



Properties of Regular Languages

Chapter-4

Dr. Naushin Nower




Topics

- 1) How to prove whether a given language is regular or not?
- 2) Closure properties of regular languages
- 3) Minimization of DFAs



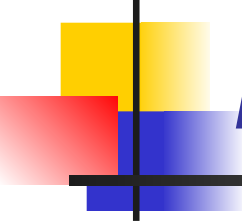
Some languages are *not* regular

When a language is regular?

if we are able to construct one of the following: DFA *or* NFA *or*  -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 = 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/ 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



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. $y \neq \epsilon$
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

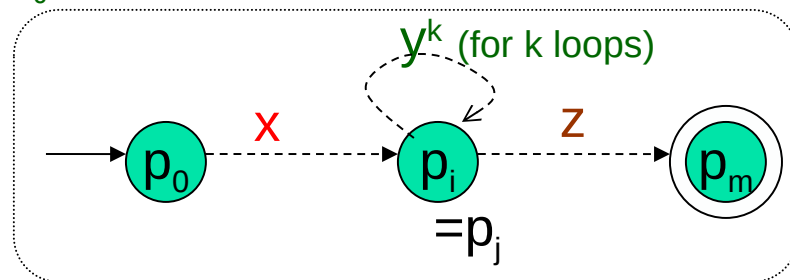
- L is regular \Rightarrow it should have a DFA.
 - Set $N :=$ number of states in the DFA
- Any string $w \in L$, s.t. $|w| \geq N$, should have the form: $w = a_1 a_2 \dots a_m$, where $m \geq N$
- Let the states traversed after reading the first N symbols be: $\{p_0, p_1, \dots, p_N\}$
 - \Rightarrow 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 $k \geq 0$

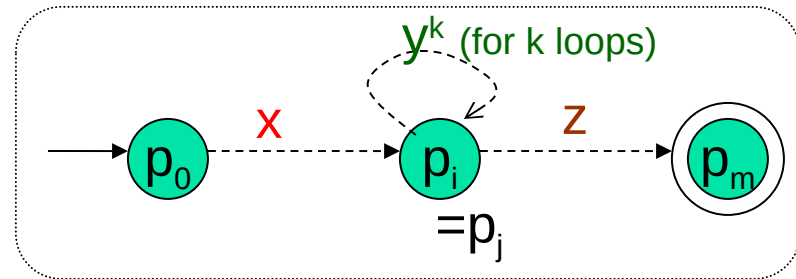


- Case $k=0$
 - DFA will reach the accept state p_m
- Case $k>0$
 - DFA will loop for \mathbf{y}^k , and finally reach the accept state p_m for \mathbf{z}

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

Pumping Lemma: Proof...

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



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





The Purpose of the Pumping Lemma for RL

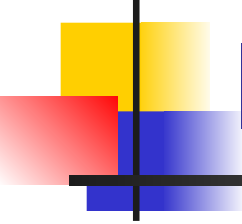
- To prove that some languages *cannot be* regular.



How to use the pumping lemma?

Think of playing a 2 person game

- Role 1: **We** claim that the language cannot be regular
- Role 2: An **adversary** who claims the language is regular
- We show that the adversary's statement will lead to a contradiction that implies pumping lemma *cannot* hold for the language.
- We win!!



How to use the pumping lemma? (The Steps)

1. (we) L is not regular.
 2. (adv.) Claims that L is regular and gives you a value for N as its P/L constant
 3. (we) Using N , choose a string $w \in L$ s.t.,
 1. $|w| \geq N$,
 2. Using w as the template, construct other words w_k of the form xy^kz and show that at least one such $w_k \notin L$
 \Rightarrow this implies we have successfully broken the pumping lemma for the language, and hence that the adversary is wrong.
- (Note: In this process, we may have to try many values of k , starting with $k=0$, and then 2, 3, .. so on, until $w_k \notin L$)

Note: We don't have any control over N , except that it is positive.

We also don't have any control over how to split $w=xyz$,
but xyz should respect the P/L conditions (1) and (2).



Using the Pumping Lemma

- What WE do?
 3. Using N , we construct our template string w
 4. Demonstrate to the adversary, either through pumping up or down on w , that some string $w_k \notin L$
(this should happen regardless of $w=xyz$)
- What the Adversary does?
 1. Claims L is regular
 2. Provides N

Note: This N can be anything (need not necessarily be the #states in the DFA.
It's the adversary's choice.)

Example of using the Pumping Lemma to prove that a language is not regular

Let $L_{eq} = \{w \mid w \text{ is a binary string with equal number of 1s and 0s}\}$

■ Your Claim: L_{eq} is not regular

■ Proof:

➤ By contradiction, let L_{eq} be regular

□ adv.

➤ P/L constant should exist

□ adv.

➤ Let N = that P/L constant

➤ Consider input $w = 0^N 1^N$

□ you

(your choice for the template string)

➤ By pumping lemma, we should be able to break $w = xyz$, such that:

□ you


1) $y \neq \epsilon$

2) $|xy| \leq N$

3) For all $k \geq 0$, the string xy^kz is also in L

Template string $w = 0^N 1^N = \underbrace{00}_{\leftarrow N} \cdot \underbrace{011}_{\rightarrow N} \rightarrow 1$

Proof...

- Because $|xy| \leq N$, xy should contain only 0s
 - (This and because $y \neq \epsilon$, implies $y = 0^+$)
- Therefore x can contain *at most* $N-1$ 0s
- Also, all the N 1s must be inside z
- By (3), any string of the form $xy^kz \in L_{eq}$ for all $k \geq 0$
- Case $k=0$: xz has at most $N-1$ 0s but has N 1s
- Therefore, $xy^0z \notin L_{eq}$
- This violates the P/L (a contradiction) 

Setting $k=0$ is referred to as "pumping down"

Setting $k>1$ is referred to as "pumping up"

Another way of proving this will be to show that if the #0s is arbitrarily pumped up (e.g., $k=2$), then the #0s will become exceed the #1s

□ you



Exercise 2

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

Note: This n is not to be confused with the pumping lemma constant N . That *can* be different.

In other words, the above question is same as proving:

- $L = \{0^m 1 0^m \mid m \geq 1\}$ is not regular



Example 3: Pumping Lemma


Claim: $L = \{ 0^i \mid i \text{ is a perfect square} \}$ is not regular

■ Proof:

- By contradiction, let L be regular.
- P/L should apply
- Let $N = P/L$ constant
- Choose $w = 0^{N^2}$
- By pumping lemma, $w = xyz$ satisfying all three rules
- By rules (1) & (2), y has between 1 and N 0s
- By rule (3), any string of the form xy^kz is also in L for all $k \geq 0$
- Case $k=0$:
 - $\#zeros(xy^0z) = \#zeros(xyz) - \#zeros(y)$
 - $N^2 - N \leq \#zeros(xy^0z) \leq N^2 - 1$
 - $(N-1)^2 < N^2 - N \leq \#zeros(xy^0z) \leq N^2 - 1 < N^2$
 - $xy^0z \notin L$
 - But the above will complete the proof ONLY IF $N > 1$.
 - ... (proof contd.. Next slide)



Example 3: Pumping Lemma

- (proof contd...)
 - If the adversary pick $N=1$, then $(N-1)^2 \leq N^2 - N$, and therefore the $\#zeros(xy^0z)$ could end up being a perfect square!
 - This means that pumping down (i.e., setting $k=0$) is not giving us the proof!
 - So lets try pumping up next...
- Case $k=2$:
 - $\#zeros(xy^2z) = \#zeros(xyz) + \#zeros(y)$
 - $N^2 + 1 \leq \#zeros(xy^2z) \leq N^2 + N$
 - $N^2 < N^2 + 1 \leq \#zeros(xy^2z) \leq N^2 + N < (N+1)^2$
 - $xy^2z \notin L$ 
- (Notice that the above should hold for all possible N values of $N>0$. Therefore, this completes the proof.)

Closure properties of Regular Languages



Closure properties for Regular Languages (RL)

This is different from Kleene closure

- Closure property:
 - If a set of regular languages are combined using an operator, then the resulting language is also regular
- Regular languages are closed under:
 - Union, intersection, complement, difference
 - Reversal
 - Kleene closure
 - Concatenation
 - Homomorphism
 - Inverse homomorphism

Now, lets prove all of this!



RLs are closed under union

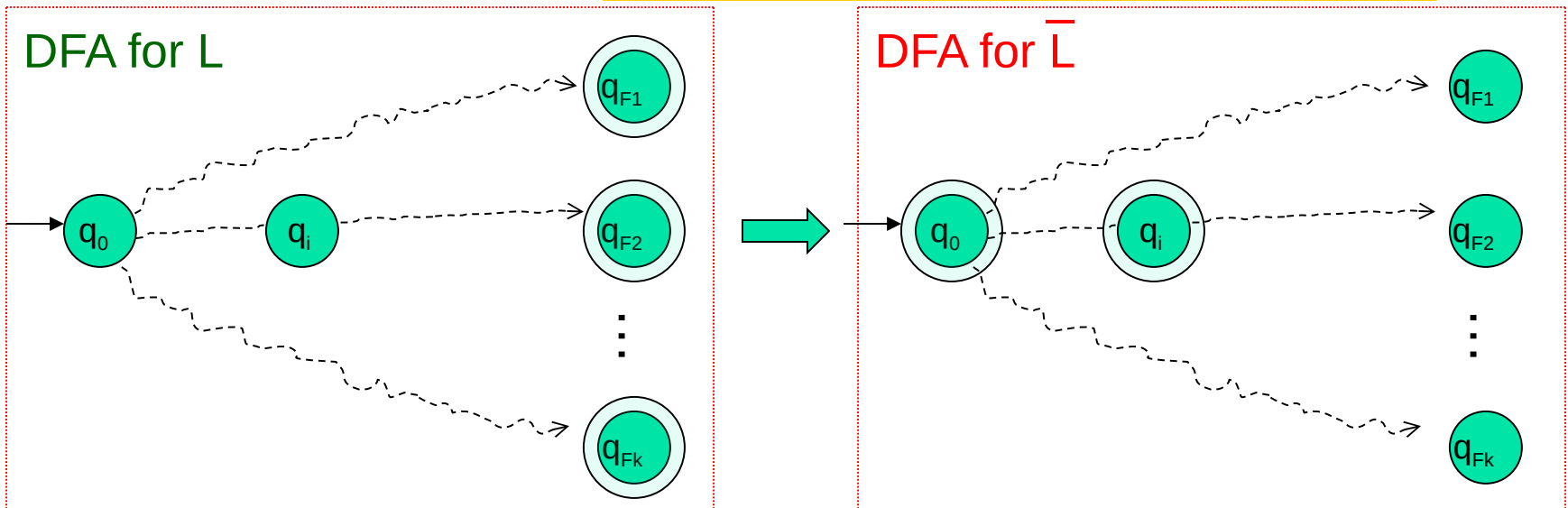
- IF L and M are two RLs THEN:
 - they both have two corresponding regular expressions, R and S respectively
 - $(L \cup M)$ can be represented using the regular expression $R+S$
 - Therefore, $(L \cup M)$ is also regular □

How can this be proved using FAs?

RLs are closed under complementation

- If L is an RL over Σ , then $\bar{L} = \Sigma^* - L$
- To show \bar{L} is also regular, make the following construction

Convert every final state into non-final, and every non-final state into a final state



Assumes q_0 is a non-final state. If not, do the opposite.



RLs are closed under intersection

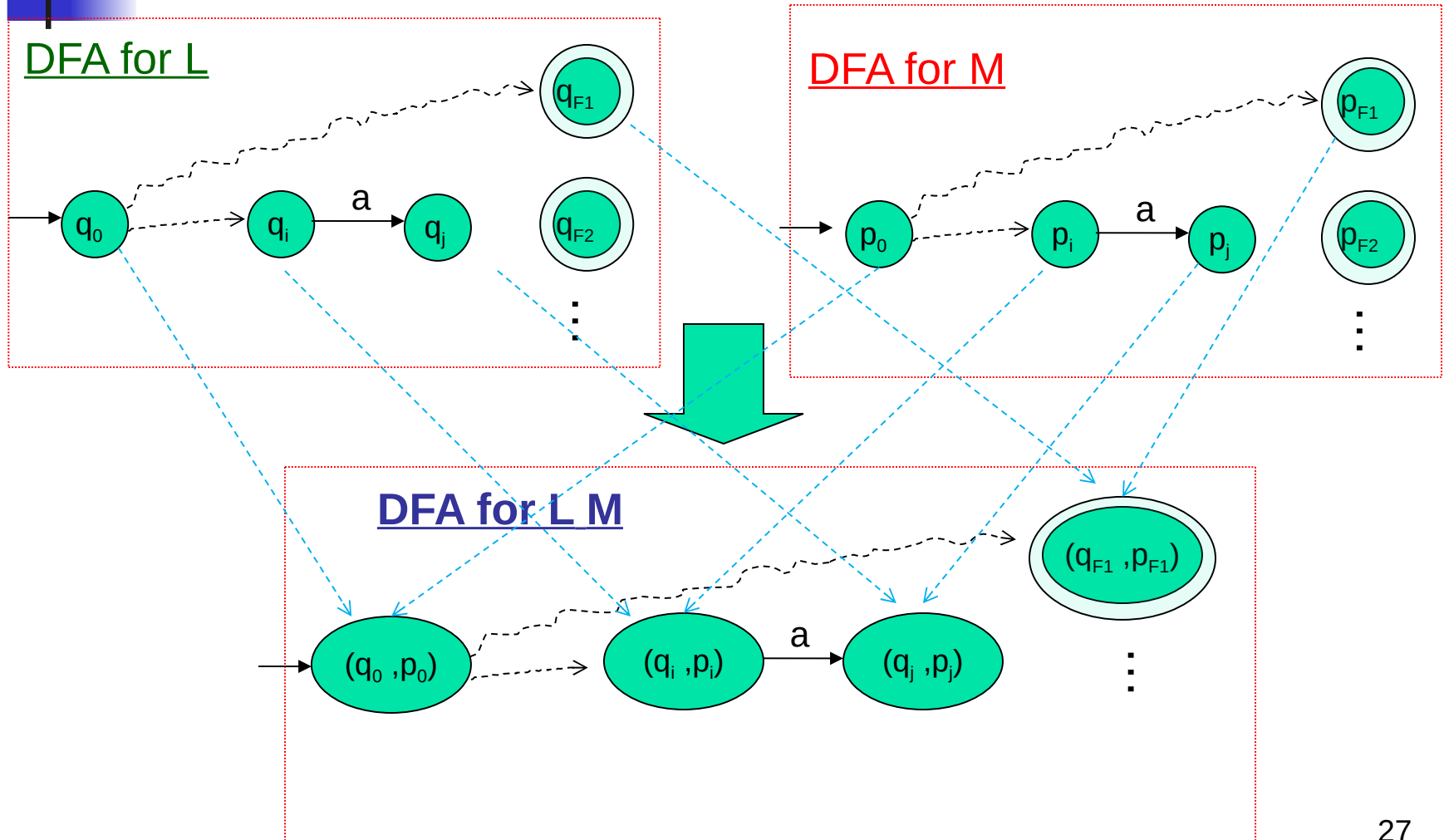
- A quick, indirect way to prove:
 - By DeMorgan's law:
 - $L \cap M = \overline{(\overline{L} \cup \overline{M})}$
 - Since we know RLs are closed under union and complementation, they are also closed under intersection
- A more direct way would be construct a finite automaton for $L \cap M$



DFA construction for $L \cap M$

- $A_L = \text{DFA for } L = \{Q_L, \Sigma, q_L, F_L, \delta_L\}$
- $A_M = \text{DFA for } M = \{Q_M, \Sigma, q_M, F_M, \delta_M\}$
- Build $A_{L \cap M} = \{Q_L \times Q_M, \Sigma, (q_L, q_M), F_L \times F_M, \delta\}$ such that:
 - $\delta((p, q), a) = (\delta_L(p, a), \delta_M(q, a))$, where p in Q_L , and q in Q_M
- This construction ensures that a string w will be accepted if and only if w reaches an accepting state in both input DFAs.

DFA construction for $L \cap M$



RLs are closed under set difference

- We observe:

- $L - M = L \cap \overline{M}$

Closed under intersection

Closed under
complementation

- Therefore, $L - M$ is also regular



RLs are closed under reversal

Reversal of a string w is denoted by w^R

- E.g., $w=00111$, $w^R=11100$

Reversal of a language:

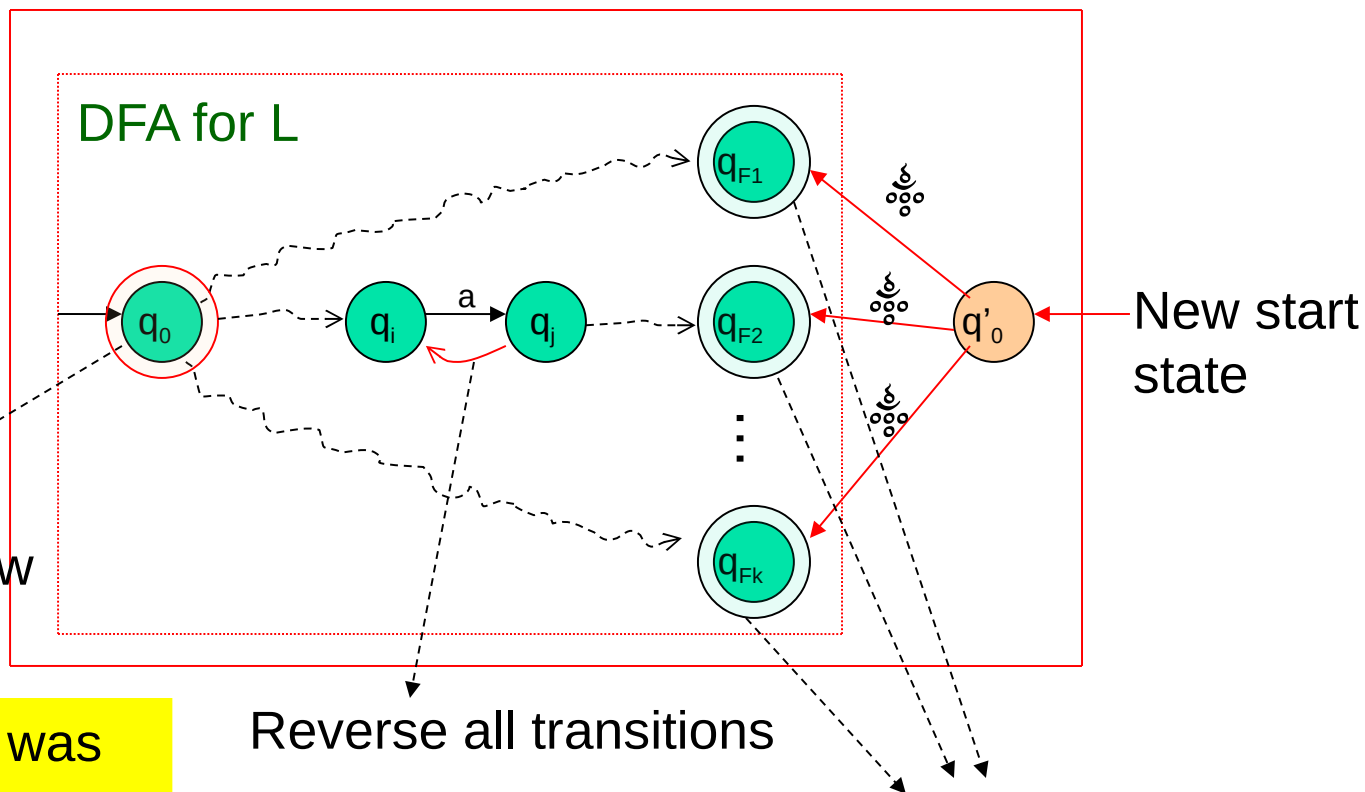
- L^R = The language generated by reversing all strings in L

Theorem: If L is regular then L^R is also regular



-NFA Construction for L^R

New -NFA for L^R



Make the old start state as the only new final state

What to do if q_0 was one of the final states in the input DFA?



If L is regular, L^R is regular (proof using regular expressions)

- Let E be a regular expression for L
- Given E , how to build E^R ?
- Basis: If $E = \epsilon$, \emptyset , or a , then $E^R = E$
- Induction: Every part of E (refer to the part as “ F ”) can be in only *one* of the three following forms:
 1. $F = F_1 + F_2$
 - $F^R = F_1^R + F_2^R$
 2. $F = F_1 F_2$
 - $F^R = F_2^R F_1^R$
 3. $F = (F_1)^*$
 - $(F^R)^* = (F_1^R)^*$

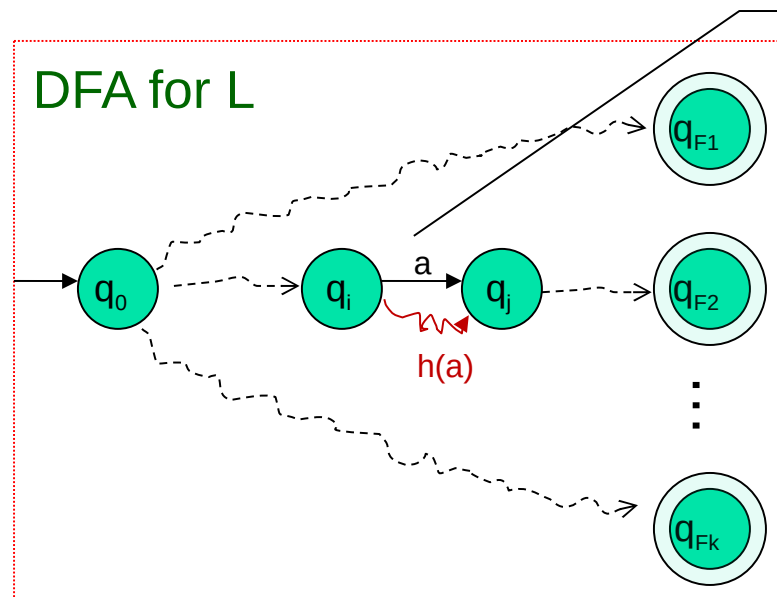


Homomorphisms

- Substitute each symbol in Σ (main alphabet) by a corresponding string in T (another alphabet)
 - $h: \Sigma \rightarrow T^*$
- Example:
 - Let $\Sigma = \{0, 1\}$ and $T = \{a, b\}$
 - Let a homomorphic function h on Σ be:
 - $h(0) = ab, h(1) = \textcircled{ab}$
 - If $w = 10110$, then $h(w) = \textcircled{ab} ab \textcircled{ab} \textcircled{ab} ab = abab$
- In general,
 - $h(w) = h(a_1) h(a_2) \dots h(a_n)$

Given a DFA for L , how to convert it into an FA for $h(L)$?

FA Construction for $h(L)$



Replace every edge “ a ” by a path labeled $h(a)$ in the new DFA

- Build a new FA that simulates $h(a)$ for every symbol a transition in the above DFA
- The resulting FA may or may not be a DFA, but will be a FA for $h(L)$

Given a DFA for M , how to convert it into an FA for $h^{-1}(M)$?

The set of strings in Σ^* whose homomorphic translation results in the strings of M

Inverse homomorphism

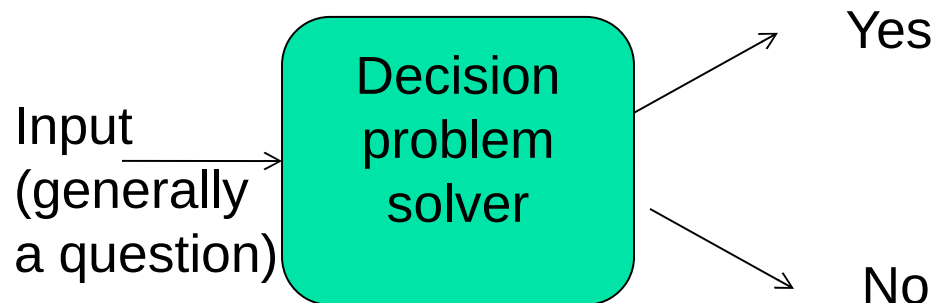
- Let $h: \Sigma \rightarrow T^*$
- Let M be a language over alphabet T
- $h^{-1}(M) = \{w \mid w \in \Sigma^* \text{ s.t.}, h(w) \in M\}$

Claim: If M is regular, then so is $h^{-1}(M)$

- Proof:
 - Let A be a DFA for M
 - Construct another DFA A' which encodes $h^{-1}(M)$
 - A' is an exact replica of A , except that its transition functions are s.t. for any input symbol a in Σ , A' will simulate $h(a)$ in A .
 - $\delta(p, a) \hat{=} \delta(p, h(a))$

Decision properties of regular languages

Any “decision problem” looks like this:





Membership question

- Decision Problem: Given L , is w in L ?
- Possible answers: Yes or No
- Approach:
 1. Build a DFA for L
 2. Input w to the DFA
 3. If the DFA ends in an accepting state, then yes; otherwise no.



Emptiness test

- Decision Problem: Is $L = \emptyset$?
- Approach:
On a DFA for L:
 1. From the start state, run a *reachability* test, which returns:
 1. success: if there is at least one final state that is reachable from the start state
 2. failure: otherwise
 2. $L = \emptyset$ if and only if the reachability test fails

How to implement the reachability test?



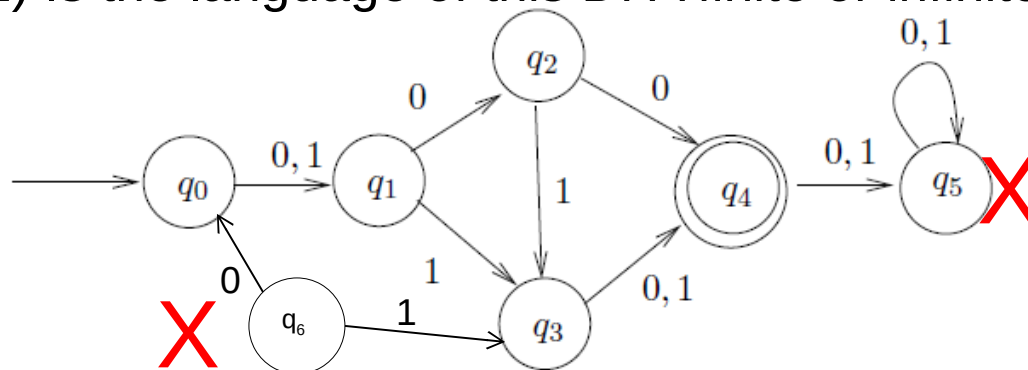
Finiteness

- Decision Problem: Is L finite or infinite?
- Approach:
 - On a DFA for L:
 1. Remove all states unreachable from the start state
 2. Remove all states that cannot lead to any accepting state.
 3. After removal, check for cycles in the resulting FA
 4. L is finite if there are no cycles; otherwise it is infinite
- Another approach
 - Build a regular expression and look for Kleene closure

How to implement steps 2 and 3?

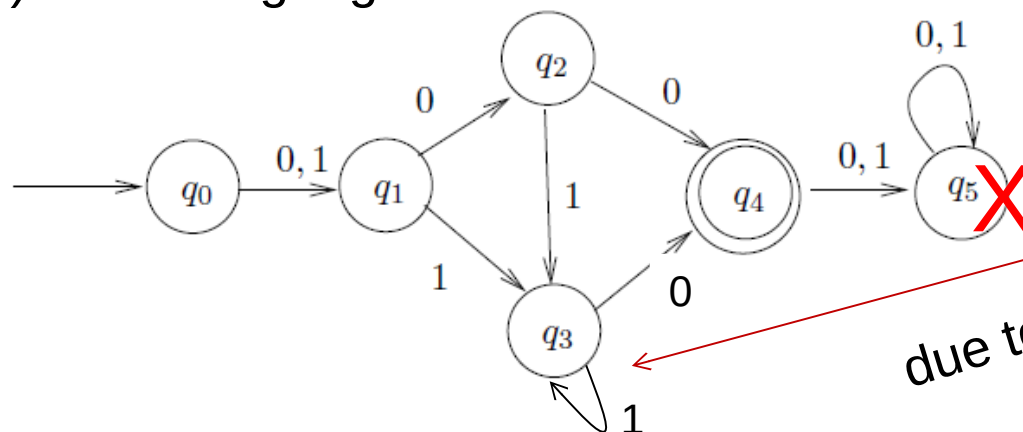
Finiteness test - examples

Ex 1) Is the language of this DFA finite or infinite?



FINITE

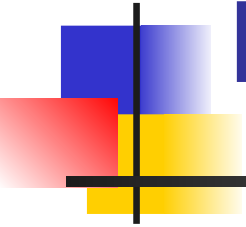
Ex 2) Is the language of this DFA finite or infinite?



INFINITE

due to this

Equivalence & Minimization of DFAs





Applications of interest

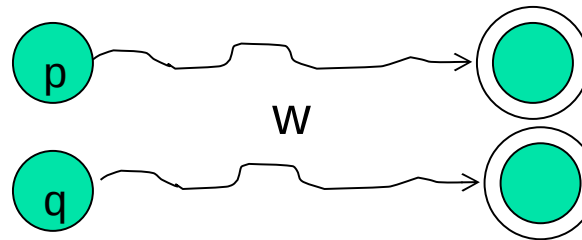
- Comparing two DFAs:
 - $L(\text{DFA}_1) == L(\text{DFA}_2)$?

- 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”?

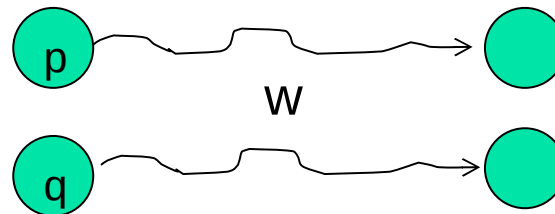
Two states p and q are said to be *equivalent* iff:

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



AND

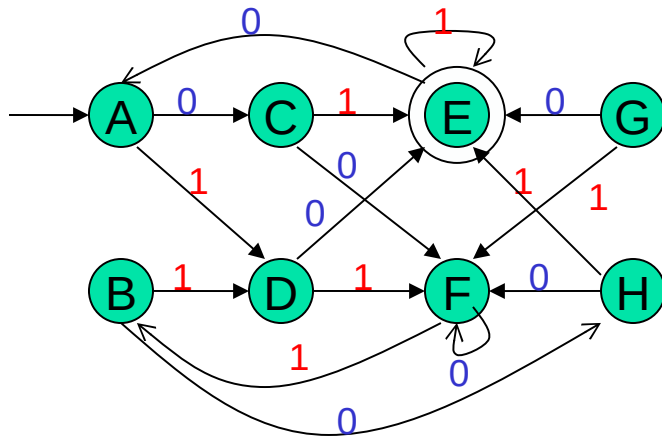
- ii) Any string w rejected by starting at p is also rejected by starting at q .



$\square p \equiv q$

Computing equivalent states in a DFA

Table Filling Algorithm



Pass #0

1. Mark accepting states \neq non-accepting states

Pass #1

2. Compare every pair of states
3. Distinguish by one symbol transition
4. Mark = or \neq or blank(tbd)

Pass #2

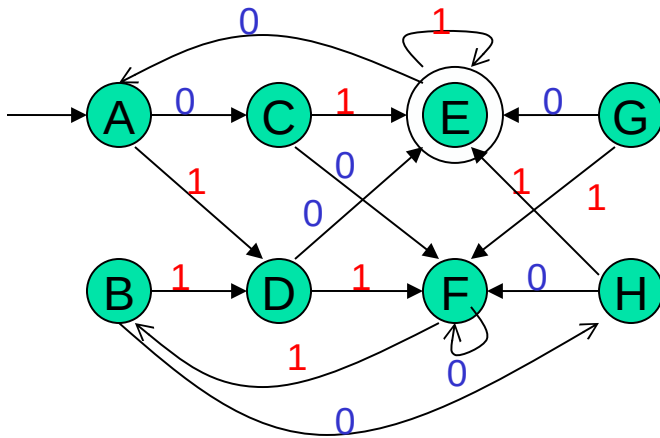
5. Compare every pair of states
6. Distinguish by up to two symbol transitions (until different or same or tbd)

....

(keep repeating until table complete)

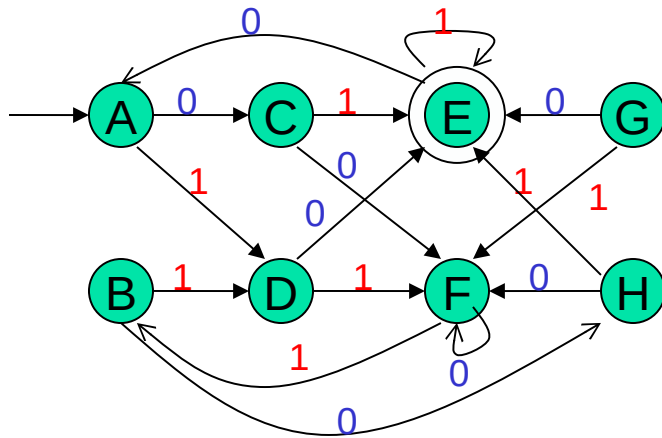
A	=							
B	=	=						
C	x	x	=					
D	x	x	x	=				
E	x	x	x	x	=			
F	x	x	x	x	x	=		
G	x	x	x	=	x	x	=	
H	x	x	=	x	x	x	x	=
	A	B	C	D	E	F	G	H

Table Filling Algorithm - step by step



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

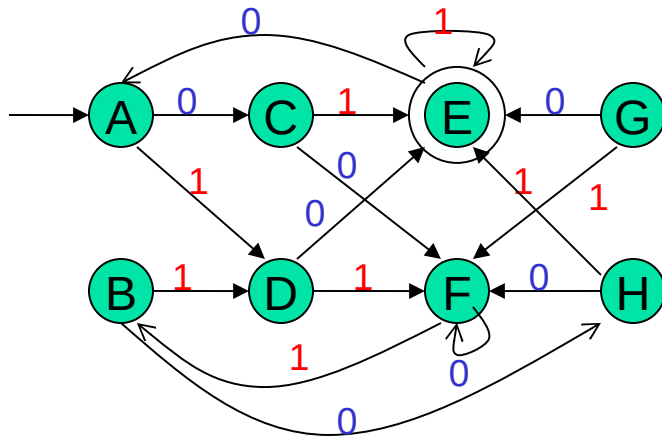
Table Filling Algorithm - step by step



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

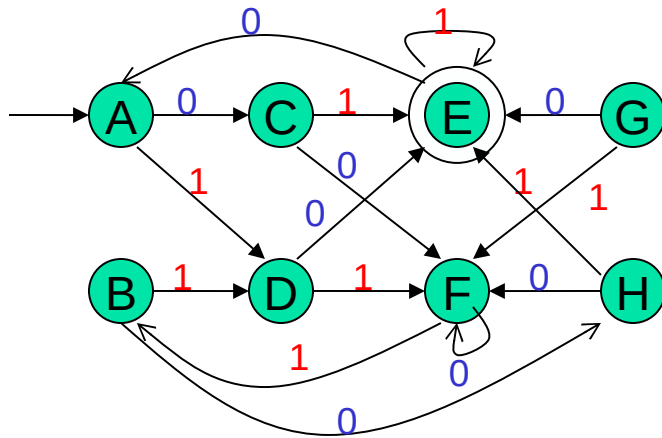


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

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

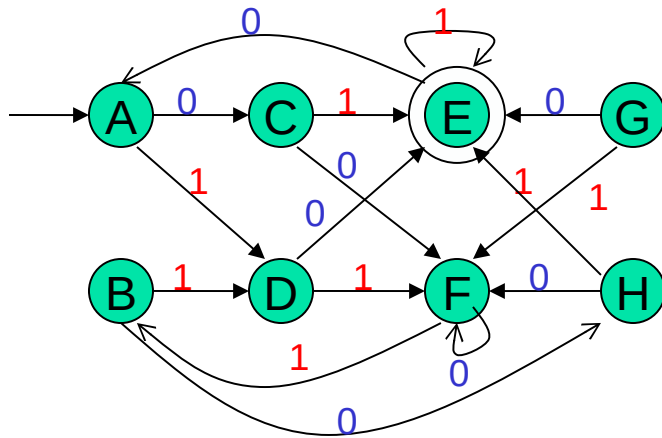


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

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

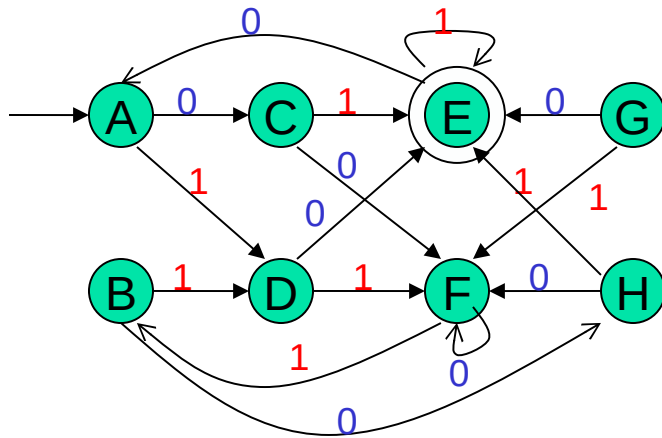


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

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

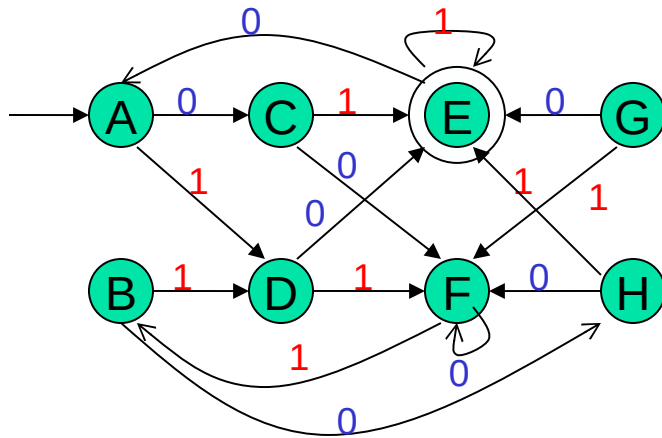


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

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

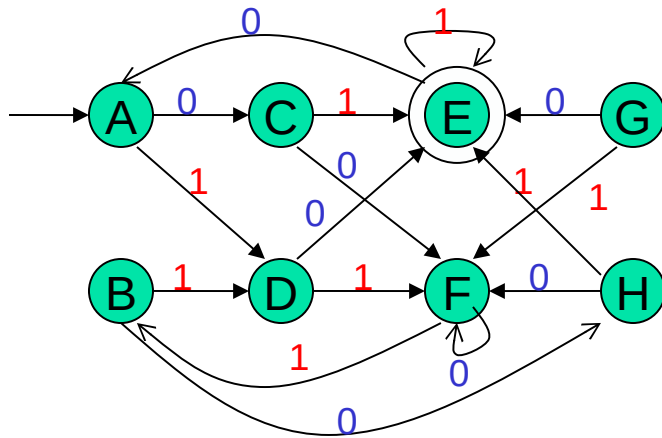


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

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

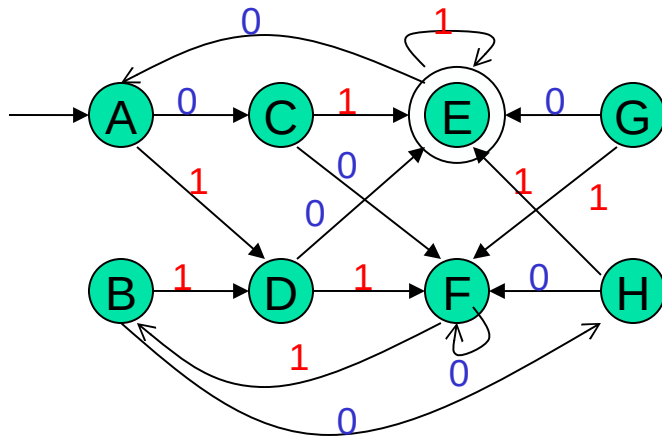


1. Mark X between accepting vs. non-accepting state
2. Look 1- hop away for distinguishing states or strings

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



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

Pass 1:

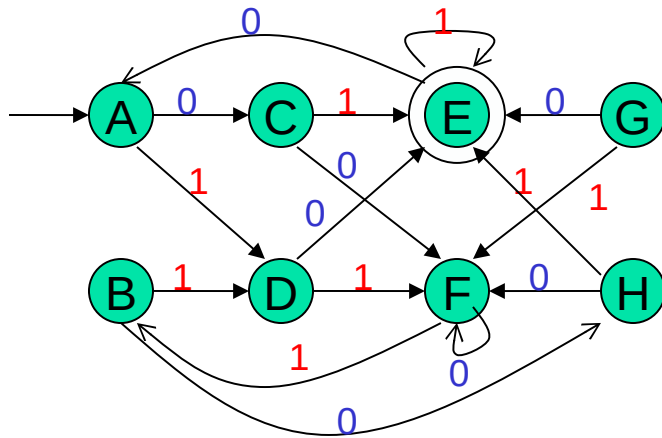
Look 1- hop away for distinguishing states or strings

Pass 2:

Look 1-hop away again for distinguishing states or strings
continue....

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 - step by step



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

Pass 1:

Look 1- hop away for distinguishing states or strings

Pass 2:

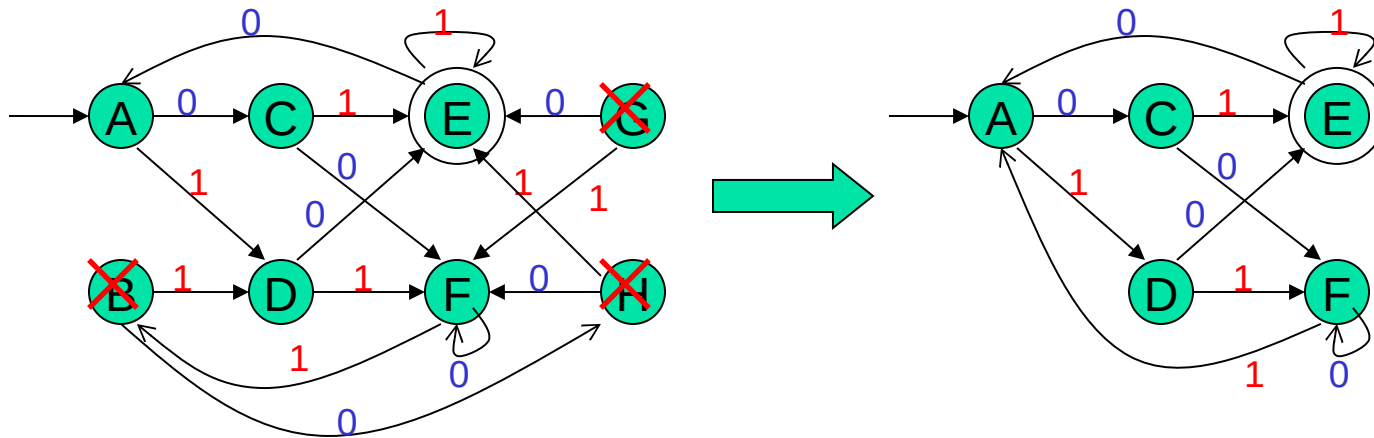
Look 1-hop away again for distinguishing states or strings
continue....

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

Equivalences:

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

Table Filling Algorithm - step by step

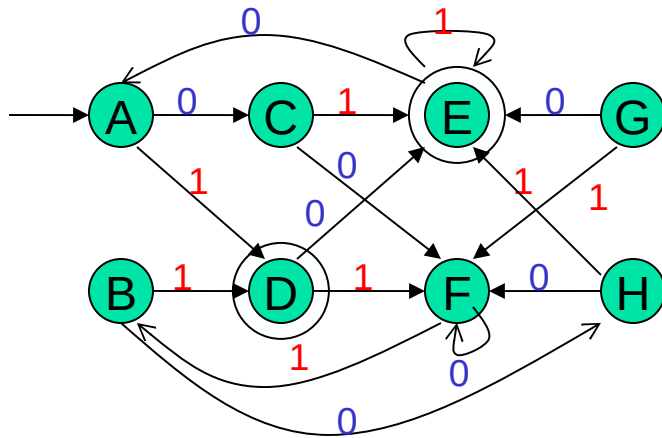


Retrain only one copy for each equivalence set of states

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

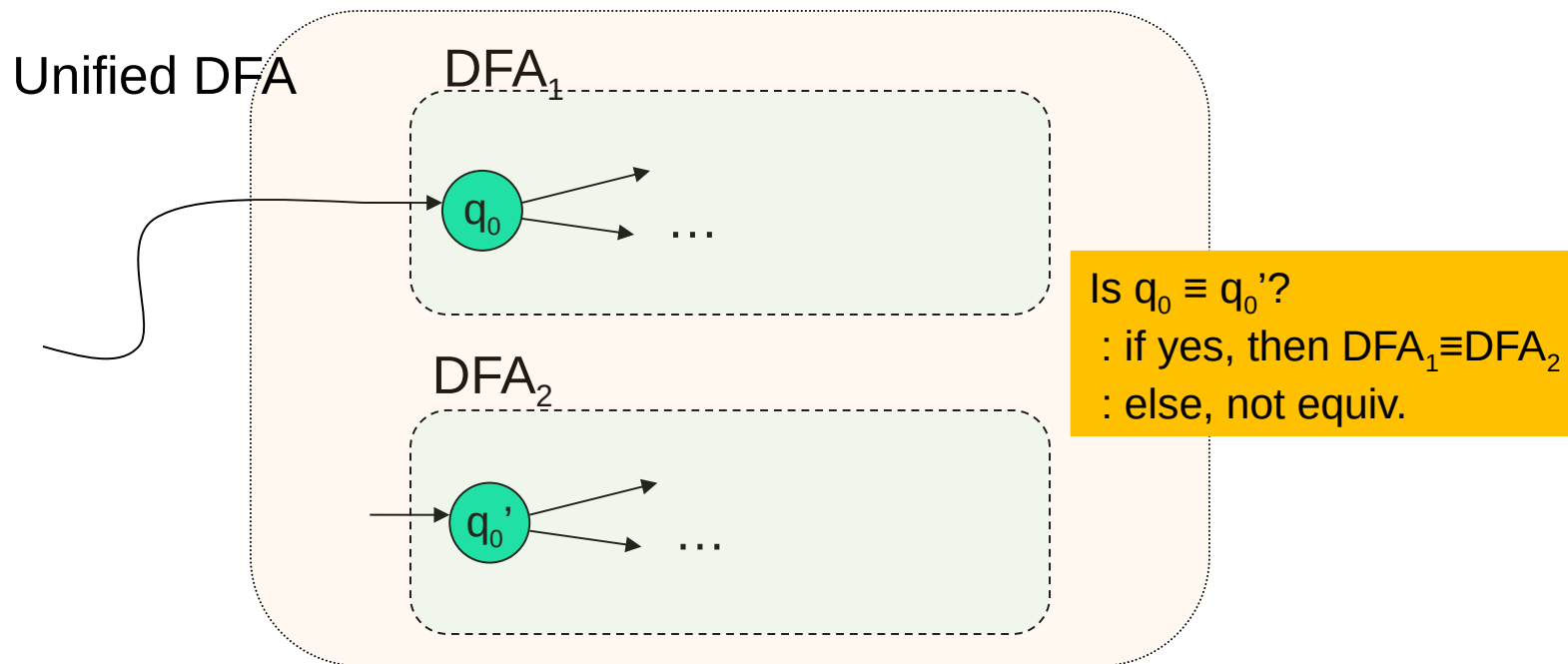
Depth-first traversal from the start state



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
- Closure properties of regular languages
- 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?