

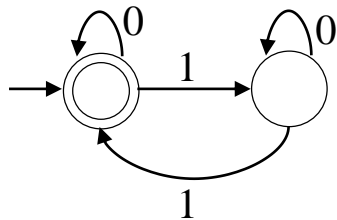
Properties of Regular Languages

- **Minimization and Equivalence of Automata**
- **Closure Properties of Regular Languages**
- **Decision Properties of Regular Languages**
- **The Pumping Lemma for Regular Languages**

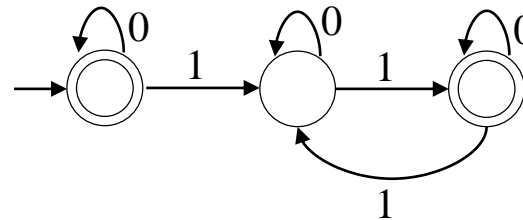
Minimization and Equivalence of Automata

DFA Minimization

- 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.

Indistinguishable States

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

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

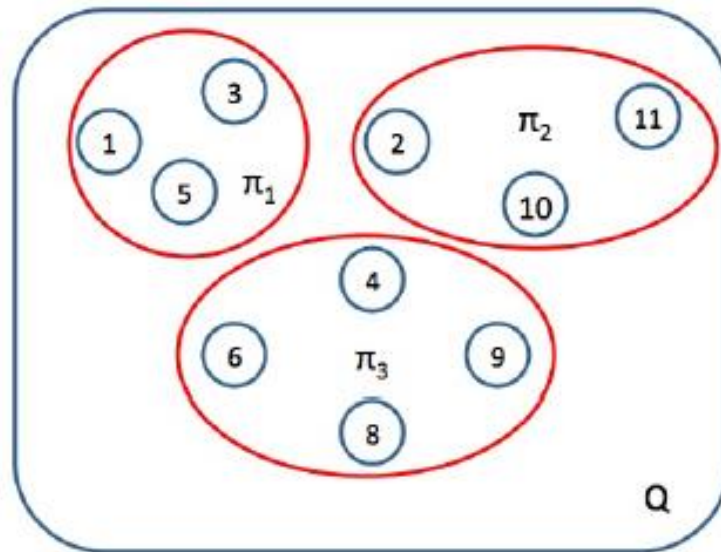
- This means that for all $w \in \Sigma^*$
 - $\widehat{\delta}(p, w) \in F$ iff $\widehat{\delta}(q, w) \in F$ and
 - $\widehat{\delta}(p, w) \notin F$ iff $\widehat{\delta}(q, w) \notin F$
- Two **indistinguishable states** *behave same for all possible strings*.
- Hence, a state p is **distinguishable**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

Indistinguishable States

- 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 indistinguishable 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.

Indistinguishable States

- An **equivalence relation** on a set of states Q induces a partitioning $\pi_1, \pi_2, \dots, \pi_k$ such that:
 - For all different i and j , $\pi_i \cap \pi_j = \phi$ and
 - $\bigcup_{i=1}^k \pi_i = Q$



Finding Distinguishable States – Table Filling Algorithm

- We can compute distinguishable states with an **inductive table filling algorithm**.

Basis:

- **Any non-accepting state is distinguishable from any accepting state.**

Induction:

- **States p and q are distinguishable if there is some input symbol a such that $\delta(p,a)$ is distinguishable from $\delta(q,a)$.**
- All other pairs of states are **indistinguishable**, and can be merged appropriately.

Finding Distinguishable States – Table Filling Algorithm

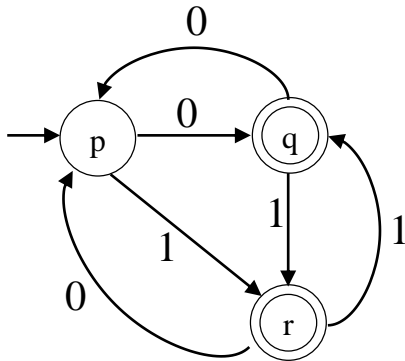
- Consider all pairs of states (p,q)
 - if $p \in F$ and $q \notin F$ or $p \notin F$ and $q \in F$, mark (p,q) as distinguishable
- Repeat the following until no previously unmarked pairs are marked:
 - $\forall p,q \in Q$ and $\forall a \in \Sigma$, find $\delta(p,a)=r$ and $\delta(q,a)=s$,
 - if (r,s) is marked as distinguishable then mark (p,q) as distinguishable.

Minimization of DFA

Table Filling Algorithm

- We can also use table filling algorithm to minimize a DFA by merging all equivalent states.
- That is, we replace a state p with its *equivalence class* found by the table filling algorithm.
- **Equivalence Classes:** $\pi = \{\pi_1, \pi_2, \dots, \pi_k\}$
 - So, each equivalence class π_i will be a state in the minimized DFA.
 - For each symbol a , $\delta_{\min}(\pi_i, a) = \pi_j$ where there are states p, q $p \in \pi_i$ $q \in \pi_j$ such that $\delta(p, a) = q$

Table Filling Algorithm: Example 1



q	x	
r	x	
	p	q

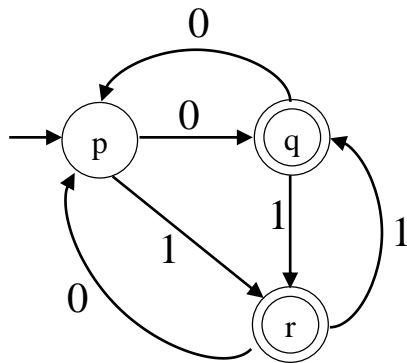
- p is distinguishable from q and r by basis, **mark them**

q	x	
r	x	
	p	q

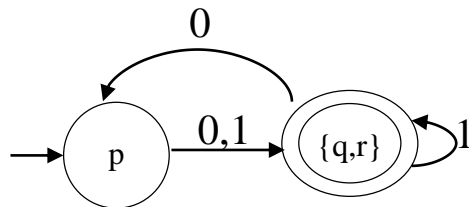
- Both q and r go to p with 0, so no string beginning with 0 will distinguish them
- Starting in either q and r, an input of 1 takes us to either,
- so they are indistinguishable.

- Equivalence relation partitions (equivalence classes): $\{ \{p\}, \{q,r\} \}$

Table Filling Algorithm: Example 1



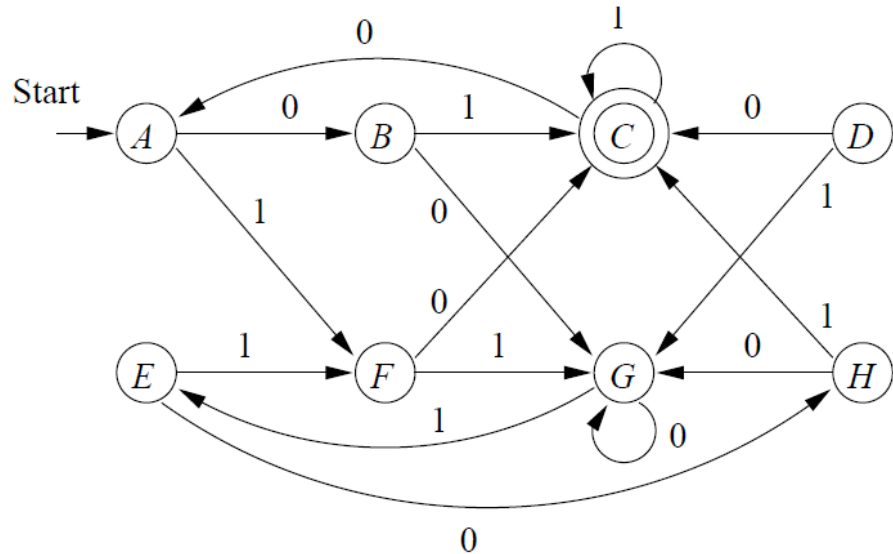
- Equivalence relation partitions (equivalence classes): $\{ \{p\}, \{q,r\} \}$
 - q and r are **indistinguishable**.



DFA with minimal states

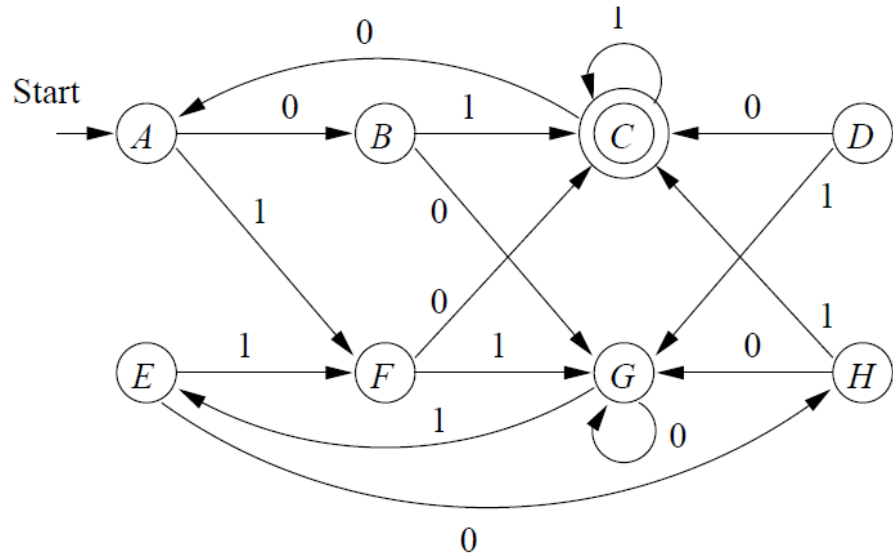
Minimization of DFA

Table Filling Algorithm: Example 2



Minimization of DFA

Table Filling Algorithm: Example 2



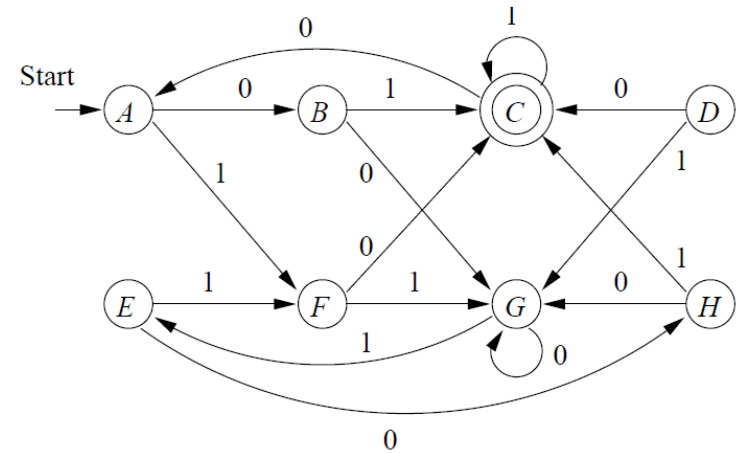
B							
C	x	x					
D			x				
E			x				
F			x				
G			x				
H			x				
	A	B	C	D	E	F	G

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 $B \not\equiv C$

$A \equiv E$ since

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

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

$A \equiv G$ since

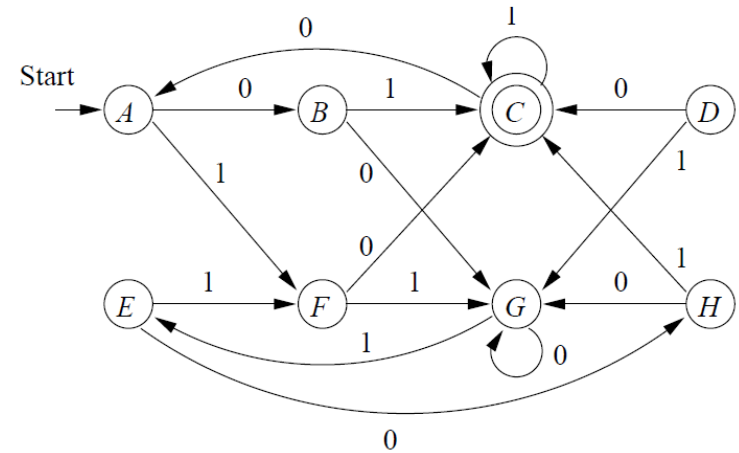
- $\delta(A,0)=B$, $\delta(G,0)=G$ and $B \equiv G$
- $\delta(A,1)=F$, $\delta(G,1)=E$ and $F \equiv E$

$A \not\equiv H$ 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 $C \not\equiv E$

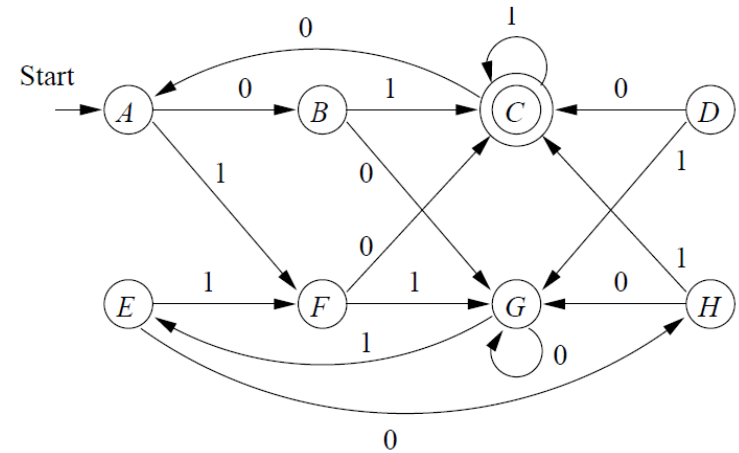
$B \equiv H$ since

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

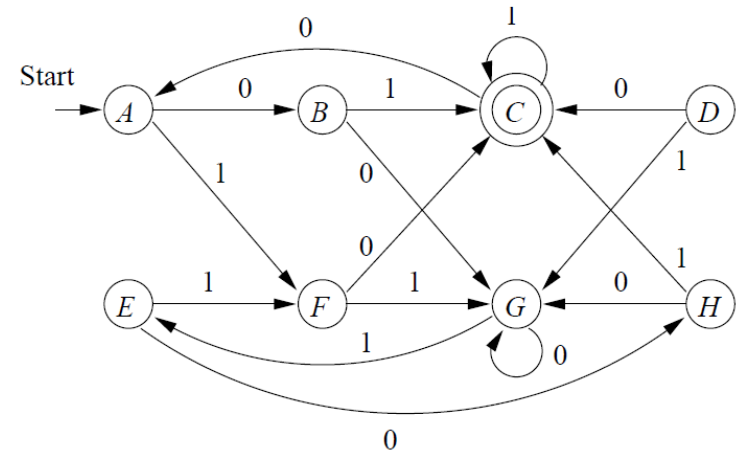
$D \not\equiv G$ since $\delta(D,0)=C$, $\delta(G,0)=G$ and $C \not\equiv G$

$D \not\equiv H$ 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

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

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

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

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

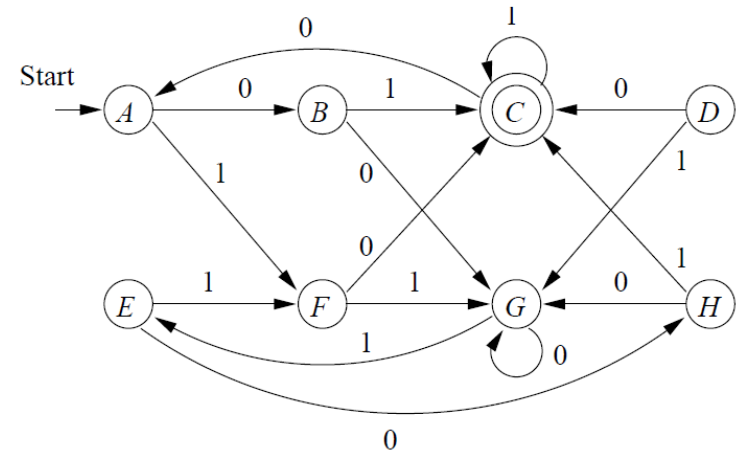
$F \not\equiv H$ since $\delta(F,0)=C$, $\delta(H,0)=G$ and $C \not\equiv G$

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

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 $C \equiv C$

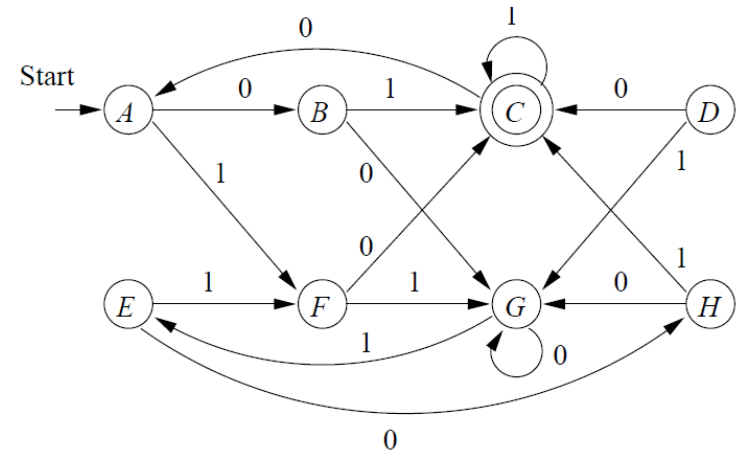
$D \equiv F$ since

- $\delta(D,0)=C$, $\delta(F,0)=C$ 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

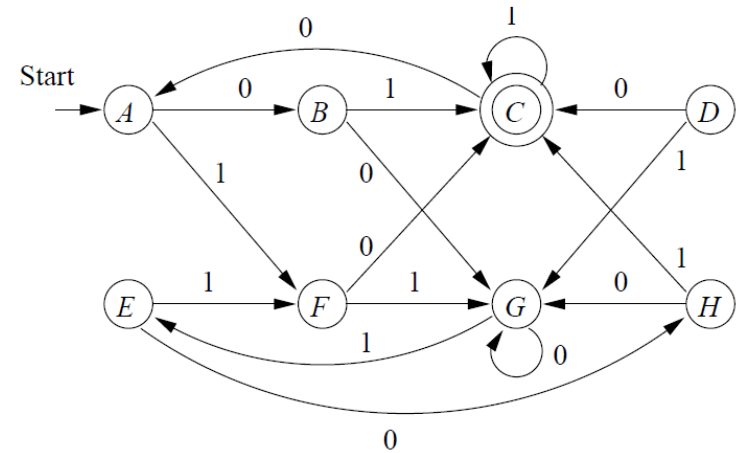
- $\delta(D,0)=C$, $\delta(F,0)=C$ and $C \equiv C$
- $\delta(D,1)=G$, $\delta(F,1)=G$ and $G \equiv G$

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:

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

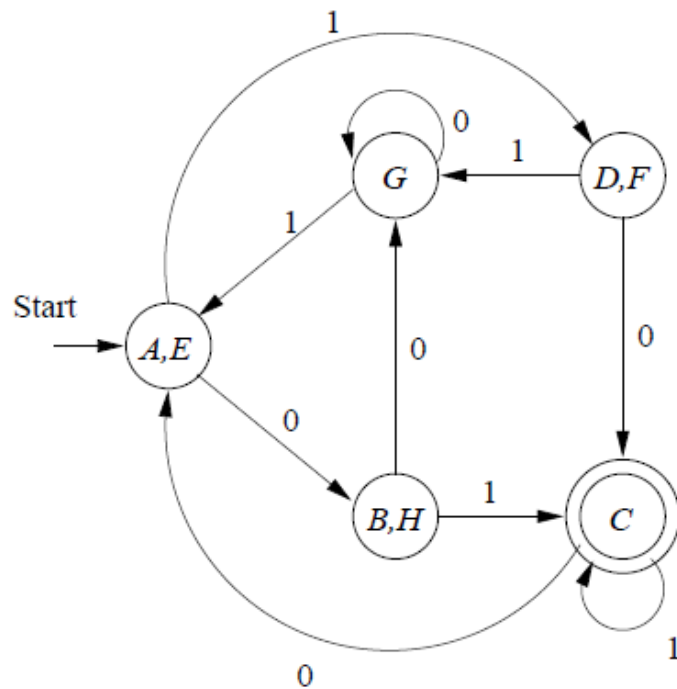
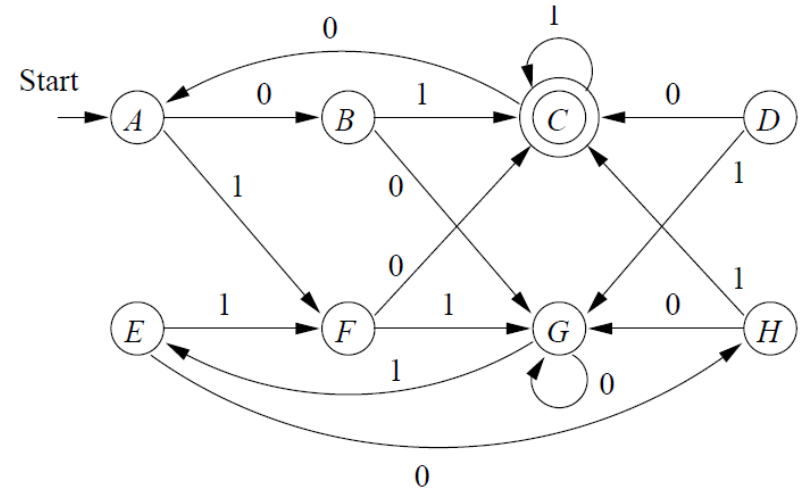
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x
	A	B	C	D	E	F	G

Minimization of DFA

Table Filling Algorithm: Example 2

Equivalence Classes:

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



Is the Minimized DFA Really Minimal?

- Let M be the **minimized DFA** found by the table filling algorithm and assume that its states $P = \{p_0, \dots, p_m\}$ and its transition function is δ .
- Since **all states of M are distinguishable (since M is minimal)**, there must be distinct strings w_1, \dots, w_m such that $\hat{\delta}(p_0, w_i) = p_i$ for all i .
- Suppose there is an equivalent DFA M_1 with transition function δ_1 but with fewer states $Q = \{q_0, \dots, q_n\}$ where $n < m$.

Is the Minimized DFA Really Minimal?

- Since M_1 has fewer states than M , then there must be strings w_k and w_j among distinct strings w_1, \dots, w_m such that $\hat{\delta}_1(q_0, w_k) = \hat{\delta}_1(q_0, w_j)$ (pigeonhole principle)
- Since p_k and p_j are distinguishable, there must be some string x such that
 - $\hat{\delta}(p_0, w_k \cdot x) = \hat{\delta}(p_k, x)$ is a final state and
 - $\hat{\delta}(p_0, w_j \cdot x) = \hat{\delta}(p_j, x)$ is NOT a final state, or vice versa.
 - So $w_k \cdot x$ is accepted and $w_j \cdot x$ is not (or vice versa)

- But

$$\hat{\delta}_1(q_0, w_k \cdot x) = \hat{\delta}_1(\hat{\delta}_1(q_0, w_k), x) = \hat{\delta}_1(\hat{\delta}_1(q_0, w_j), x) = \hat{\delta}_1(q_0, w_j \cdot x)$$

- So M_1 either accepts both $w_k \cdot x$ and $w_j \cdot x$ or rejects both.
- **So M_1 and M can not be equivalent. So M_1 can not exist.**

Testing Equivalence of Regular Languages with Table Filling Algorithm

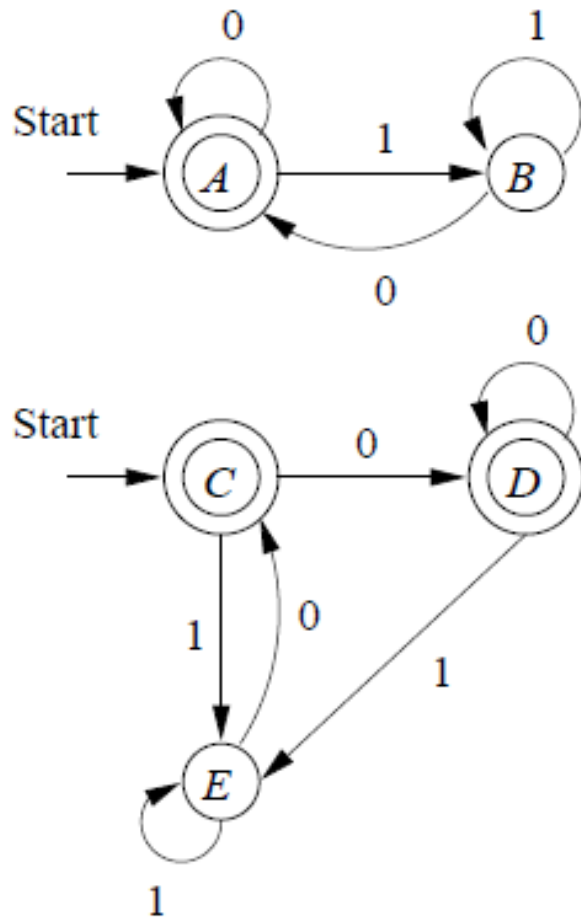
1st Approach to Test Equivalence:

- 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 that is the union of the two DFA's (never mind there are two start 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



<i>B</i>	<i>x</i>			
<i>C</i>		<i>x</i>		
<i>D</i>		<i>x</i>		
<i>E</i>	<i>x</i>		<i>x</i>	<i>x</i>
	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>

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

Closure Properties of Regular Languages

Closure Properties of Regular Languages

- *Closure Properties of Regular Languages* are the theorems indicate that the regular languages are closed under certain operations.
- A closure property of regular languages say that
 - **If a language is created from regular languages using the operation mentioned in the theorem, it is also a regular language.**
- Closure properties of regular languages also indicate that how their DFAs can be created from DFAs of other regular languages using certain operations.

Principal Closure Properties of Regular Languages

1. The **union** of two regular languages is regular.
2. The **concatenation** of regular languages is regular.
3. The **closure (star)** of a regular language is regular.
4. The **complement** of a regular language is regular.
5. The **intersection** of two regular languages is regular.
6. The **difference** of two regular languages is regular.
7. The **reversal** of a regular language is regular.
8. A **homomorphism** (substitution of strings for symbols) of a regular language is regular.
9. The **inverse homomorphism** of a regular language is regular.

Union of Two Regular Languages is Regular

Theorem: (closure under union)

If L and M are regular languages, then $L \cup M$ is regular.

Proof:

- Since L and M are regular, they have regular expressions; say $L = L(R)$ and $M = L(S)$.
- Then $L \cup M = L(R+S)$ by the definition of the $+$ operator for regular expressions.
- Thus, $L \cup M$ is regular.

Concatenation of Regular Languages is Regular

Theorem: (closure under concatenation)

If L and M are regular languages, then LM is regular.

Proof:

- Since L and M are regular, they have regular expressions; say $L=L(R)$ and $M=L(S)$.
- Then $LM=L(RS)$ by the definition of the concatenation operator for regular expressions.
- Thus, LM is regular

Closure of a Regular Language is Regular

Theorem: (closure under star)

If L is a regular language, then L^* is regular.

Proof:

- Since L is regular, it has a regular expression; say $L=L(R)$.
- Then $L^*=L(R^*)$ by the definition of the closure operator for regular expressions.
- Thus, L^* is regular.

Complement of a Regular Language is Regular

Theorem: (closure under complement)

If L is a regular language over alphabet Σ , then its complement $\bar{L} = \Sigma^* - L$ is also a regular language.

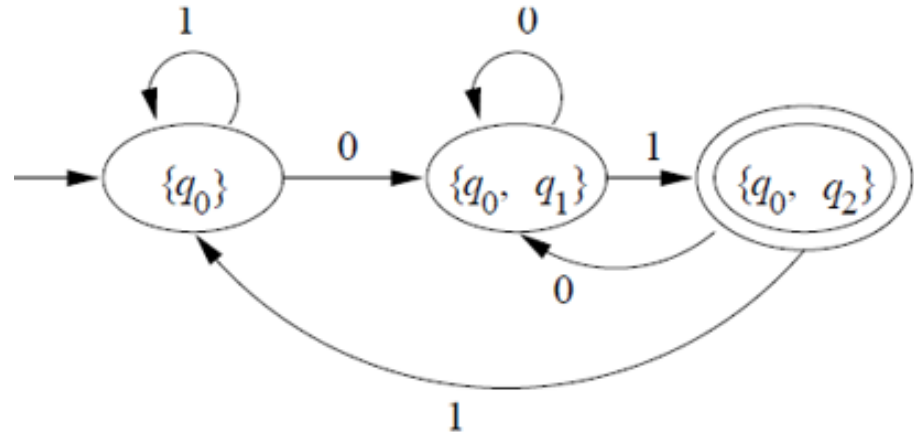
Proof:

- Let $L = L(A)$ for some DFA $A = (Q, \Sigma, \delta, q_0, F)$.
- Then $\bar{L} = L(B)$, where B is the DFA $B = (Q, \Sigma, \delta, q_0, Q - F)$.
- That is, B is exactly like A , but the accepting states of A have become non-accepting states of B , and vice versa.
- Then, w is in $L(B)$ if and only if $\hat{\delta}(q_0, w) \in Q - F$, which occurs if and only if w is not in $L(A)$.
- Thus, \bar{L} is regular.

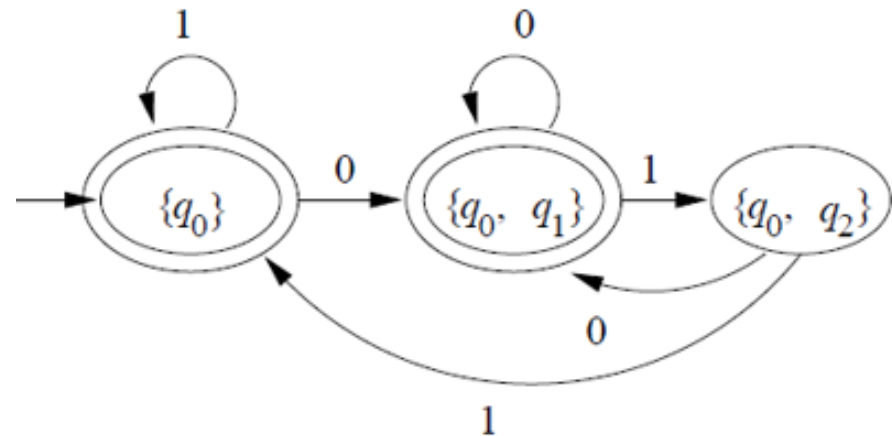
Closure Under Complement - Example

$L=L(A)$: strings of 0's and 1's
that end in 01;

In regular-expression terms,
 $L(A) = L((0+1)^*01)$.



$L=L(A)$: strings of 0's and 1's
that do not end in 01;



Intersection of Two Regular Languages is Regular

Theorem: (closure under intersection)

If L and M are regular languages, then $L \cap M$ is regular.

Proof: (Simple Proof)

- By DeMorgan's law $L \cap M = \overline{\overline{L} \cup \overline{M}}$.
- We already know that regular languages are closed under complement and union.
- So, $L \cap M$ is regular when L and M are regular.

Closure Under Intersection

DFA Construction Proof

Theorem: (closure under intersection)

If L and M are regular languages, then $L \cap M$ is regular.

Proof: (DFA Construction Proof)

- Let L be the language of DFA $A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$ and M be the language of DFA $A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$.
- We will construct an automaton that simulates A_L and A_M in parallel, and accepts if and only if both A_L and A_M accept.
- If A_L goes from state p to state s on reading \mathbf{a} , and A_M goes from state q to state t on reading \mathbf{a} , then $A_{L \cap M}$ will go from state (p,q) to state (s,t) on reading \mathbf{a} .
- Formally, $A_{L \cap M} = (Q_L \times Q_M, \Sigma, \delta_{L \cap M}, (q_L, q_M), F_L \times F_M)$ where

$$\delta_{L \cap M}((p,q), \mathbf{a}) = (\delta_L(p, \mathbf{a}), \delta_M(q, \mathbf{a}))$$

Closure Under Intersection

DFA Construction Proof (cont.)

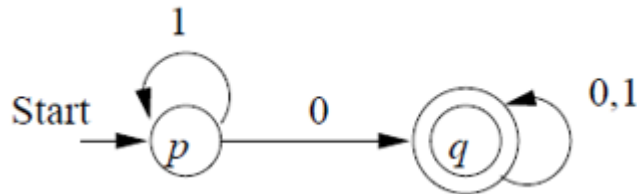
- By induction $|w|$, it can be shown that

$$\hat{\delta}_{L \cap M}((q_L, q_M), w) = (\hat{\delta}_L(q_L, w), \hat{\delta}_M(q_M, w))$$

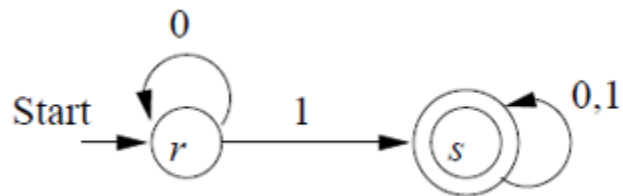
- To see why $L(A_{L \cap M}) = L(A_L) \cap L(A_M)$, first we can observe that an induction on $|w|$ proves that $\hat{\delta}_{L \cap M}((q_L, q_M), w) = (\hat{\delta}_L(q_L, w), \hat{\delta}_M(q_M, w))$.
- But $A_{L \cap M}$ accepts w if and only if $\hat{\delta}_{L \cap M}((q_L, q_M), w)$ is a pair of accepting states.
- That is, $\hat{\delta}_L(q_L, w)$ must be in F_L and $\hat{\delta}_M(q_M, w)$ must be in F_M .
- w is accepted by $A_{L \cap M}$ if and only if both A_L and A_M accept w .
- Thus, $A_{L \cap M}$ accepts the intersection of $L(A_L)$ and $L(A_M)$.

Closure Under Intersection

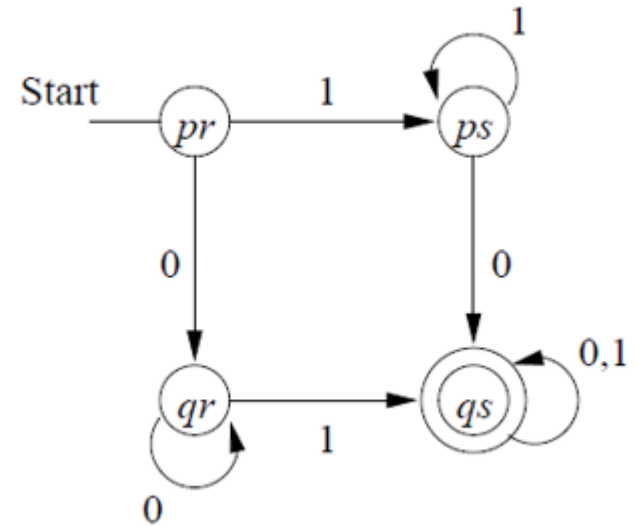
Product Construction -- Example



DFA A_L



DFA A_M



DFA $A_{L \cap M}$

- DFA $A_{L \cap M}$ is the cross product of A_L and A_M .
- $L(A_{L \cap M}) = L(A_L) \cap L(A_M)$.
 - $L(A_L)$ is the set of strings containing at least one 0.
 - $L(A_M)$ is the set of strings containing at least one 1.
 - $L(A_{L \cap M})$ is the set of strings containing at least one 0 and one 1.

Difference of Two Regular Languages is Regular

Theorem: (closure under difference)

If L and M are regular languages, then $L-M$ is regular.

Proof:

- We can observe that $L - M = L \cap \bar{M}$.
- By closure under complement, \bar{M} is regular.
- By closure under intersection, $L \cap \bar{M}$ is regular.
- Thus, $L-M$ is regular.

Reversal of a Regular Language is Regular

- The reversal of a string $a_1a_2 \dots a_n$ is the string written backwards, that is, $a_n a_{n-1} \dots a_1$.
- We use w^R for the reversal of string w .
 - Thus, 0010^R is 0100 , and
 - $\varepsilon^R = \varepsilon$.
- The reversal of a language L , written L^R is the language consisting of the reversals of all its strings.
 - For instance, if $L = \{001, 10, 111\}$, then $L^R = \{100, 01, 111\}$.
- Reversal is another operation that preserves regular languages; that is, if L is a regular language, so is L^R .

Reversal of a Regular Language is Regular

Theorem: (closure under reversal)

If L is a regular language, then its reversal L^R is regular.

Proof: (proof by automaton creation)

- Let L be recognized by a Finite Automaton A .
- Turn A into a FA for L^R , by
 1. Reverse all the arcs in the transition diagram for A ,
 2. Make the start state of A be the only accepting state for the new automaton.
 3. Create a new start state p_0 with transitions on ε to all the accepting states of A .
- The result is an automaton that, simulates A "in reverse" and therefore accepts a string w if and only if A accepts w^R .

Closure Under Reversal

Induction Proof on Regular Expressions

Theorem: (closure under reversal)

If L is a regular language, then its reversal L^R is regular.

Proof: (induction proof on regular expressions)

- Assume L is defined by regular expression E .
- The proof is a structural induction on the size of E .
- We can show that there is another regular expression E^R such that $L(E^R) = (L(E))^R$; that is, the language of E^R is the reversal of the language of E .

Closure Under Reversal

Induction Proof on Regular Expressions (cont.)

BASIS:

- If E is ϵ , ϕ , or a , for some symbol a , then E^R is the same as E .
- We know $\{\epsilon\}^R = \{\epsilon\}$, $\phi^R = \phi$, and $\{a\}^R = \{a\}$.

INDUCTION:

- There are three cases, depending on the form of E .

Case 1. $E = F + G \rightarrow E^R = F^R + G^R$

- The reversal of the union of two languages is obtained by computing the reversals of the two languages and taking the union of those languages.
- So, $L(F^R + G^R) = (L(F + G))^R$

Closure Under Reversal

Induction Proof on Regular Expressions (cont.)

Case 2. $E = F G \rightarrow E^R = G^R F^R$

- We reverse the order of the two languages, as well as reversing the languages themselves.
 - For instance, if $L(F) = \{01, 111\}$ and $L(G) = \{00, 10\}$, then $L(FG) = \{0100, 0110, 11100, 111100\}$.
 - Its reversal is $\{0010, 0110, 00111, 01111\}$
 - If we concatenate the reversals of $L(G)$ and $L(F)$ in that order, we get $\{00, 01\}\{10, 111\} = \{0010, 00111, 0110, 01111\}$ which is the same language as $(L(FG))^R$.
- In general, if a word w in $L(E)$ is the concatenation of x from $L(F)$ and y from $L(G)$, then $w^R = y^R x^R$
- So, $L(G^R F^R) = (L(FG))^R$

Closure Under Reversal

Induction Proof on Regular Expressions (cont.)

Case 3. $E = F^* \rightarrow E^R = (F^R)^*$

- Let w be a string in $L(F^*)$ that can be written as $w_1 w_2 \dots w_n$, where each w_i is in $L(F)$.
- Then, $w^R = w_n^R w_{n-1}^R \dots w_1^R$
- Since each w_i^R is in $L(F^R)$, w^R is in $L((F^R)^*)$
- This means that “if w is in $L(F^*)$, then w^R is in $L((F^R)^*)$ ”
- Conversely, let w be a string in $L((F^R)^*)$ that can be written as $w_1 w_2 \dots w_n$, where each w_i is the reversal of a string in $L(F)$.
- Since each w_i^R is in $L(F)$, w^R is in $L(F^*)$
- This means that “if w is in $L((F^R)^*)$, then w^R is in $L(F^*)$ ”
- Thus, w is in $L(F^*)$ if and only if w^R is in $L((F^R)^*)$
- $L((F^R)^*) = (L(F^*))^R$

Closure Under Reversal -- Example

- Let M be $L((0+1)0^*)$
- Then M^R (ie $(L((0+1)0^*))^R$) can be found as follows:

$$\begin{aligned}M^R &= (L((0+1)0^*))^R \\&= L(((0+1)0^*))^R \\&= L((0^*)^R(0+1)^R) \\&= L((0^R)^*(0^R+1^R)) \\&= L(0^*(0+1))\end{aligned}$$

Homomorphism of a Regular Language is Regular

- A **homomorphism** on an alphabet is a function h that gives a string for each symbol in that alphabet.
 - Suppose Σ and Γ are alphabets, the function $h: \Sigma \rightarrow \Gamma^*$ is called a **homomorphism**.
- Extend to strings by $h(a_1 \dots a_n) = h(a_1) \dots h(a_n)$.
- Extend to languages $h(L) = \{ h(w) \mid w \in L \}$

Example:

- $h(0) = ab; h(1) = \epsilon$.
- $h(01010) = ababab$.
- $h(\{010110, 11, 1001\}) = \{ababab, \epsilon, abab\}$

Homomorphism of a Regular Language is Regular

Theorem: (closure under homomorphism)

If L is a regular language and h is a homomorphism on its alphabet, then $h(L)=\{ h(w) \mid w \in L \}$ is regular.

Proof:

- Let E be a regular expression for L .
- Apply h to each symbol in E .
- Language of the resulting regular expression is $h(L)$.
- Thus, $h(L)$ is regular.

Closure Under Homomorphism -- Example

- Let $h(0) = ab$; $h(1) = \epsilon$.
- Let L be the language of regular expression $01^* + 10^*$.
- Then $h(L)$ is the language of regular expression $ab(\epsilon)^* + \epsilon(ab)^*$.

$$ab(\epsilon)^* + \epsilon(ab)^* = ab + (ab)^* = (ab)^*$$

Decision Properties of Regular Languages

Decision Properties of Regular Languages

- 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.
- Some Decision Properties:
 - Is language L empty?
 - Is language L finite?

Decision Properties of Regular Languages

language is empty or not

Theorem: There exist algorithms for determining whether a regular language is empty or not.

Proof:

- Represent the language with a DFA.
- If there is a path from the start state to some final state, the language is not empty.

Decision Properties of Regular Languages

language is finite or infinite

Theorem: There exist algorithms for determining whether a regular language is finite or infinite.

Proof:

- Represent the language with a DFA.
- Find all states that form a cycle.
- If any of these cycle states are on path from the start state to a final state, then the language is infinite.

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.

More Decision Problems


- To decide if $L_1 \subseteq L_2$, check if $L_1 - L_2 = \phi$
- To decide if $\epsilon \in L$, check if $q_0 \in F$
- To decide if L contains a string w such that $w = w^R$
 - Let M be the DFA for L .
 - Construct M^R .
 - Construct $M \cap M^R$ using the cross-product construction
 - Check if $L(M \cap M^R) \neq \phi$.

The Pumping Lemma for Regular Languages

The Pumping Lemma for Regular Languages

- There are languages which are NOT regular.
- Every regular language satisfies the **pumping lemma**.
- **A non-regular language can be shown that it is NOT regular using the pumping lemma.**
- $L_{01} = \{0^n1^n \mid n \geq 1\}$ is not regular.
 - We can use the pumping lemma to show that this language is not regular.

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. $y \neq \varepsilon$, ie. $|y| > 0$
 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 .

Number of states of
DFA for L

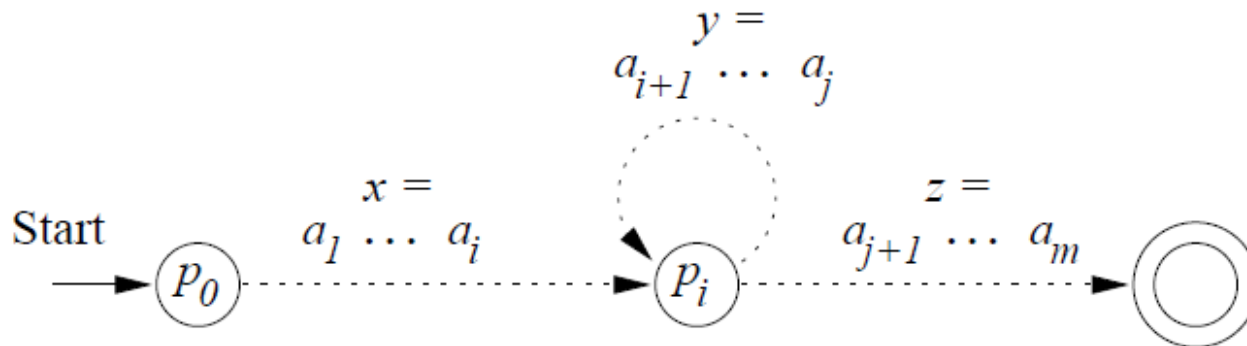
The Pumping Lemma - Proof

Proof:

- Suppose L is regular
- Then L is recognized by some DFA A with n states, and $L = L(A)$.
- Let a string $w = a_1 a_2 \dots a_m \in L$, where $m > n$
- Let $p_i = \hat{\delta}(q_0, a_1 a_2 \dots a_i)$
- Then, there exists j such that $i < j$ and $p_i = p_j$
- Now we have $w = xyz$ where
 1. $x = a_1 a_2 \dots a_i$
 2. $y = a_{i+1} a_{i+2} \dots a_j$
 3. $z = a_{j+1} a_{j+2} \dots a_m$

The Pumping Lemma – Proof (cont.)

- That is, x takes us to p_i , once; y takes us from p_i back to p_i . (since p_i is also p_j), and z is the balance of w .
- So we have the following figure, and every string longer than the number of states must cause a state to repeat.



- Since y can repeat 0 or more times

→ $xy^kz \in L$ for any $k \geq 0$.

Q.E.D.

Applications of The Pumping Lemma

- 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.

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 $|w| > n$.**
 - **Tricky Part 1: You should find a string w so that you can create a contradiction in step 5. 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, $w = xyz$, $|xy| \leq n$, $y \neq \epsilon$ and $xy^kz \in L_{01}$.

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

- Since $y \neq \epsilon$ 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, $xy^{p-m}z \in L_{pr}$ 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: 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

Applications of The Pumping Lemma – Example 5

Example 5: 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

Properties of Regular Languages – Summary

Minimizing Deterministic Finite Automata:

- We can partition states of any DFA into groups of mutually indistinguishable states.
- Members of two different groups are always distinguishable.
- If we replace each group by a single state we get an equivalent DFA that has as few states as any DFA for the same language.

Testing Distinguishability of States:

- Two states of a DFA are distinguishable if there is an input string that takes exactly one of the two states to an accepting state.
- By starting with only the fact that pairs consisting of one accepting and one non-accepting state are distinguishable and trying to discover additional pairs of distinguishable states by finding pairs whose successors on one input symbol are distinguishable we can discover all pairs of distinguishable states.

Properties of Regular Languages – Summary

Closure Properties of Regular Languages:

- There are many operations that preserve the property of being a regular language.
- Among these are union, concatenation, closure, intersection, complement, difference, reversal, homomorphism.

Decision Properties of Regular Languages:

- Testing emptiness of regular languages
- Testing whether a regular language is finite or not.

The Pumping Lemma for Regular Languages:

- If a language is regular then every sufficiently long string in the language has a nonempty substring that can be pumped that is repeated any number of times while the resulting strings are also in the language.
- This fact can be used to prove that many different languages are not regular.