Properties of Regular Languages

Properties of Regular Languages

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.

Closure Properties:

- One important kind of fact about the regular languages is called a ***closure property***.
- These properties let us build recognizers for languages that are constructed from other languages by certain operations.
- As an example, the intersection of two regular languages is also regular. Thus, given automata that recognize two different regular languages, we can construct mechanically an automaton that recognizes exactly the intersection of these two languages.

Decision Properties:

- Are two automata equivalent? How can we test it?
- ***Minimization of DFAs*** ➔
  1. We can use to test equivalency of two DFAs,
  2. Minimization of DFAs saves money

Proving Languages not to be Regular

- Not every language is a regular language.
- A powerful technique, known as the ***pumping lemma***, can be used to show a certain language NOT to be regular.
- $L_{01} = \{0^n1^n \mid n \geq 1\}$ is not regular.
  - We can use the pumping lemma to show that this language is not regular.

The Pumping Lemma Informally

- Suppose $L_{01} = \{0^n 1^n \mid n \geq 1\}$ were regular.
- Then it would be recognized by some DFA A with k states.
- Let A read 0^k as input:

ϵ	p_0
0	p_1
00	p_2
$\ldots$	$\ldots$
0^k	p_k

Since there are only k states

$\Rightarrow \exists i < j : p_i = p_j$ Call this state q.

- Now, we can fool A:
  - If $\widehat{\delta}(q, 1^i) \in F$ the machine will foolishly accept $0^j 1^i$
  - If $\widehat{\delta}(q, 1^i) \notin F$ the machine will foolishly rejects $0^i 1^i$

- Therefore L_{01} cannot be regular.

The Pumping Lemma for Regular Languages

Theorem 4.1. 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. $y \neq \varepsilon$
    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.

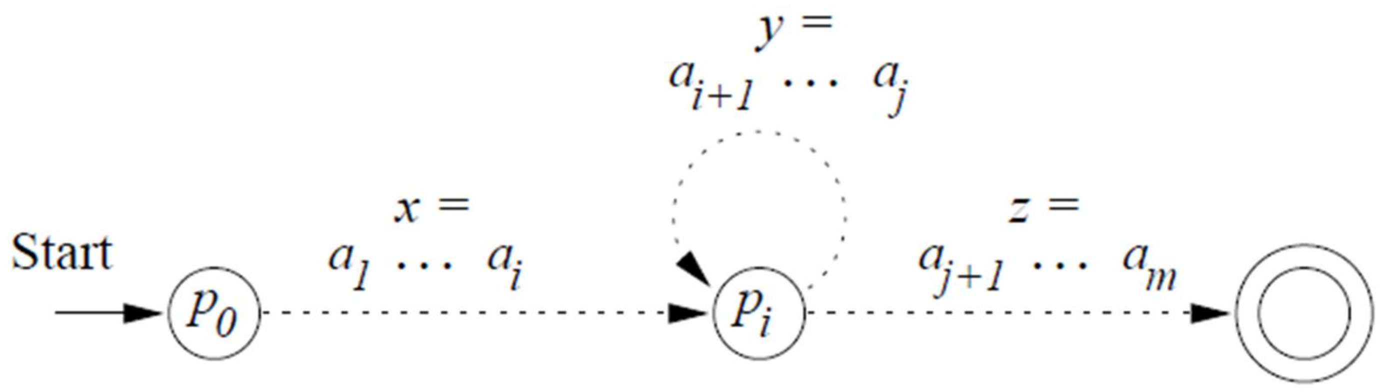
The Pumping Lemma - Proof

Proof:

- Suppose L is regular
- Then L is recognized by some DFA A with n states, and L=L(A).
- Let a string $w=a_1a_2...a_m \in L$, where $m>n$
- Let $p_i = \hat{\delta}(q_0, a_1a_2...a_i)$
- Then, there exists j such that $i<j$ and $p_i = p_j$
- Now we have w=xyz where
  1. $x = a_1a_2...a_i$
  2. $y = a_{i+1}a_{i+2}...a_j$
  3. $z = a_{j+1}a_{j+2}...a_m$

The Pumping Lemma - Proof

- 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 $k>0$. Q.E.D.

Applications of the Pumping Lemma – Example 1

- We use the pumping lemma to prove that the language is **not** regular.

Example1: Let us show that the language L_{eq} consisting of all strings with an equal number of 0's and l's is not a regular language.

- Suppose L_{eq} were regular
- Then $0^n1^n \in L_{eq}$
- By the pumping lemma $w=xyz$, $|xy| \leq n$, $y \neq \varepsilon$ and $xy^kz \in L_{eq}$

$$w = \underbrace{000\cdots}_{x}\underbrace{\cdots 0}_{y}\underbrace{0111\cdots 11}_{z}$$

- If y repeats 0 times, xy must be in L_{eq}
- Since $y \neq \varepsilon$, xy has fewer 0’s than 1’s
- So, there is a contradiction with our assumption (L_{eq} is regular)
- Proof by contradiction, we prove that L_{eq} is NOT regular

Applications of the Pumping Lemma – Example 2

Example2: Let us show that the language L_{pr} consisting of all strings of l's whose length is a prime is not a regular language.

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

$$w = \overbrace{\underbrace{111\cdots\cdots}_{x}\underbrace{\cdots 1}_{\substack{y \\ |y|=m}}\underbrace{1111\cdots 11}_{z}}^{p}$$

- Now, $xy^{p-m}z \in L_{pr}$
- $|xy^{p-m}z| = |xz| + (p-m)|y| = (p-m)+(p-m)m = (1+m)(p-m)$
- But, $(1+m)(p-m)$ is not prime unless one of the factors is 1.
  - $y \neq \varepsilon$ ➔ $(1+m) > 1$
  - $m=|y| \leq |xy| \leq n$ and $p \geq n+2$ ➔ $(p-m) \geq (n+2)-n \geq 2$
- So, contradiction ➔ Proof by contradiction, L_{eq} is NOT regular

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 also indicate that how regular languages (their DFAs) can be created from other regular languages using certain operations.

Principal Closure Properties of Regular Languages

1. The union of two regular languages is regular.
2. The intersection of two regular languages is regular.
3. The complement of a regular language is regular.
4. The difference of two regular languages is regular.
5. The reversal of a regular language is regular.
6. The closure (star) of a regular language is regular.
7. The concatenation of regular languages 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.

The union of two regular languages is regular – Closure Under Union

Theorem 4.4. For any regular languages L and M, L∪M is regular.

Proof.

- Since L and M are regular, they have regular expressions; say L = L(R) and M = L(S).
- Then L U M = L(R + S) by the definition of the + operator for regular expressions. □

The closure of a regular language is regular, and the concatenation of regular languages is regular

Theorem. (closure under concetanation)

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

Proof.

- Let L=L(R) and M=L(S), then LM=L(RS)

Theorem. (closure under star)

- If L is a regular language, then so is L*.

Proof.

- Let L=L(R) , then L*=L(R*)

The complement of a regular language is regular - Closure Under Complementation

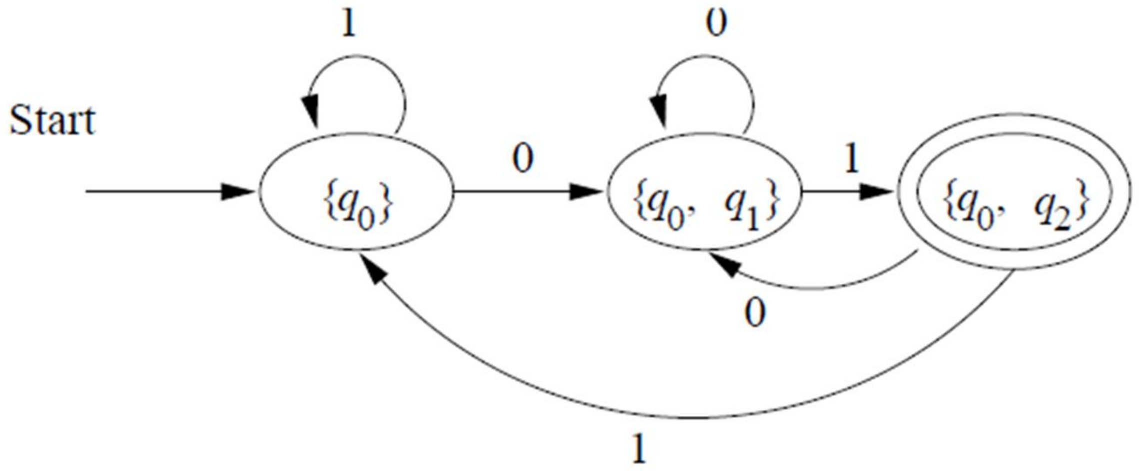
Theorem 4.5. If L is a regular language over alphabet Σ, then $\overline{L} = \Sigma^* - L$ is also a regular language.

Proof.

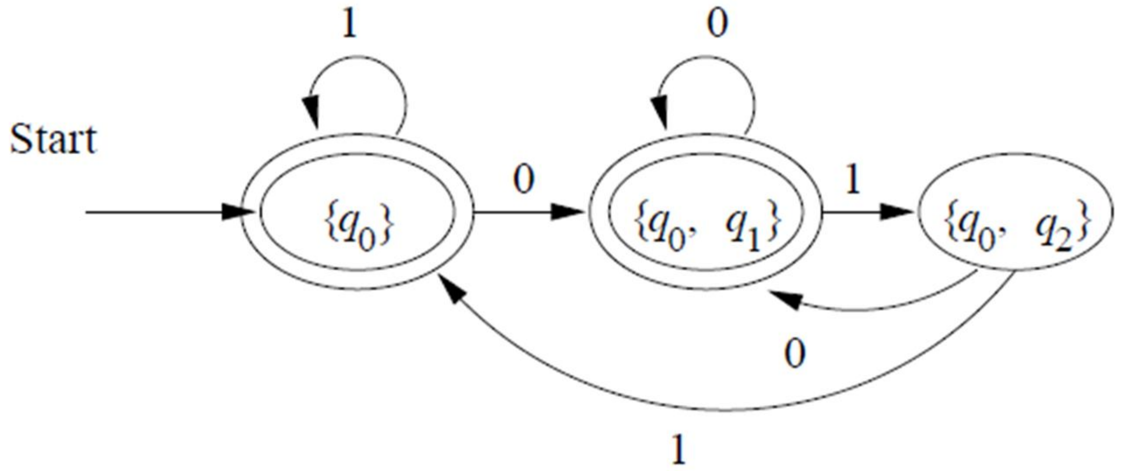
- Let L = L(A) for some DFA A = (Q, Σ, δ, q_0, F)
- Then $\overline{L}$ = L(B), where B is the DFA (Q, Σ, δ, q_0, Q-F)
- That is., B is exactly like A, but the accepting states of A have become non-accepting states of B, and vice versa.
- Then w is in L(B) if and only if $\hat{\delta}(q_0,w)$ is in Q-F,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

Closure Under Complementation - Example

L=L(A) : strings of 0's and l's that end in 01;
in regular-expression terms,
L(A) = L((0+l)*01).



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



Closure Under Complementation – Pumping Lemma Example

Problem: Show that the language M consisting of those strings of 0's and l's that have an unequal number of 0’s and 1’s is NOT regular.

- It would be hard to use the pumping lemma to show that M is not regular.
- However, M is still NOT regular.
- The reason is that M = $\overline{\text{L}}$.
- Since the complement of the complement is the set we started with, it also follows that L = $\overline{\text{M}}$.
- If M is regular, then by Theorem 4.5, L is regular.
- But we know L (all of strings of equal number of 0’s and 1’s) is not regular; so we have a proof by contradiction that M is not regular. □

The intersection of two regular languages is regular Closure Under Intersection

Theorem 4.8. If L and M are regular languages, then so is L$\cap$M.

Proof 1. (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. □

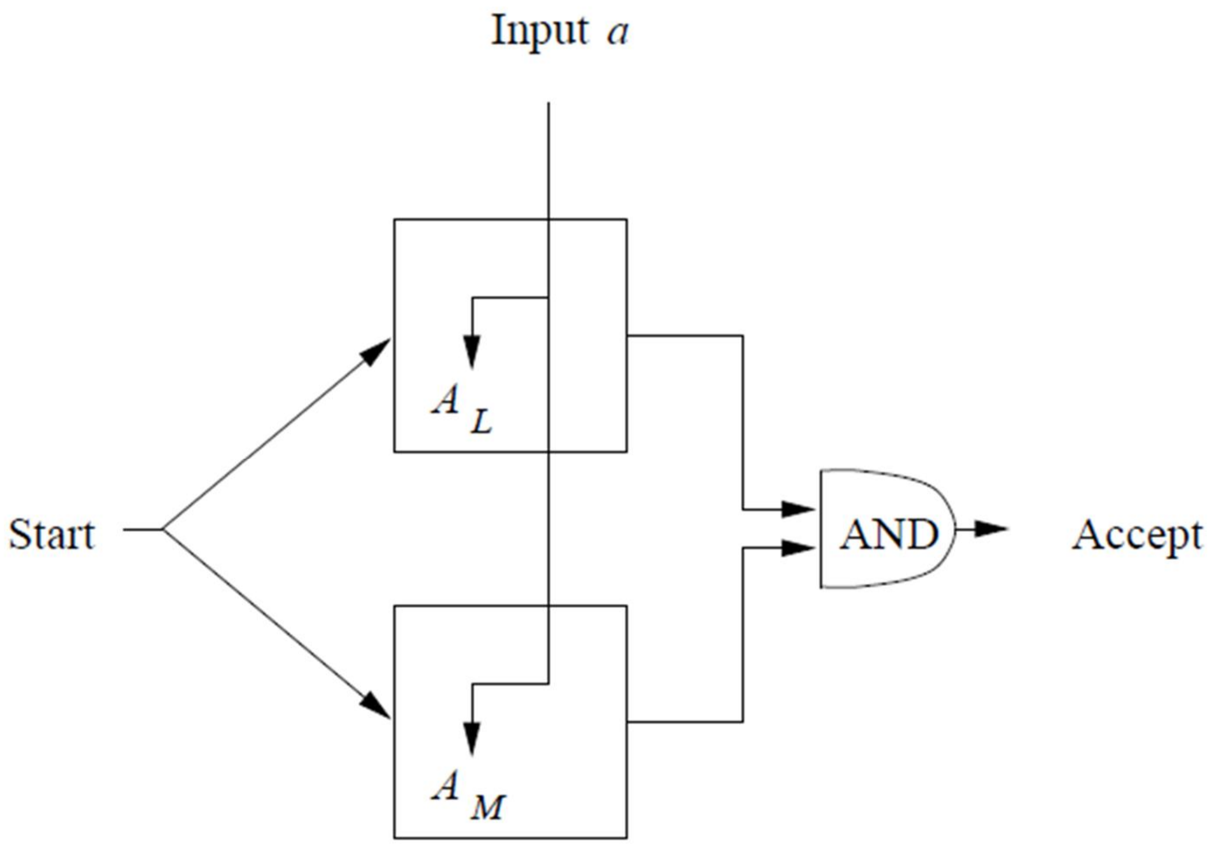
Closure Under Intersection – Proof2

Theorem 4.8. If L and M are regular languages, then so is $L \cap M$.

Proof 2. (DFA Construction Proof)

- Let L be the language of $A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$, and M be the language of $A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$,
  - For simplicity, we will assume that A_L and A_M are DFAs. For NFAs, we can have a similar construction.
- We shall 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 a, and A_M goes from state q to state t on reading a, then $A_{L \cap M}$ will go from state (p,q) to state (s,t) on reading a.

Closure Under Intersection – Proof2



An automaton simulating two other automata and accepting if and only if both accept.

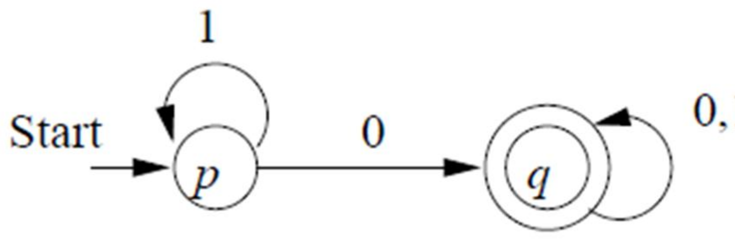
Closure Under Intersection – Proof2

- Formally $A_{L\cap M} = (Q_L x Q_M, \Sigma, \delta_{L\cap M}, (q_L,q_M), F_L x F_M)$ where

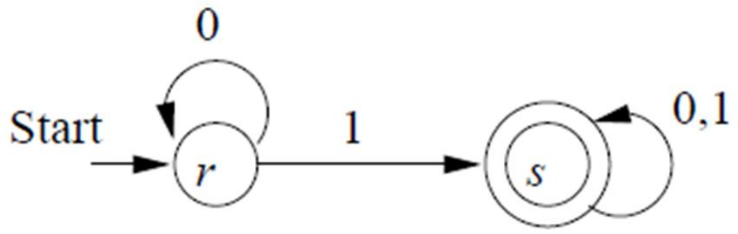
  $\delta_{L\cap M}((p,q),a) = (\delta_L(p,a), \delta_M(q,a))$
- By induction on |w|, it can be shown that

  $\hat{\delta}_{L\cap M}((q_L,q_M),w) = (\hat{\delta}_L(q_L,w), \hat{\delta}_M(q_M,w))$
- To see why $L(A_{L\cap M})=L(A_L)\cap L(A_M)$, first we can observe that an easy 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 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 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 $L(A_M)$. □

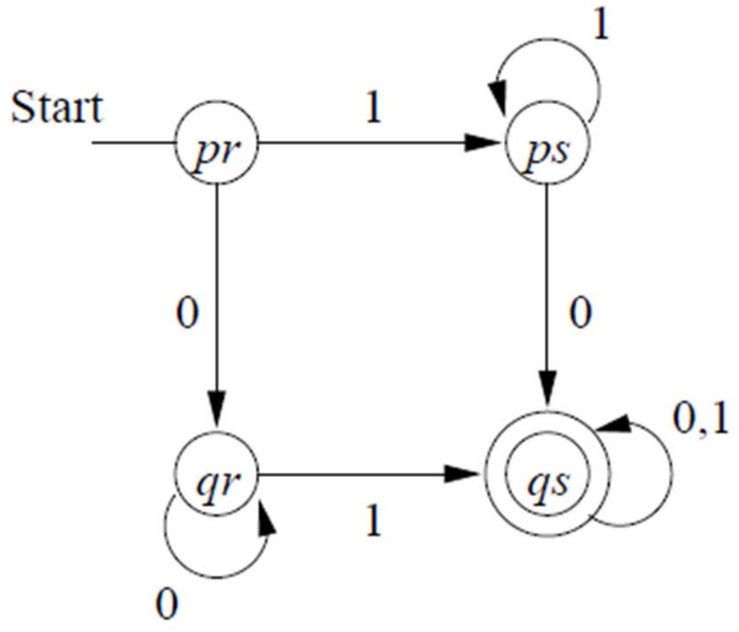
Closure Under Intersection – Product ConstructionExample



(a)



(b)



(c)

- (c) = (a)x(b), ie. $L_c = L_a \cap L_b$

The difference of two regular languages is regular - Closure Under Difference

Theorem 4.10. If L and M are regular languages, then so is L - M.

Proof.

- Observe that L-M = $L \cap \overline{M}$.
- By Theorem 4.5. $\overline{M}$ is regular, and
- By Theorem 4.8. $L \cap \overline{M}$ is regular.
- Therefore L-M is regular. □

The reversal of a regular language is regular - Reversal

- The reversal of a string $a_1a_2 \ldots a_n$ is the string written backwards, that is, $a_na_{n-1} \ldots a_1$.
- 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 - Theorem

Theorem 4.11. If L is a regular language, so is L^R.

Proof 1. (Automaton Creation)

- Let L be recognized by an Finite Automaton (FA) A.
- Turn A into an 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.

Reversal – Theorem – Proof2

Theorem 4.11. If L is a regular language, so is L^R.

Proof 2. (Using Regular Expressions)

- Assume L is defined by regular expression E.
- The proof is a structural induction on the size of E.
- We 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.

Reversal – Theorem – Proof2

BASIS:

- If E is **ε**, **ϕ**, or **a**, for some symbol a, then E^R is the same as E.
- We know $\{\varepsilon\}^R = \{\varepsilon\}$, $\phi^R = \phi$, 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, $L(F^R+G^R) = (L(F+G))^R$

Reversal – Theorem – Proof2

Case 2. $E = F\,G$ ➔ $E^R = G^R\,F^R$

- 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, $L(G^RF^R) = (L(FG))^R$

Reversal – Theorem – Proof2

Case 3. E = F* ➔ $E^R = (F^R)^*$

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

Reversal – Example

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

$$M^R = (L((0+1)0^*))^R$$

$$= L(((0+1)0^*))^R)$$

$$= L((0^*)^R(0+1)^R)$$

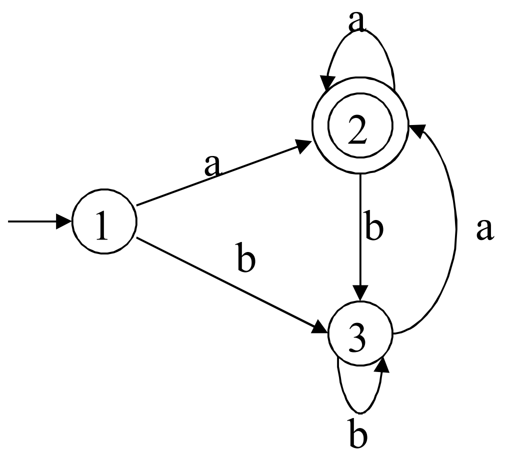
$$= L((0^R)^* (0^R+1^R))$$

$$= L(0^*(0+1))$$

Minimizing Number of States of a DFA

- partition the set of states into two groups:
  - G_1 : set of accepting states
  - G_2 : set of non-accepting states
- For each new group G
  - partition G into subgroups such that states s_1 and s_2 are in the same group iff for all input symbols a, states s_1 and s_2 have transitions to states in the same group.
- Start state of the minimized DFA is the group containing the start state of the original DFA.
- Accepting states of the minimized DFA are the groups containing the accepting states of the original DFA.

Minimizing DFA - Example

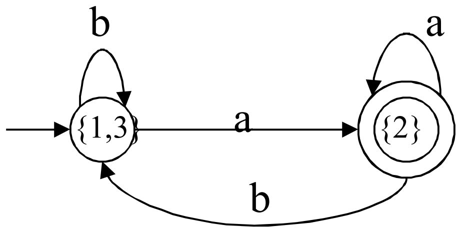


$G_1 = \{2\}$
$G_2 = \{1,3\}$

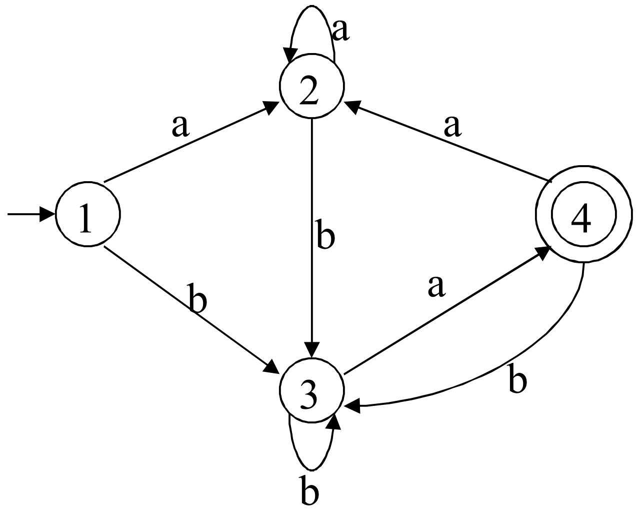
G_2 cannot be partitioned because

$\delta(1,a)=2$	$\delta(1,b)=3$
$\delta(3,a)=2$	$\delta(2,b)=3$

So, the minimized DFA (with minimum states)



Minimizing DFA – Another Example



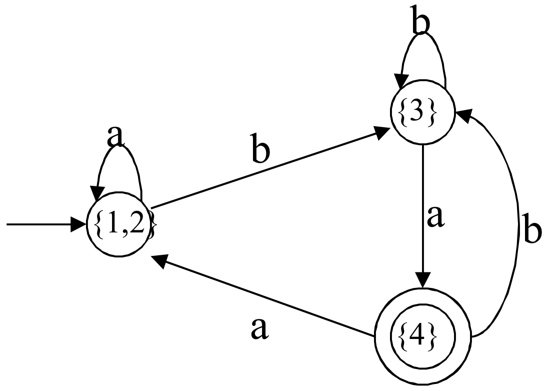
Groups: {1,2,3} {4}

$$\begin{tikzcd} & \{1,2,3\} \arrow[dl] \arrow[dr] & \\ \{1,2\} & & \{3\} \end{tikzcd}$$

no more partitioning

a	b
1->2	1->3
2->2	2->3
3->4	3->3

So, the minimized DFA



Equivalence and Minimization of Automata

Equivalence of States:

- Let A = (Q,Σ,δ,q_0,F) be a DFA, and $\{p,q\}\subseteq Q$, we define

$$p \equiv q \;\Leftrightarrow\; \forall w \in \Sigma^* \;:\; \hat{\delta}(p,w) \in F \text{ iff } \hat{\delta}(q,w) \in F$$

- If $p \equiv q$ we say that p and q are ***equivalent***.
- If $p \neq q$ we say that p and q are ***distinguishable***.

- In other words, p and q are distinguishable iff

$$\exists w : \hat{\delta}(p,w) \in F \text{ and } \hat{\delta}(q,w) \notin F, \text{ or vice versa}$$

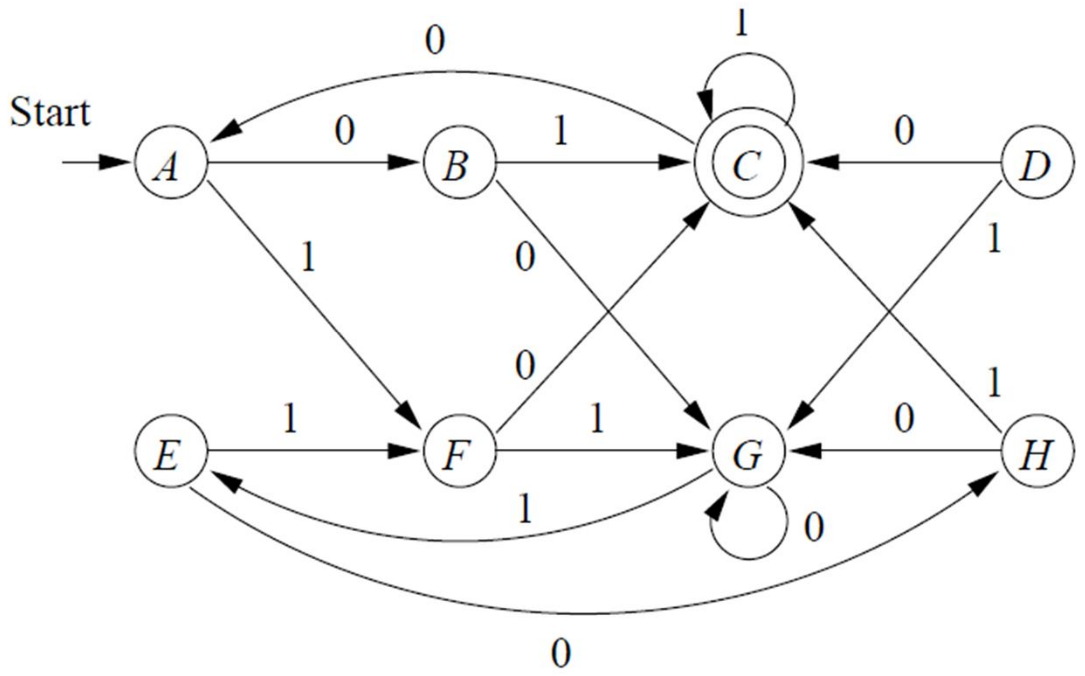
Finding Distinguishable States – Table Filling Algorithm

- We can compute distinguishable pairs of states with the following inductive table filling algorithm.

Basis: If $p \in F$ and $q \notin F$, then $p \not\equiv q$.

Induction: If $\exists a \in \Sigma : \delta(p, a) \not\equiv \delta(q, a)$,
then $p \not\equiv q$.

Table Filling Algorithm - Example



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's

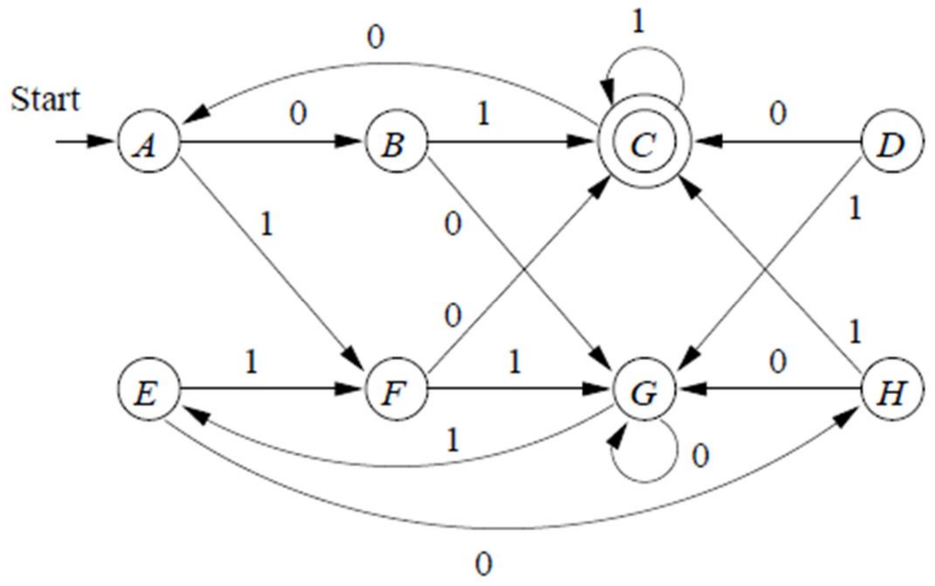
- We can also use table filling algorithm to minimize a DFA by merging all equivalent states. That is, we replace a state p with its *equivalance class* found by the table filling algorithm.

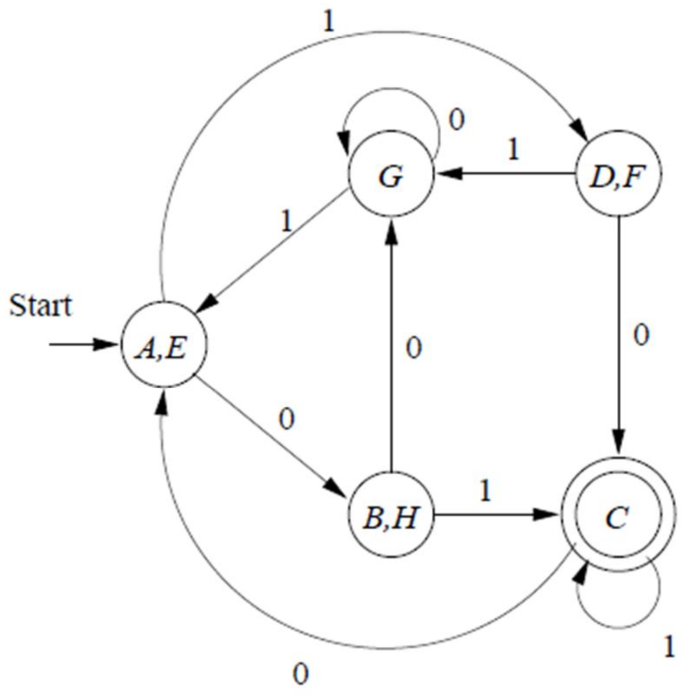
B	*x*						
C	*x*	*x*					
D	*x*	*x*	*x*				
E		*x*	*x*	*x*			
F	*x*	*x*	*x*		*x*		
G	*x*	*x*	*x*	*x*	*x*	*x*	
H	*x*		*x*	*x*	*x*	*x*	*x*
	A	*B*	*C*	*D*	*E*	*F*	*G*

Equivalance Classes:

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

Minimization of DFA's - Example





Equivalance Classes:
{ {A,E}, {B,H}, {C}, {D,F}, {G} }

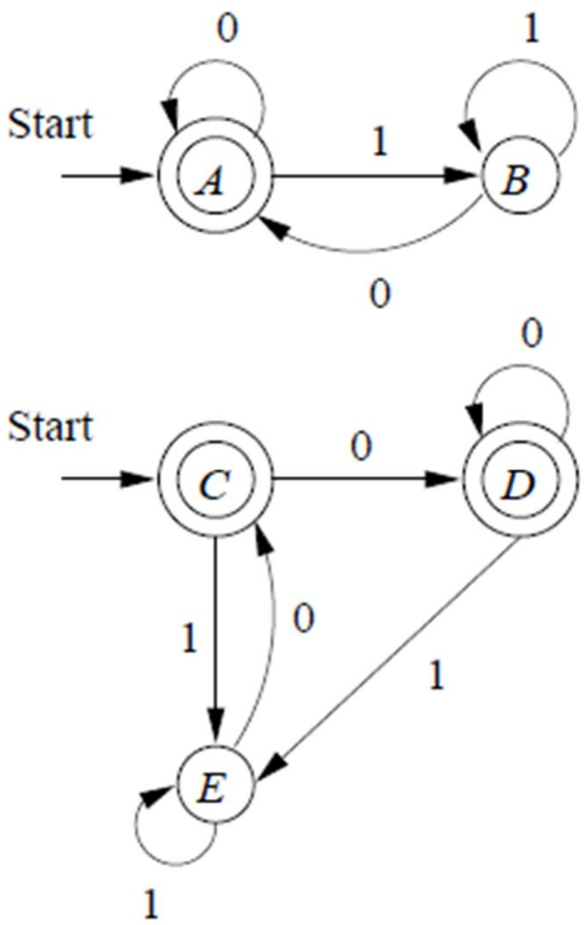
Testing Equivalence of Regular Languages with Table Filling Algorithm

- Let L and M be regular languages, to test if L = M
  1. Convert both L and M to DFAs
  2. Imagine the DFA that is the union of the two DFA's (never mind there are two start states)
  3. If table filling algorithm says that the two start states are distinguishable, then L ≠ M, otherwise L = M.

Another Way To Test Equiavalance:

- Minimize DFAs,
- Check whether they are isomorphic

Testing Equivalence of Regular Languages with Table Filling Algorithm - Example



B	x			
C		x		
D		x		
E	x		x	x
	A	B	C	D

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