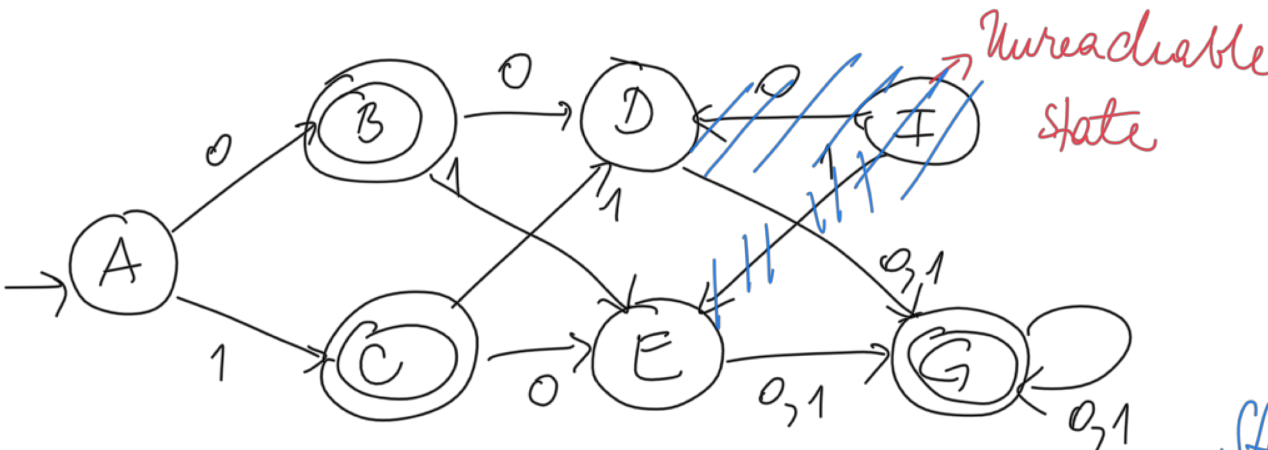
$\delta(H,1) = E$
$\delta(C,1) = E$
$\delta(F,1) = B$

$\delta(D,1) = F$
$\delta(G,1) = F$

3 – Equivalence
sets don't change!



DFA Minimization with unreacheable states



Step 1. Eliminate the unreach. State

Step 2. Proceed with State equivalence or table filling methods.

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

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?