

COM3064

Automata Theory

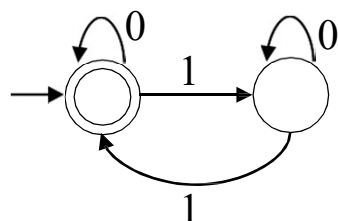
Week 5: Properties of Regular Languages

Lecturer: Dr. Sevgi YİĞİT SERT
Spring 2023

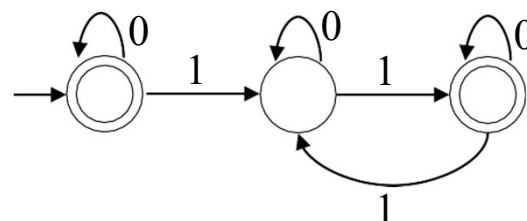
Resources: Introduction to The Theory of Computation, M. Sipser,
Introduction to Automata Theory, Languages, and Computation, J.E. Hopcroft, R. Motwani, and J.D. Ullman
BBM401 Automata Theory and Formal Languages, İlyas Çiçekli

Minimization and Equivalence of Automata

- Every DFA defines a regular language
- In general, there can be many DFAs for a given regular language.
- These DFAs accept the same regular language.
 - Language: The set of strings of 0's and 1's containing even number of 1's



A minimal DFA



- In practice, we are interested in the **DFA with the minimal number of states**.
 - Use less memory
 - Use less hardware (flip-flops)
- We can find a **minimal DFA** for any given DFA and their languages are equal.

Equivalent States

- Let $N = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and $\{p, q\} \subseteq Q$, we say that p and q are **equivalent (indistinguishable) states** if:

$$\text{for all } w \in \Sigma^* \quad | \quad \delta^*(p, w) \in F \text{ iff } \delta^*(q, w) \in F$$

- For all input strings w , $\delta^*(p, w)$ is an accepting state if and only if $\delta^*(q, w)$ is an accepting state.
- This means that *for all* $w \in \Sigma^*$
 - $\delta^*(p, w) \in F$ iff $\delta^*(q, w) \in F$ and
 - $\delta^*(p, w) \notin F$ iff $\delta^*(q, w) \notin F$
- Two **equivalent states** *behave same for all possible strings*.
- Hence, a state p is **distinguishable** from state q if there is at least one string w such that either $\delta^*(p, w) \in F$ or $\delta^*(q, w) \in F$ and the other is **NOT**.
 - There exists a string w such that

$$(\delta^*(p, w) \in F \text{ and } \delta^*(q, w) \notin F) \text{ or } (\delta^*(p, w) \notin F \text{ and } \delta^*(q, w) \in F)$$

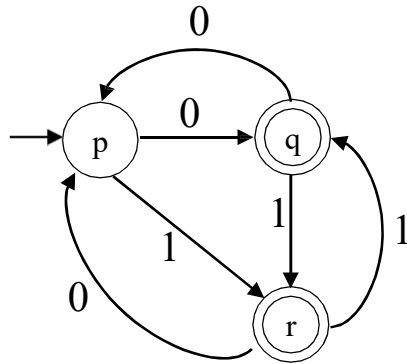
Equivalent States

- Equivalent (**indistinguishable**) states behave the same for all possible strings.
 - So, we do not need all of states from a set of indistinguishable states.
 - We can eliminate all of them by keeping only one of them to represent that set of equivalent states.
- Indistinguishability is an **equivalence relation**:
 - **Reflexive**: Each state is indistinguishable from itself
 - **Symmetric**: If p is indistinguishable from q, then q is indistinguishable from p
 - **Transitive**: If p is indistinguishable from q, and q is indistinguishable from r, then p is indistinguishable from r.

Finding Distinguishable States – Table Filling Algorithm

- We can compute equivalent states with a **table filling algorithm**.
- All other pairs of states are equivalent, and can be merged appropriately.
- **Step 1:**
 - Consider all pairs of states (p, q)
 - if $p \in F$ and $q \notin F$ or $p \notin F$ and $q \in F$, **mark (p, q) as distinguishable**
- **Step 2:** Repeat the following until no previously unmarked pairs are marked:
 - $\forall p, q \in Q$ and $\forall a \in \Sigma$, find $\delta(p, a) = r$ and $\delta(q, a) = s$,
 - **if (r, s) is marked as distinguishable then mark (p, q) as distinguishable.**
- We use table filling algorithm to minimize a DFA by merging all equivalent states.
- We replace a state p with its ***equivalence class*** found by the table filling algorithm.

Table Filling Algorithm - Example



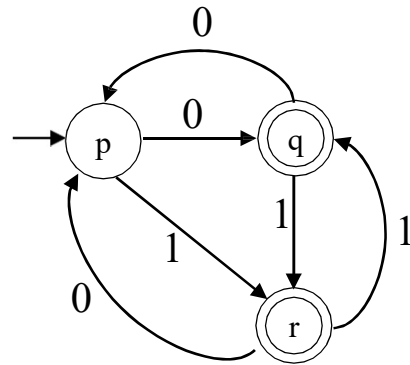
q	x	
r	x	
	p	q

- p is distinguishable from q and r , **mark them**

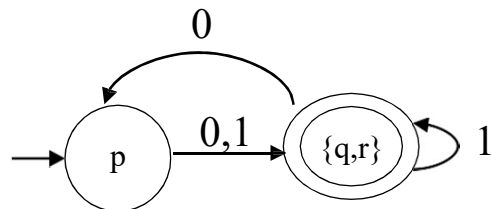
q	x	
r	x	
	p	q

- Both q and r go to p with 0, so no string beginning with 0 will distinguish them
- Starting in either q and r , an input of 1 takes us to either, so they are indistinguishable.
- Equivalence relation partitions (equivalence classes): $\{\{p\}, \{q, r\}\}$

Table Filling Algorithm - Example

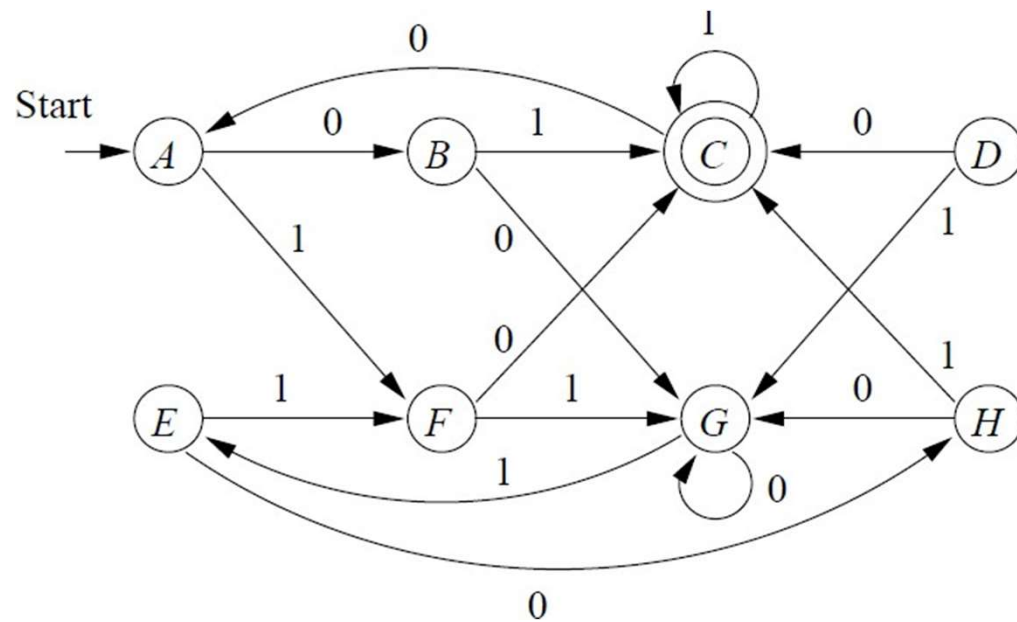


- Equivalence relation partitions (equivalence classes): $\{\{p\}, \{q, r\}\}$
- q and r are **equivalent** (indistinguishable).

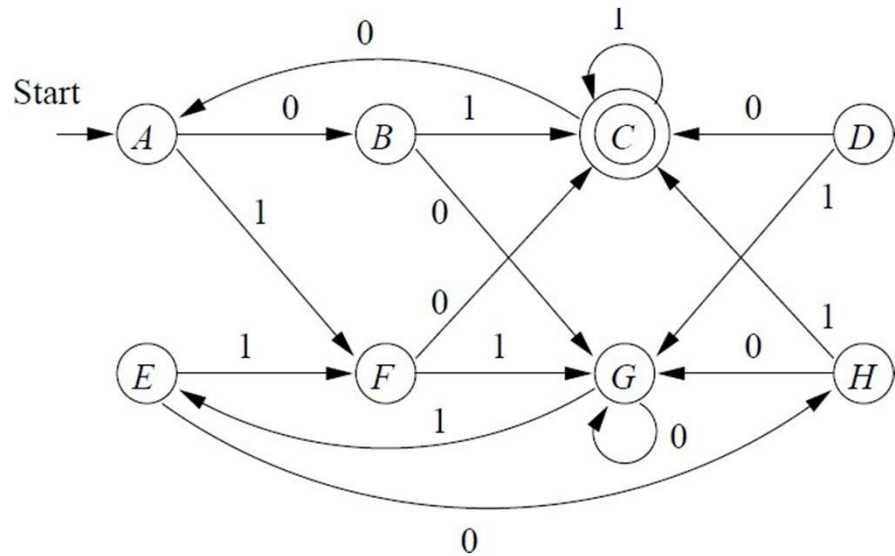


DFA with minimal states

Minimization of DFA: Table Filling Algorithm - Example 2



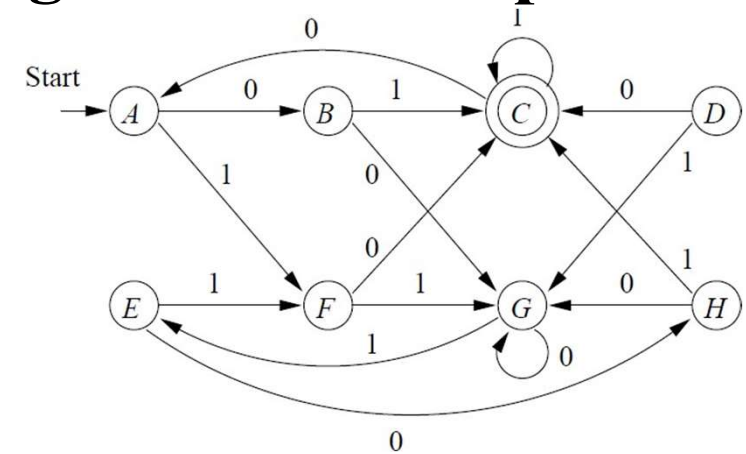
Minimization of DFA: Table Filling Algorithm - Example 2



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

- **PASS 0:** Distinguish accepting states from non-accepting states
- C is only accepting state, it is distinguishable from all other non-accepting states.

Minimization of DFA: Table Filling Algorithm - Example 2



PASS 1: Consider column A

$A \not\equiv B$ since $\delta(A,1) = F$, $\delta(B,1) = C$ and $F \not\equiv C$

$A \not\equiv D$ since $\delta(A,0) = B$, $\delta(D,0) = C$ and $B \not\equiv C$

$A \stackrel{?}{\equiv} E$ since

- $\delta(A,0) = B$, $\delta(E,0) = H$ and $B \stackrel{?}{\equiv} H$
- $\delta(A,1) = F$, $\delta(E,1) = F$ and $F \equiv F$

$A \not\equiv F$ since $\delta(A,0) = B$, $\delta(F,0) = C$ and $B \not\equiv C$

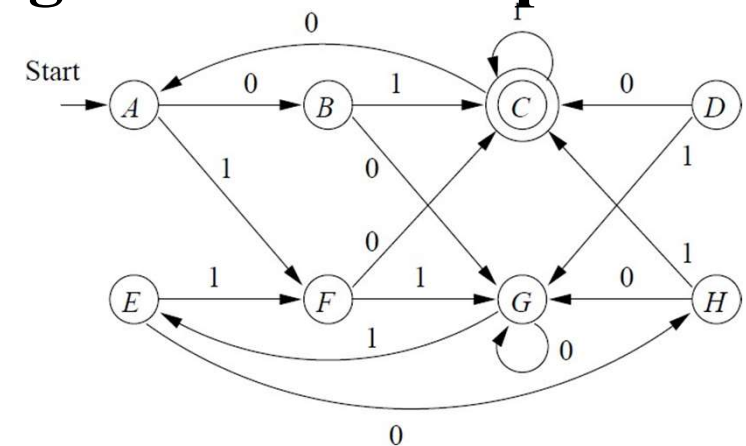
$A \stackrel{?}{\equiv} G$ since

- $\delta(A,0) = B$, $\delta(G,0) = G$ and $B \stackrel{?}{\equiv} G$
- $\delta(A,1) = F$, $\delta(G,1) = E$ and $F \equiv E$

$A \not\equiv H$ since $\delta(A,1) = F$, $\delta(H,1) = C$ and $F \not\equiv C$

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

Minimization of DFA: Table Filling Algorithm - Example 2



PASS 1: Consider column B

$B \not\equiv D$ since $\delta(B,1) = C$, $\delta(D,1) = G$ and $C \not\equiv G$

$B \not\equiv E$ since $\delta(B,1) = C$, $\delta(E,1) = F$ and $C \not\equiv F$

$B \not\equiv F$ since $\delta(B,1) = C$, $\delta(F,1) = G$ and $C \not\equiv G$

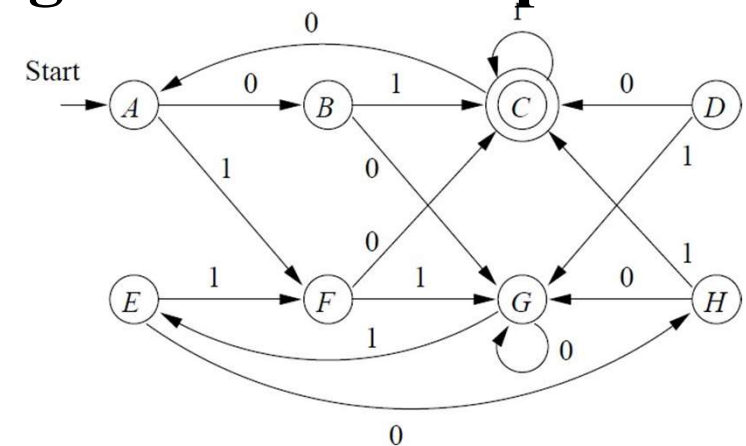
$B \not\equiv G$ since $\delta(B,1) = C$, $\delta(G,1) = E$ and $C \not\equiv E$

$B \equiv H$ since

- $\delta(B,0) = G$, $\delta(H,0) = G$ and $G \equiv G$
- $\delta(B,1) = C$, $\delta(H,1) = C$ and $C \equiv C$

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

Minimization of DFA: Table Filling Algorithm - Example 2



PASS 1: Consider column D

$D \not\equiv E$ since $\delta(D,0) = C$, $\delta(E,0) = H$ and $C \not\equiv H$

$D \equiv F$ since

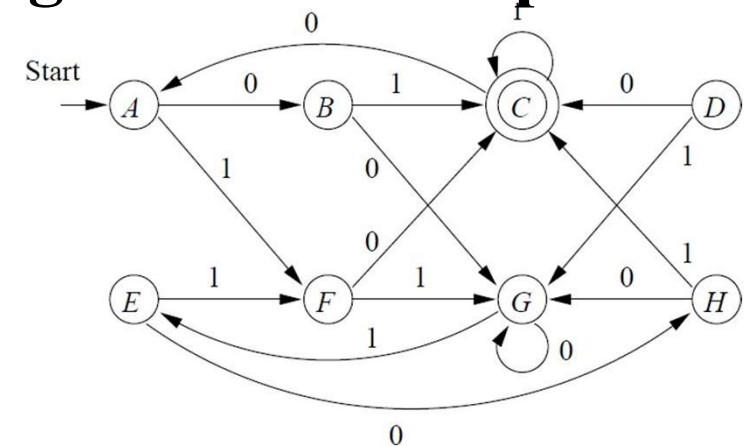
- $\delta(D,0) = C$, $\delta(F,0) = C$ and $C \equiv C$
- $\delta(D,1) = G$, $\delta(F,1) = G$ and $G \equiv G$

$D \not\equiv G$ since $\delta(D,0) = C$, $\delta(G,0) = G$ and $C \not\equiv G$

$D \not\equiv H$ since $\delta(D,0) = C$, $\delta(H,0) = G$ and $C \not\equiv G$

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

Minimization of DFA: Table Filling Algorithm - Example 2



PASS 1: Consider columns E, F, G

$E \neq F$ since $\delta(E,0) = H$, $\delta(F,0) = C$ and $H \neq C$

$E \neq G$ since $\delta(E,1) = F$, $\delta(G,1) = E$ and $F \neq E$

$E \neq H$ since $\delta(E,1) = F$, $\delta(H,1) = C$ and $F \neq C$

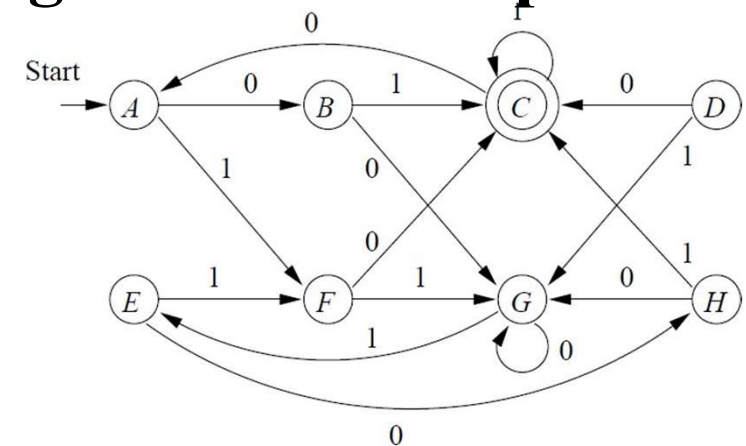
$F \neq G$ since $\delta(F,0) = C$, $\delta(G,0) = G$ and $C \neq G$

$F \neq H$ since $\delta(F,0) = C$, $\delta(H,0) = G$ and $C \neq G$

$G \neq H$ since $\delta(G,1) = E$, $\delta(H,1) = C$ and $E \neq C$

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

Minimization of DFA: Table Filling Algorithm - Example 2



PASS 2: Consider columns A, B, D

$A \equiv E$ since

- $\delta(A,0) = B$, $\delta(E,0) = H$ and $B \equiv H$
- $\delta(A,1) = F$, $\delta(E,1) = F$ and $F \equiv F$

$A \equiv G$ since $\delta(A,1) = F$, $\delta(G,1) = E$ and $F \not\equiv E$

$B \equiv H$ since

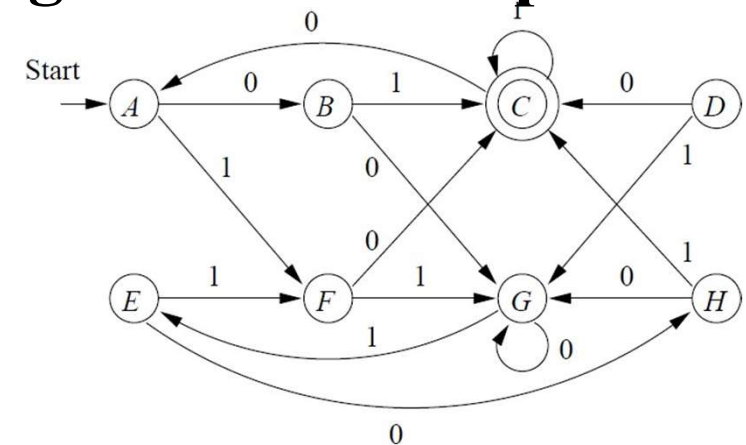
- $\delta(B,0) = G$, $\delta(H,0) = G$ and $G \equiv G$
- $\delta(B,1) = C$, $\delta(H,1) = C$ and $C \equiv C$

$D \equiv F$ since

- $\delta(D,0) = C$, $\delta(F,0) = C$ and $C \equiv C$
- $\delta(D,1) = G$, $\delta(F,1) = G$ and $G \equiv G$

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

Minimization of DFA: Table Filling Algorithm - Example 2



PASS 3: Consider columns A, B, D

$A \equiv E$ since

- $\delta(A,0) = B, \delta(E,0) = H$ and $B \equiv H$
- $\delta(A,1) = F, \delta(E,1) = F$ and $F \equiv F$

$B \equiv H$ since

- $\delta(B,0) = G, \delta(H,0) = G$ and $G \equiv G$
- $\delta(B,1) = C, \delta(H,1) = C$ and $C \equiv C$

$D \equiv F$ since

- $\delta(D,0) = C, \delta(F,0) = C$ and $C \equiv C$
- $\delta(D,1) = G, \delta(F,1) = G$ and $G \equiv G$

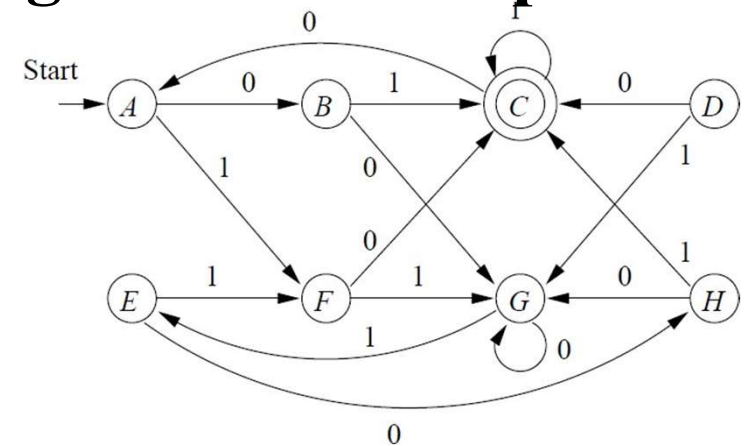
No new marked states in PASS 3. We are done, and we found all distinguishable states (marked ones).

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

Minimization of DFA: Table Filling Algorithm - Example 2

Equivalence Classes:

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

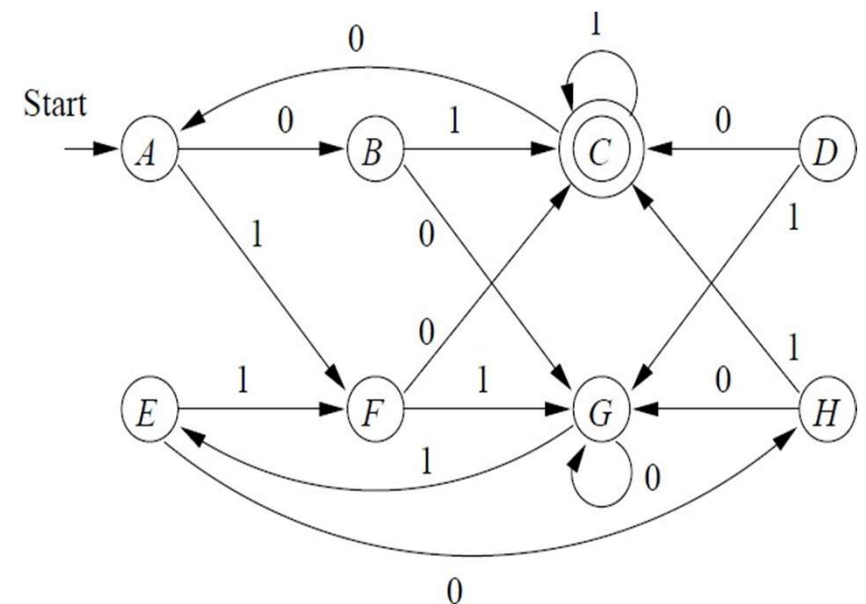
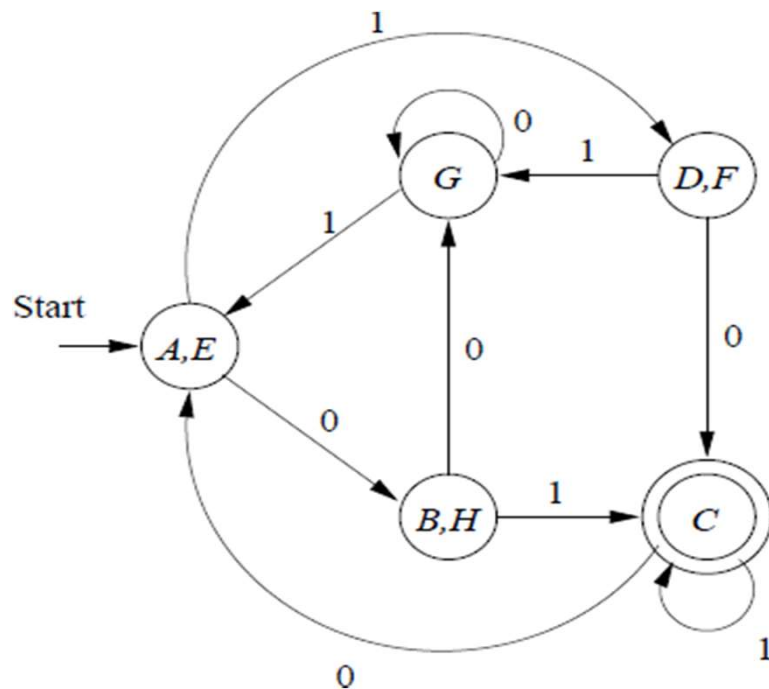


B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

Minimization of DFA: Table Filling Algorithm - Example 2

Equivalence Classes:

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



Testing Equivalence of Regular Languages with Table Filling Algorithm

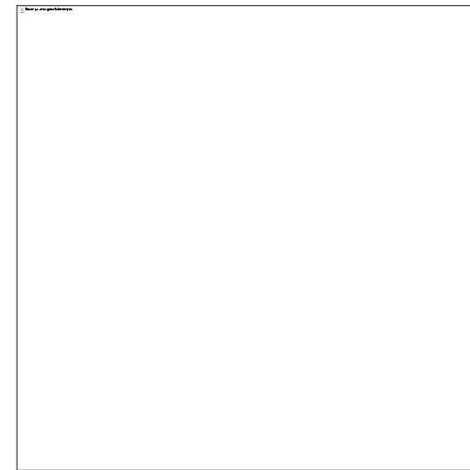
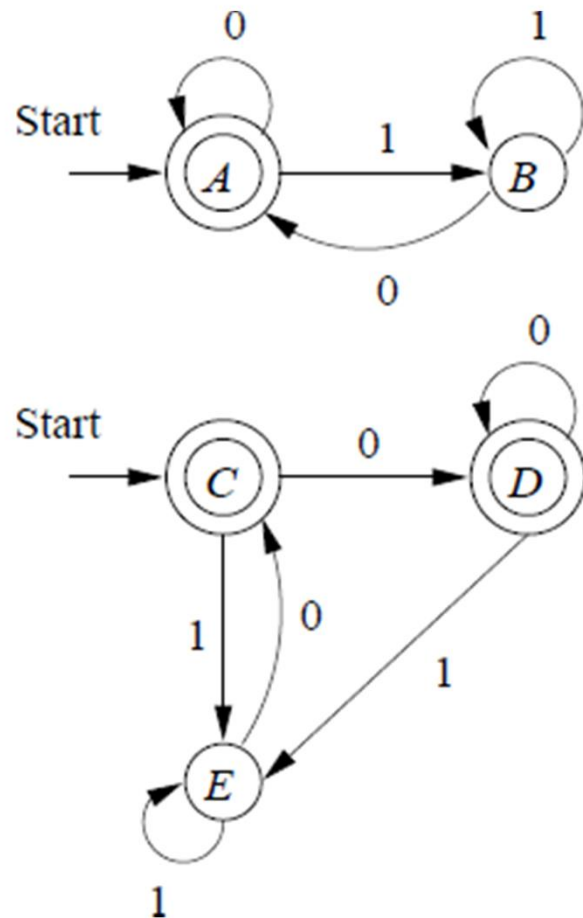
1st Approach to Test Equivalence:

- Minimize their DFAs,
- Check whether they are isomorphic (ie. they are same with renaming states)

2nd Approach to Test Equivalence:

- Let L and M be regular languages, to test whether $L = M$
- Create DFAs for both L and M
- Imagine the DFA which is a single DFA with the union of states
- Apply the table filling algorithm and find equivalent states
- If the table filling algorithm says that the two start states are distinguishable, then $L \neq M$, otherwise $L = M$.

Testing Equivalence of Regular Languages with Table Filling Algorithm - Example



- Since A and C are equivalent, these two DFAs are equivalent.
- Their languages are also equivalent.

The Pumping Lemma for Regular Languages

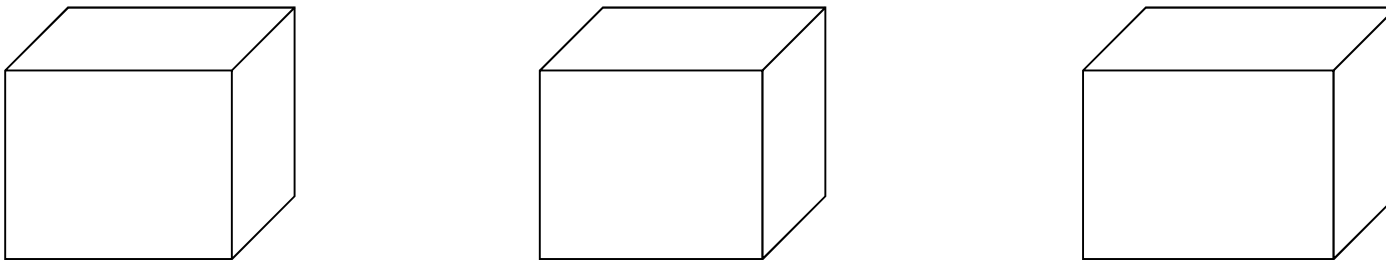
- There are languages which are NOT regular.
- How can we prove that a language L is not regular?
- We know that there is a DFA for each regular expression.
- **Prove that there is no DFA that accepts L :** This is not easy to prove
- Every regular language satisfies the **pumping lemma**.
- **A non-regular language can be shown that it is NOT regular using the pumping lemma.**
- $L_{01} = \{0^n 1^n \mid n \geq 1\}$ is not regular.
 - We can use the pumping lemma to show that this language is not regular.

The Pigeonhole Principle

4 pigeons

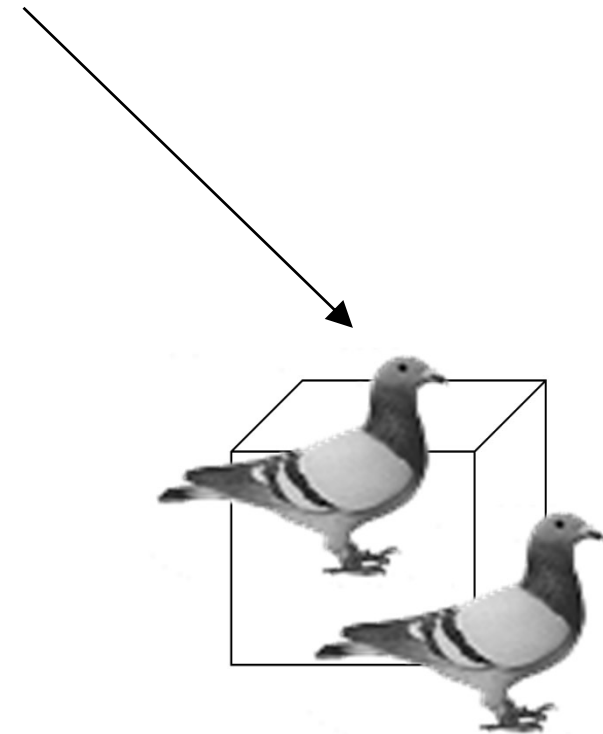
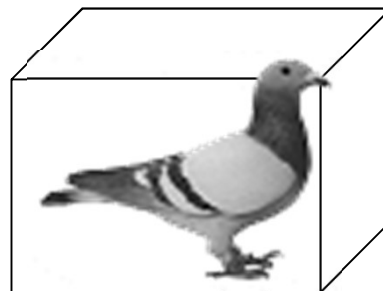
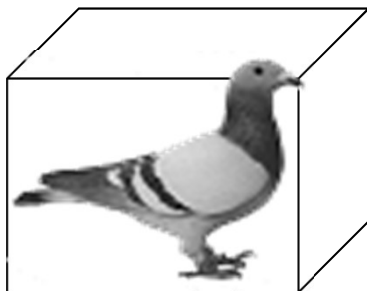


3 pigeonholes



The Pigeonhole Principle

A pigeonhole must
contain at least two pigeons



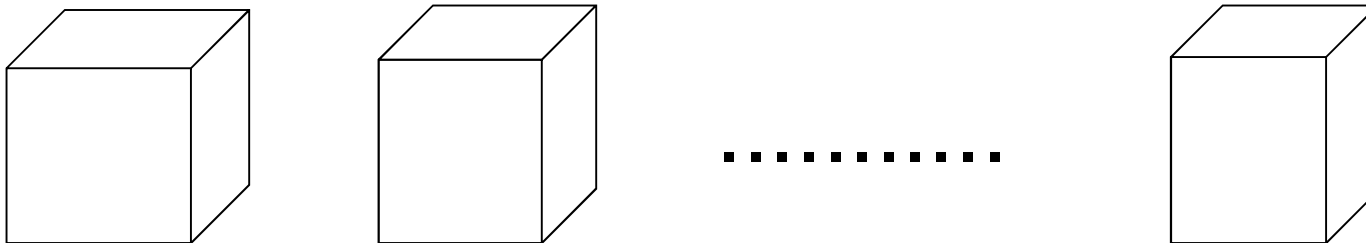
The Pigeonhole Principle

n pigeons



m pigeonholes

$n > m$



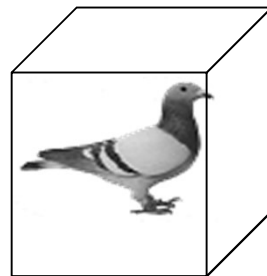
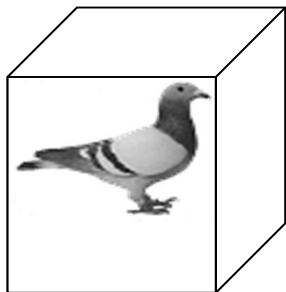
The Pigeonhole Principle

n pigeons

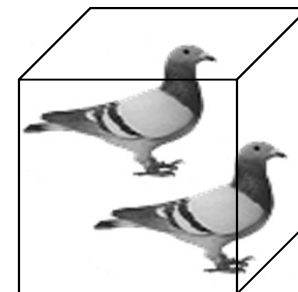
m pigeonholes

$$n > m$$

There is a pigeonhole
with at least 2 pigeons

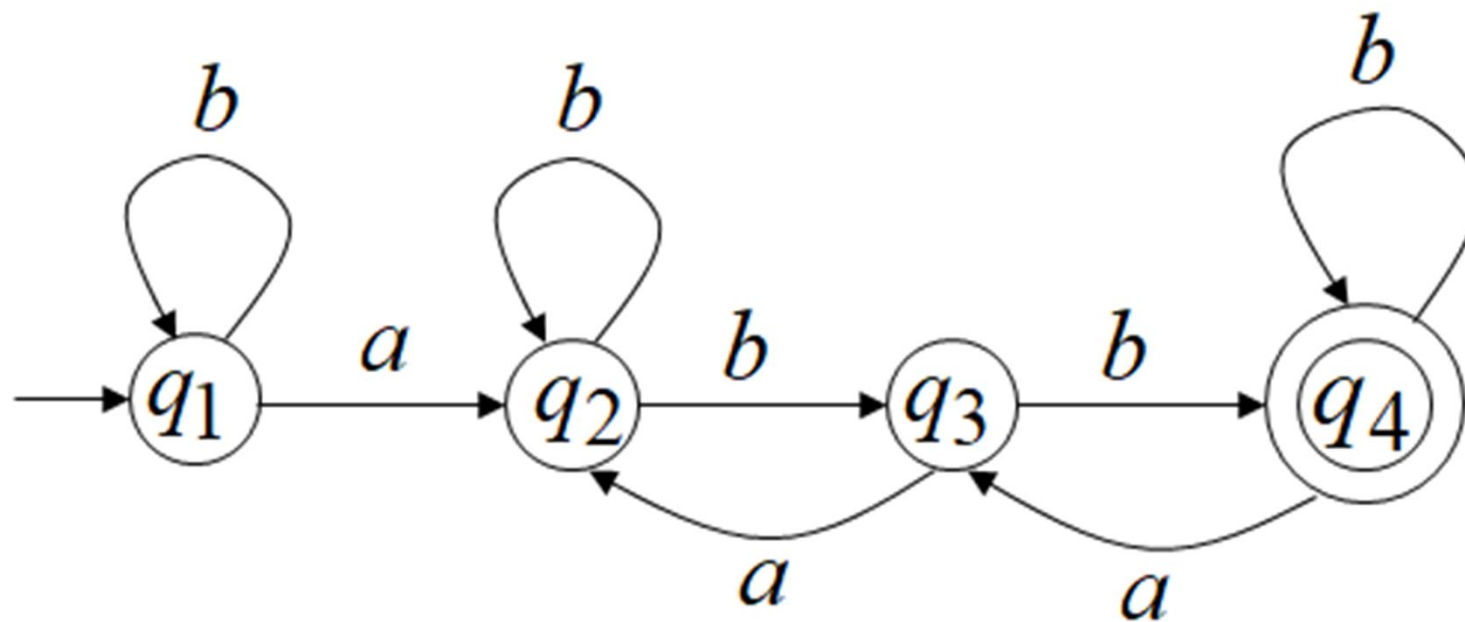


.....



The Pigeonhole Principle and DFAs

DFA with 4 states



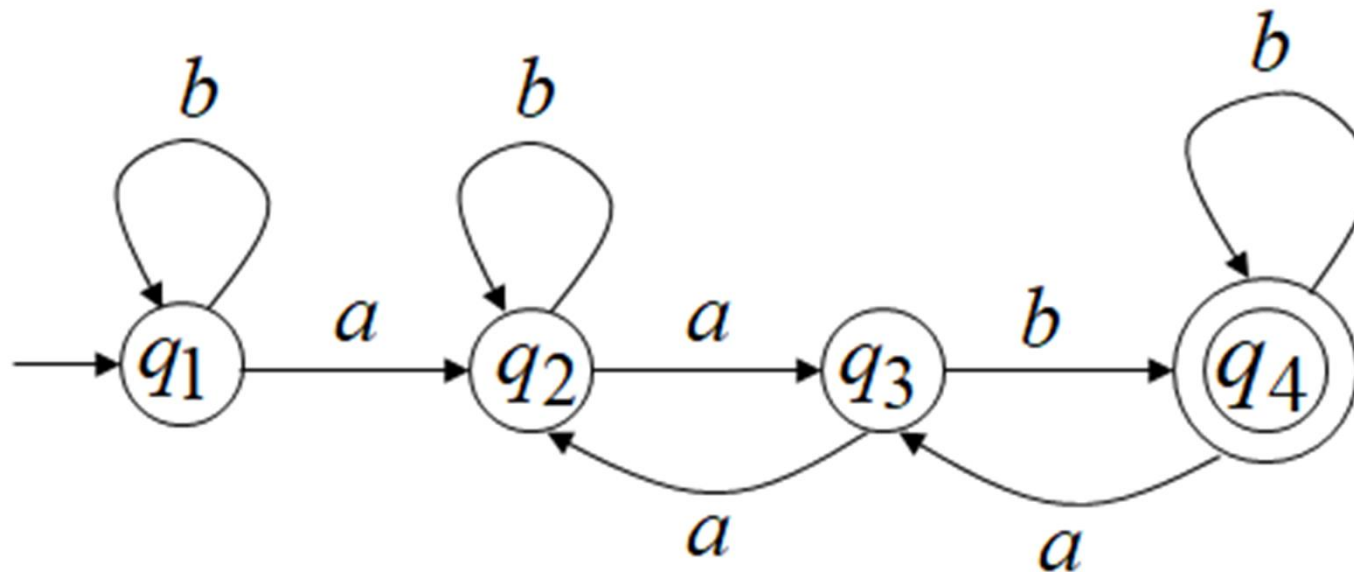
The Pigeonhole Principle and DFAs

In walks of strings: a

aa

no state is repeated

aab



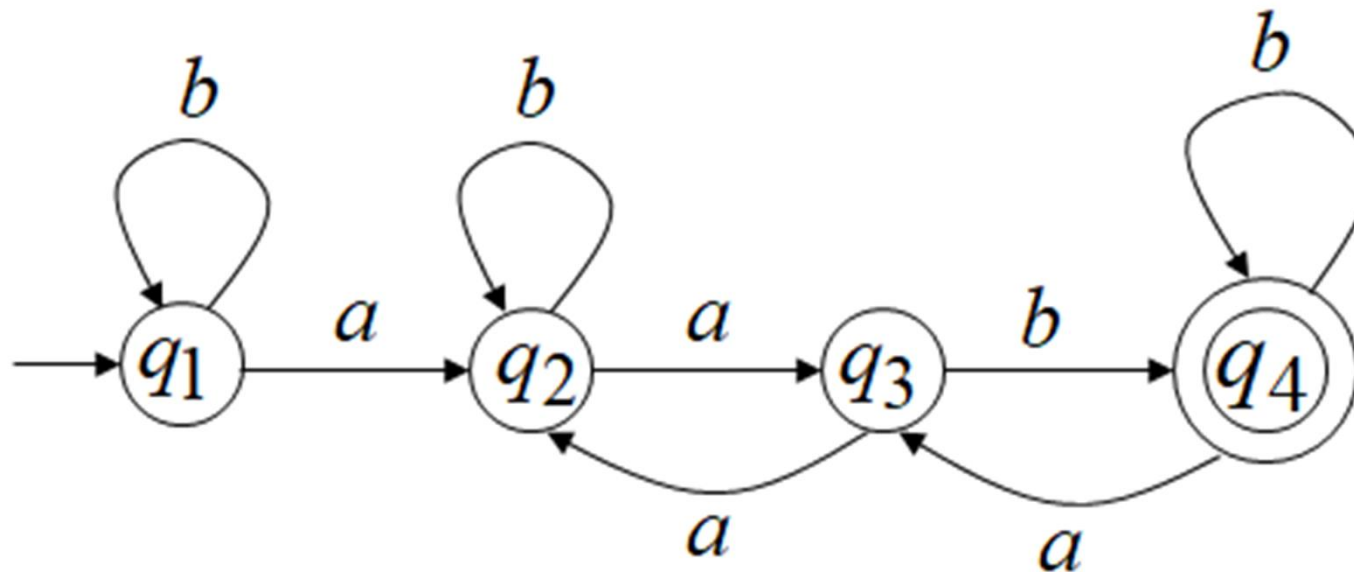
The Pigeonhole Principle and DFAs

In walks of strings: aabb

bbaa

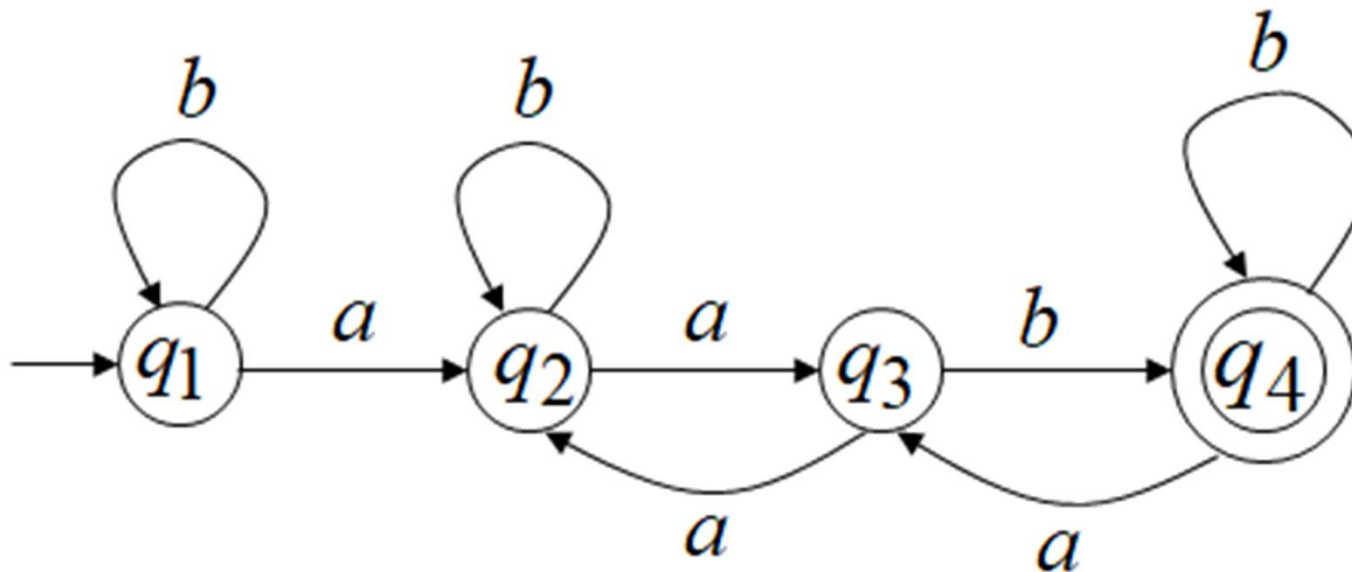
abbabb

a state is repeated



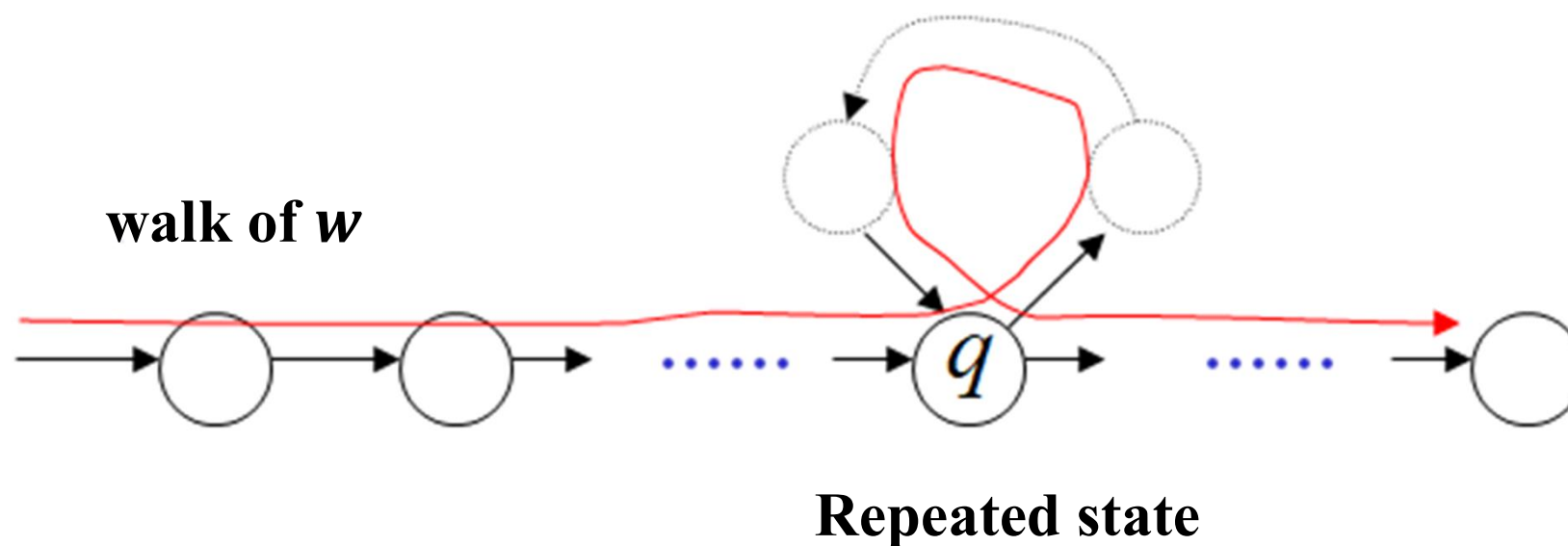
The Pigeonhole Principle and DFAs

- If string w has length $|w| \geq 4$
- Then the transitions of string w are more than the states of DFA
- Thus, **a state must be repeated**



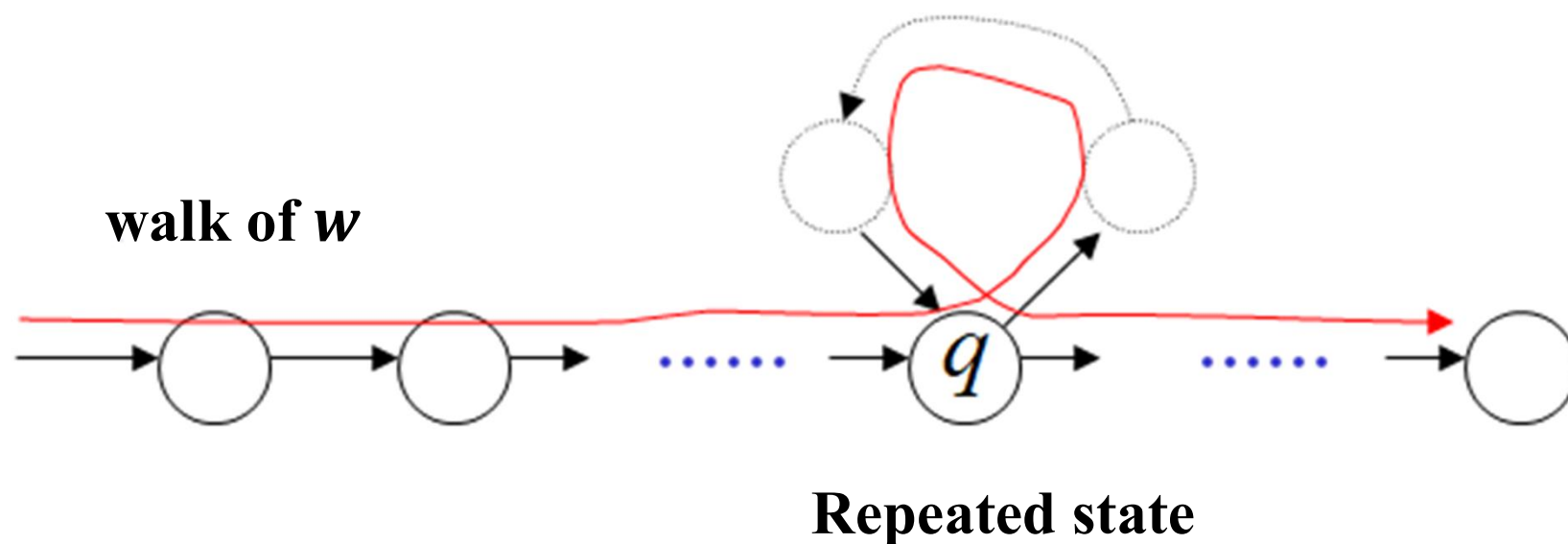
The Pigeonhole Principle and DFAs

- String w has length \geq number of states
- A state q must be repeated in the walk of w



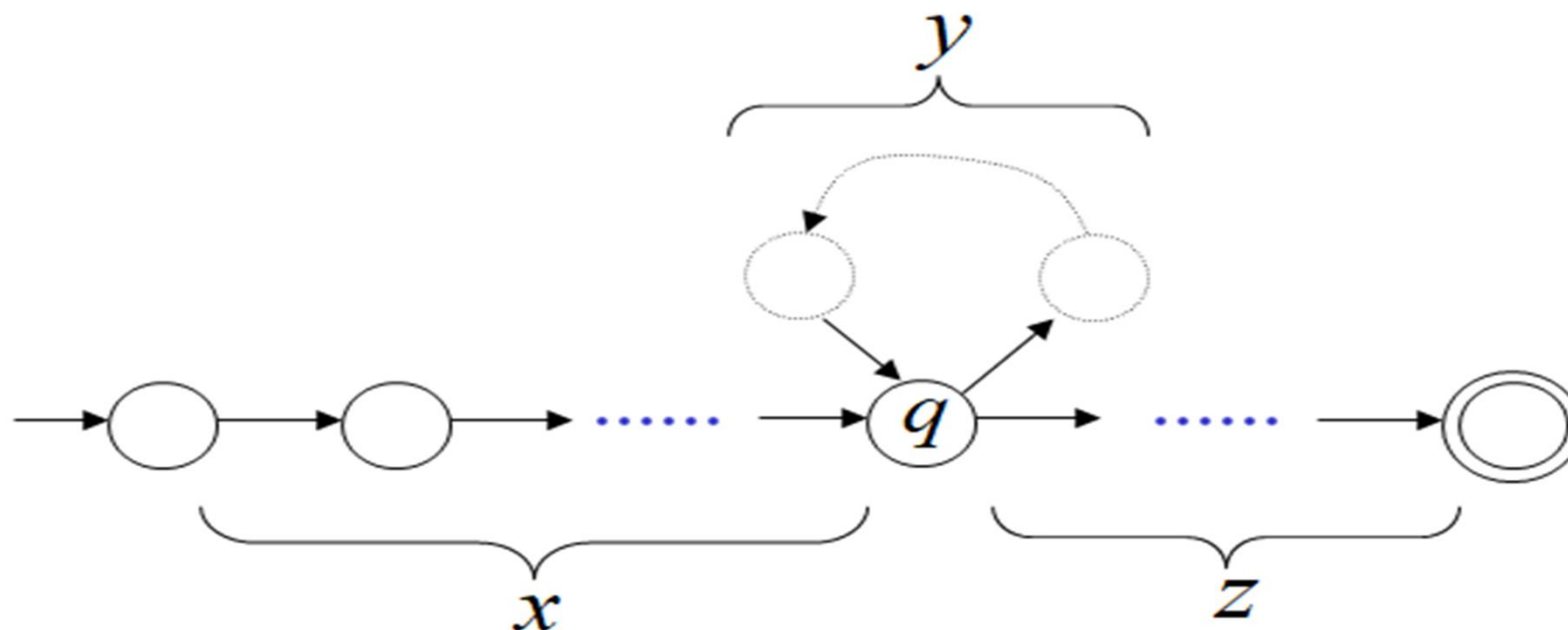
The Pigeonhole Principle and Pumping Lemma

- Take an infinite regular language L and a string w with $w \in L$
- There exists a DFA that accepts L with n states
- If string w has length $|w| \geq n$ (number of states of DFA)
- From the pigeonhole principle, a state is repeated in the walk w




The Pigeonhole Principle and Pumping Lemma

- $w = xyz$
- $|xy| \leq n$ (number of states of DFA)
- $|y| > 0$



The Pumping Lemma for Regular Languages

- Let L be a regular language.
- Then there exists a constant n such that for every string w in L such that $|w| > n$, we can break w into three strings, $w = xyz$, such that:
 1. $y \neq \epsilon$, i. e. $|y| > 0$
 2. $|xy| \leq n$ 
 3. For all $k \geq 0$, the string xy^kz is also in L .
- That is, we can always find a nonempty string y not too far from the beginning of w that can be "**pumped**"; that is, repeating y any number of times, or deleting it (the case $k=0$), keeps the resulting string in the language L .

Using the Pumping Lemma

- In order to show that a language L is NOT a regular language using the Pumping Lemma (Proof by Contradiction):
 1. Suppose L were a regular language.
 2. Then there is an integer n given us by the pumping lemma, which we do not know, we must plan for any possible n .
 3. Pick a string w which must be in L , it must be defined using n and $|w| \geq n$.
 - Tricky Part 1: You should find a string w so that you can create a contradiction in step 4. YOU CANNOT SELECT A SPECIFIC STRING.
 4. Break w into xyz , subject only to the constraints that $|xy| \leq n$ and $y \neq \epsilon$.
 5. Pick i and show that xy^iz is NOT in L in order to create a contradiction.
 - Tricky Part 2: You have to show that xy^iz is NOT in L using only the constraints that $|xy| \leq n$ and $y \neq \epsilon$. You may need to look at more than one cases. YOU CANNOT GIVE A SPECIFIC EXAMPLE.
 6. Conclude that L is NOT a regular language (proof by contradiction).

Applications of The Pumping Lemma – Example 1

Example 1: Let us show that the language $L_{01} = \{0^n 1^n \mid n \geq 1\}$ is NOT regular.

Proof: (proof by contradiction)

- Suppose L_{01} were a regular language (**our assumption**)
- Then, $w = 0^n 1^n \in L_{01}$ for any n
- By the pumping lemma, $w = xyz$, $|xy| \leq n$, $y \neq \varepsilon$ and $xy^kz \in L_{01}$.

$$w = \underbrace{000 \dots 0}_x \underbrace{0}_y \underbrace{0111 \dots 11}_z$$

- Since $y \neq \varepsilon$ and $|xy| \leq n$, y must contain only one or more 0s.
- If y repeats 0 times, $xy^0z = xz$ must be in L_{01} by the pumping lemma.
- xz has fewer 0's than 1's because y can only contain one or more 0s
- So, there is a contradiction with **our assumption** (L_{01} is regular)
- **Proof by contradiction**, we prove that L_{01} is NOT regular

Applications of The Pumping Lemma – Example 2

Example 2: Let us show that the language L_{eq} is the set of all strings with an equal number of 0's and 1's is NOT a regular language.

Proof: (proof by contradiction)

- The proof is exactly same as the proof of L_{01} .

Applications of The Pumping Lemma – Example 3

Example 3: Let us show that the language L_{pr} is the set of all strings of 1's whose length is a prime is NOT a regular language.

Proof: (proof by contradiction)

- Suppose L_{pr} were a regular language (**our assumption**)
- Choose a prime $p \geq n+2$ (this is possible since there are infinite number of primes.)

$$w = \underbrace{111 \dots 1}_{x} \underbrace{1}_{y} \underbrace{1111 \dots 11}_{z}$$

$|y|=m$

- Now, $i = p-m$ and $xy^{p-m}z \in L_{pr}$ by the pumping lemma.
- $|xy^{p-m}z| = |xz| + (p-m)|y| = (p-m) + (p-m)m = (1+m)(p-m)$
- But, $(1+m)(p-m)$ is not prime unless one of the factors is 1.
 - $y \neq \epsilon \rightarrow (1+m) > 1$
 - $m=|y| \leq |xy| \leq n$ and $p \geq n+2 \rightarrow (p-m) \geq (n+2)-n \geq 2$
- So, there is a contradiction with **our assumption**
 - \rightarrow Proof by contradiction, L_{pr} is NOT regular.

Applications of The Pumping Lemma – Example 4

Example 4: Show that the language L is the set of all strings of 0's and 1's that have an unequal number of 0's and 1's is NOT regular.

Proof: (proof by contradiction)

- It would be hard to use the pumping lemma directly to show that L is not regular. We can use closure properties of regular languages in addition to the pumping lemma in order to prove.
- Suppose that L were regular (**our assumption**)
- The complement of this language \bar{L} is the set of all strings of 0's and 1's that have equal number of 0's and 1's.
- By the *closure under complement theorem*, \bar{L} must be regular
- But, we showed that \bar{L} is NOT regular previously using the pumping lemma
- So, there is a contradiction with *our assumption* (L is regular)
- **Proof by contradiction**, we prove that L is NOT regular

Applications of The Pumping Lemma – Exercise

Exercise : Let us show that the language L is the set of all strings with the number of 0's is more the number of 1's is NOT a regular language.

Applications of The Pumping Lemma – Exercise

Exercise : Let us show that the language L is the set of all strings with the number of 0's is more the number of 1's is NOT a regular language.

Proof: (proof by contradiction)

- Suppose L were a regular language (**our assumption**)
- Then, $w = 0^{n+1}1^n \in L$ for any n since $n+1 > n$
- By the pumping lemma, $w = xyz$, $|xy| \leq n$, $y \neq \epsilon$ and $xy^kz \in L$.

$$w = \underbrace{000}_x \dots \underbrace{00}_y \underbrace{111 \dots 11}_z$$

- Since $0 \leq |xy| \leq n$, y must contain only 0's.
- By the pumping lemma, $xy^0z \in L$. But the number of 0's cannot be more than the number 1's because $0 < |y| \leq n$ and y must contain only 0's.
- So, there is a contradiction with **our assumption** (L is regular)
- **Proof by contradiction**, we prove that L is NOT regular