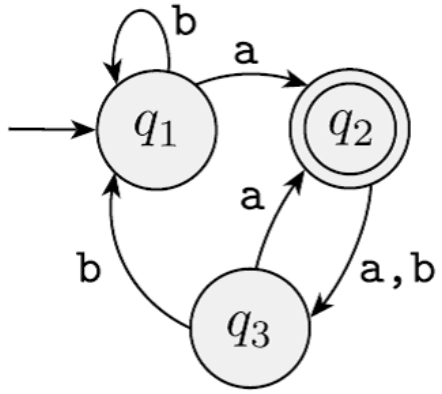
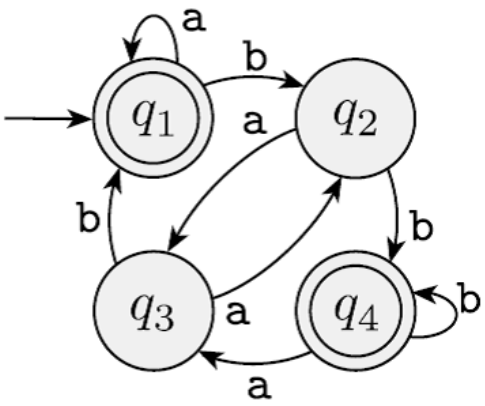
Problem

The following are the state diagrams of two DFAs,M_1 andM_2. Answer the following questions about each of these machines.



M_1



M_2

a. What is the start state?

b. What is the set of accept states?

c. What sequence of states does the machine go through on input aabb?

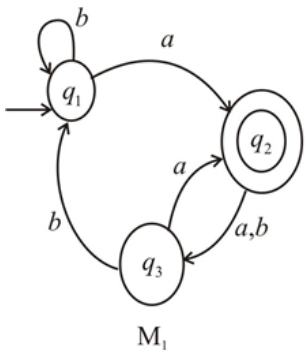
d. Does the machine accept the string aabb?

e. Does the machine accept the string "?

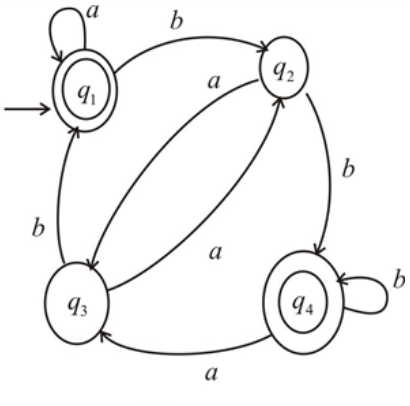
Step-by-step solution

Step 1 of 7

State diagrams for two DFAs, M_1 and M_2 are



M_1



M_2

Comment

Step 2 of 7

(a) Start state is indicated by the arrow pointing towards it.

• For the DFA (Deterministic Finite Automata) M_1, an arrow is pointing towards the state q_1. So q_1 is the start state.

• For the DFA M_2, arrow is pointing towards the state q_1. So q_1 is the start state.

Comment

Step 3 of 7

(b) The accept state is the one which is identified by a double circle.

• In state diagram of M_1, the state q_2 has double circle. So $\{q_2\}$ is the accept state.

• In the sate diagram of M_2, the states q_1 and q_4 have double circles. So the set $\{q_1, q_4\}$ are the accept states.

Comment

Step 4 of 7

(c)

• When we given the string *aabb* as input to the machine M_1 then the following transitions will takes place

(1) Start in state q_1

(2) Read *a*, follow transition from q_1 to q_2

(3) Read *a,* follow transition from q_2 to q_3.

(4) Read *b*, follow transition from q_3 to q_1.

(5) Read *b*, follow transition from q_1 to q_1

From (1), (2), (3), (4), (5)

So the machine M_1 will go through following sequence of states on input *aabb*. q_1, q_2, q_3, q_1, q_1

• When we give the input *aabb* to machine M_2 then the following transitions will takes place

(1) Start in state q_1

(2) Read *a*, follow transition from q_1 to q_1

(3) Read *b*, follow transition from q_1 to q_2

(5) Read *b*, follow transition from q_2 to q_4

From (1), (2), (3), (4), (5)

The machine M_2 will go through the following sequence of states on input *aabb*

q_1, q_1, q_1, q_2, q_4

Comment

Step 5 of 7

(d)

• Machine M_1 will go through following sequence of sates on input *aabb*

1. Start in state q_1

2. Read *a*, follow transition from q_1 to q_2

3. Read *a*, follow transition from q_2 to q_3

4. Read *b*, follow transition from q_3 to q_1

5. Read *b*, follow transition from q_1 to q_1

On reading the input *aabb*, M_1 finally entered into state q_1, which is not an accept state.

So M_1 reject the input *aabb*.

• Machine M_2 will go through the following sequence of sates on input *aabb*

1. Start in state q_1

2. Read *a*, follow transition from q_1 to q_1

3. Read *a*, follow transition from q_1 to q_1

4. Read *b*, follow transition from q_1 to q_2

5. Read *b*, follow transition from q_2 to q_4

On reading the input *aabb*, M_2 finally entered into state q_4.

q_4 accept state of M_2.

Hence M_2 accept the input *aabb*.

Comment

Step 6 of 7

(e)

• On giving the input $\in$(empty string) to machine M_1, M_1 always in the start state q_1.

But $\{q_1\}$ in not the accept sate of M_1.

Thus M_1 does not accept the string $\in$

•

Comment

Step 7 of 7

On giving the input $\in$(empty string) to the machine M_2, M_2 always in the start state q_1.

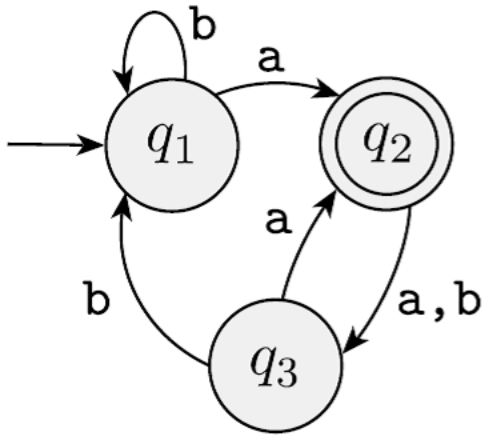
But $\{q_1\}$ is also accept state of M_2.

Thus M_2 accept the string $\in$.

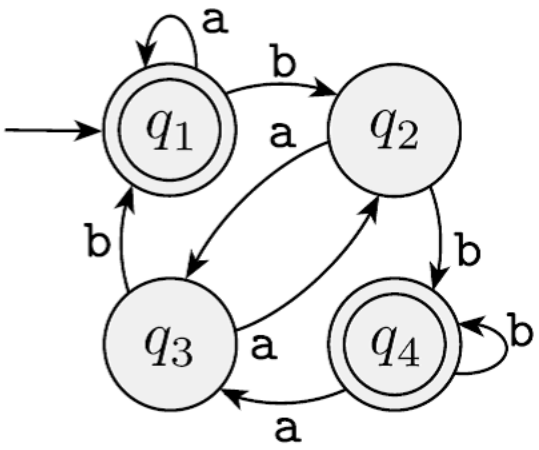
Comment

Problem

Give the formal description of the machinesM1 and M2 pictured in Exercise 1.1.



M_1

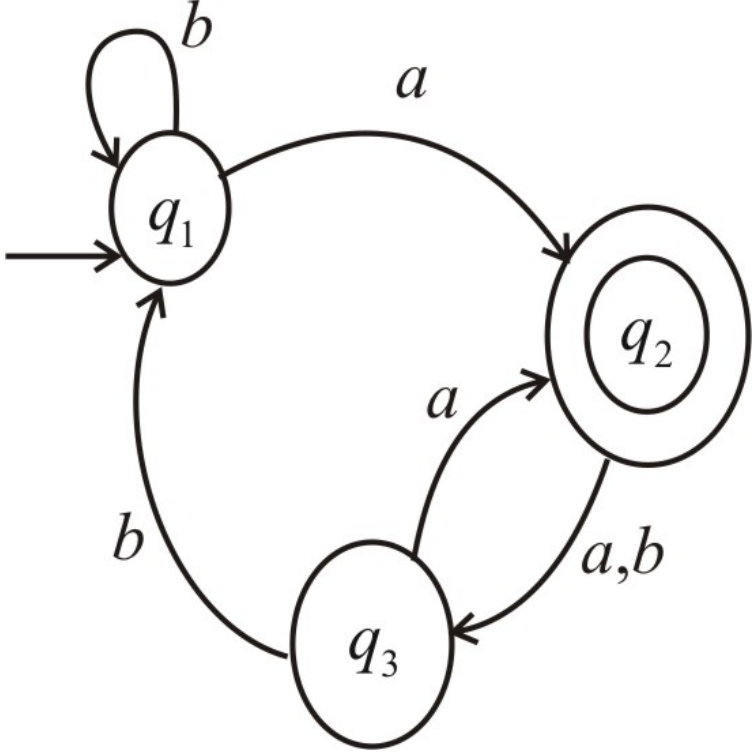


M_2

Step-by-step solution

Step 1 of 4

Given state diagram for the machine M_1 is



Comment

Step 2 of 4

Formal definition of a finite automata is

A finite automaton is a 5-tupel (Q,Σ,δ,q_0,F)

Where

1. Q is a finite set called states
2. Σ is a finite set called alphabet
3. $\delta : Q\times\Sigma \to Q$ is the transition function.
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states.

Now we can describe M_1 formally by writing $M_1 = (Q,\Sigma,\delta,q_1,F)$, where

1. $Q = \{q_1,q_2,q_3\}$
2. $\Sigma = \{a,b\}$
3. δ is described as

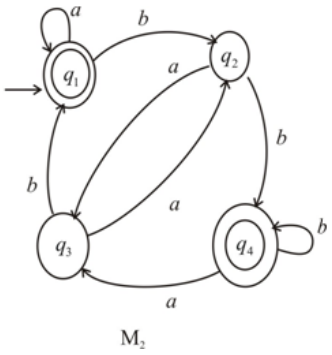
	a	b
q_1	q_2	q_1
q_2	q_3	q_3
q_3	q_2	q_3

4. q_1 is the start state
5. Set of accept states $F = \{q_2\}$.

Comments (3)

Step 3 of 4

Given state diagram for the machine M_2 is



M_2

Comment

Step 4 of 4

Now we can describe M_2 formally by writing

$M_2 = (Q, \Sigma, \delta, q_1, F)$, where

1. Set of states $Q = \{q_1, q_2, q_3, q_4\}$

2. Set of alphabet $\Sigma = \{a, b\}$

3. δ is described as

	a	b
q_1	q_1	q_2
q_2	q_3	q_4
q_3	q_2	q_1
q_4	q_3	q_4

4. q_1 is the start state

5. Set of accept states $F = \{q_1, q_4\}$

Comments (4)

Problem

The formal description of a DFA M is {q1, q2, q3, q4, q5}, {u, d}, $\delta, q_3,$ q3, {q3}), where _ is given by the following table. Give the state diagram of this machine.

	u	d
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_2	q_4
q_4	q_3	q_5
q_5	q_4	q_5

Step-by-step solution

Step 1 of 1

Given formal description of DFA *M* is

$M = (Q, \Sigma, \delta, q_3, F)$

$Q =$ Set of states $= \{q_1, q_2, q_3, q_4, q_5\}$

$\Sigma =$ Set of alphabet $= \{u, d\}$

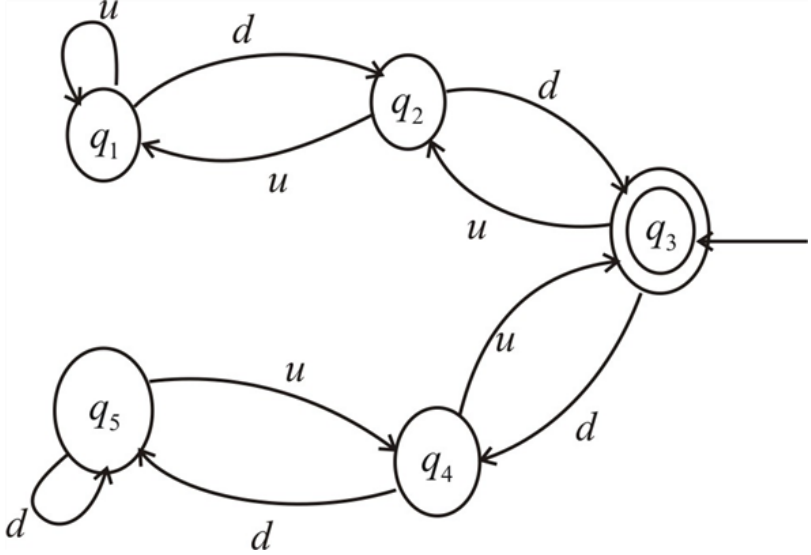
$\delta =$ The transition function is described as

	u	d
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_2	q_4
q_4	q_3	q_4
q_5	q_4	q_5

Start state $= \{q_3\}$ indicated with an arrow

Set of accept states Final state $= \{q_3\}$ indicated by double circle

Now we will construct state diagram by using the above details.



M

So this is the state diagram for the given description of machine *M*.

Comments (2)

Problem

Each of the following languages is the intersection of two simpler languages. In each part, construct DFAs for the simpler languages, then combine them using the construction discussed in footnote 3 (page 46) to give the state diagram of a DFA for the language given. In all parts, ∑ = {a, b}.

a. {wl w has at least three a's and at least two b's}

A**b.** {wl w has exactly two a's and at least two b's}

c. {wl w has an even number of a's and one or two b's}

A**d.** {wl w has an even number of a's and each a is followed by at least one b}

e. {wl w starts with an a and has at most one b}

f. {wl w has an odd number of a's and ends with a b}

g. {wl w has even length and an odd number of a's}

Step-by-step solution

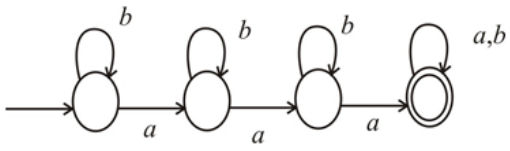
Step 1 of 18

a.

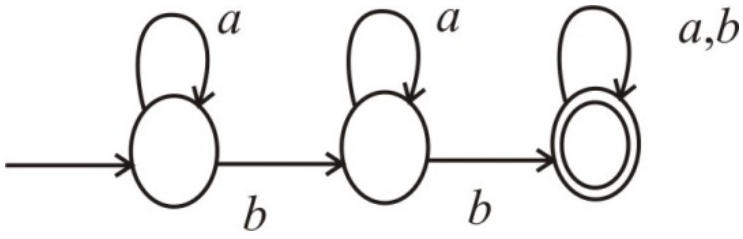
Consider the Language $L = \{w \mid w \text{ has at least three } a\text{'s and at least two } b\text{'s}\}$. The language *L* is the intersection of two simpler languages L_1 and L_2.

Now $L_1 = \{w \mid w \text{ has at least three } a\text{'s}\}$ and $L_2 = \{w \mid w \text{ has at least two } b\text{'s}\}$. Let *M* be the DFA (Deterministic Finite automata) that recognizes *L*. Let M_1 and M_2 be the DFAs that recognizes L_1 and L_2.

The state diagram of M_1 that recognizes $L_1 = \{w \mid w \text{ has at least three } a\text{'s}\}$ is as follows:



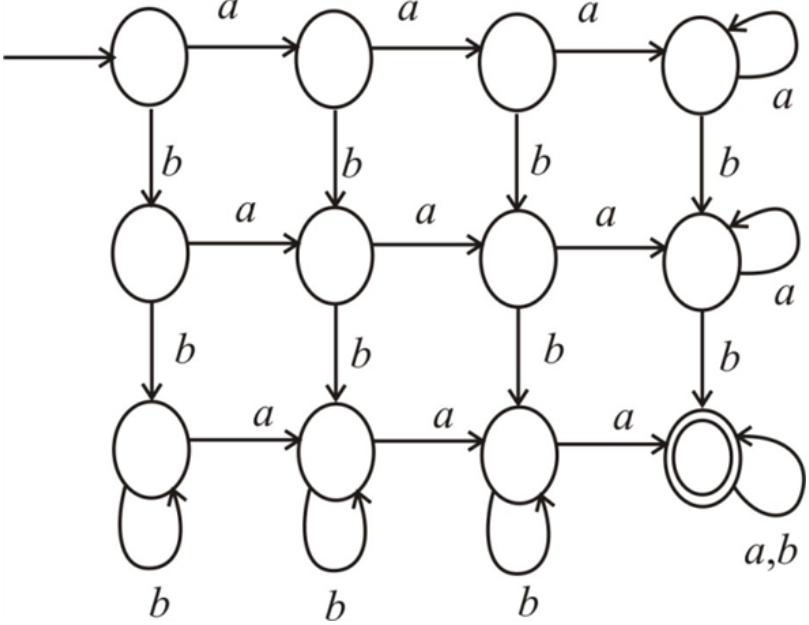
The state diagram of M_2 that recognizes $L_2 = \{w \mid w \text{ has at least two } b\text{'s}\}$ is as follows:



Comments (1)

Step 2 of 18

The machine *M* will accept the input if and only if both M_1 and M_2 accept it. Because, the language *L* is the intersection of the languages L_1 and L_2. Therefore, the state diagram of *M* that recognizes the Language L is as follows:



Comment

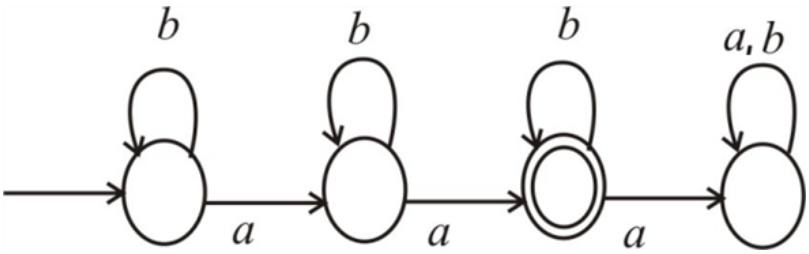
Step 3 of 18

b.

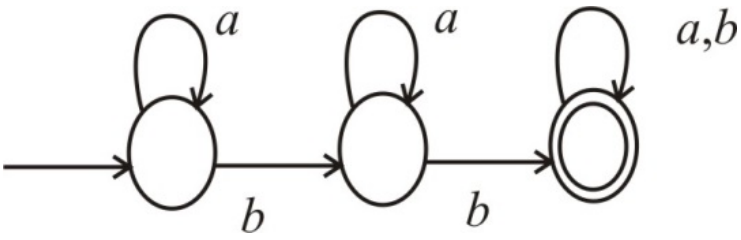
Consider the language $L = \{ w \mid w$ has at exactly two *a*'s and at least two *b*'s$\}$. The language *L* is the intersection of two simpler languages say L_1 and L_2.

Now $L_1 = \{ w \mid w$ has exactly two *a*'s$\}$ and $L_2 = \{ w \mid w$ has at least two *b*'s$\}$. Let *M* be the DFA(Deterministic Finite automata) that recognizes *L* and M_1 and M_2 be the DFAs that recognizes L_1 and L_2.

The state diagram of M_1 that recognizes L_1 is as follows



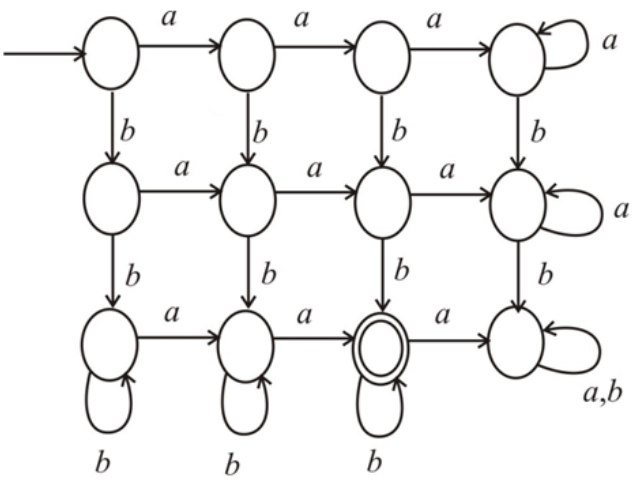
The state diagram of M_2 that recognizes L_2 is as follows



Comments (1)

Step 4 of 18

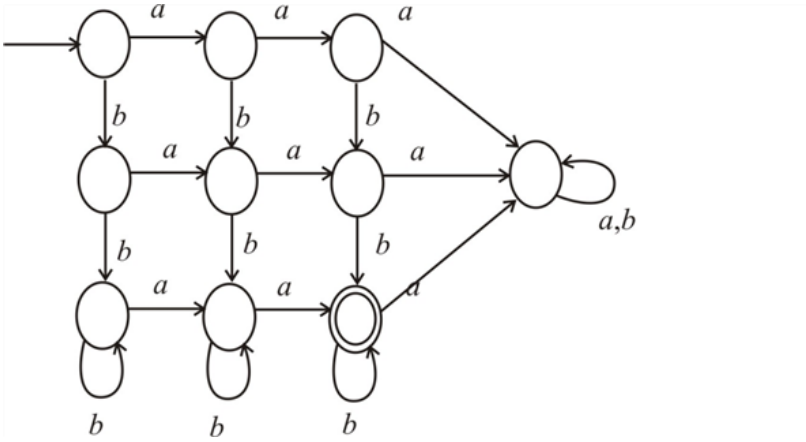
The machine *M* will accept the input if and only if both M_1 and M_2 accept it. Because, the language *L* is the intersection of L_1 and L_2. The state diagram of *M* that recognizes *L* is as follows:



Combine some states to get more simplified form as follows:

Comment

Step 5 of 18



Comments (1)

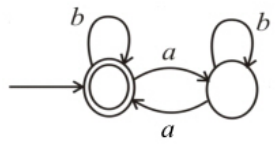
Step 6 of 18

c.

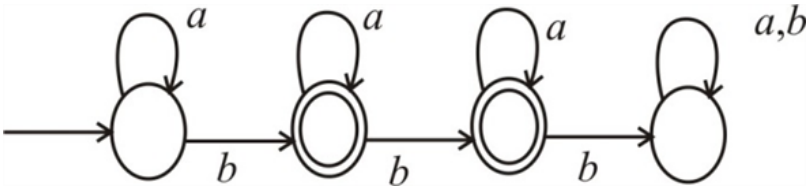
Consider the language $L = \{ w \mid w$ has even number of *a*'s and one or two *b*'s$\}$. The language *L* is the intersection of two simpler languages say L_1 and L_2.

Now $L_1 = \{ w \mid w$ has even number of *a*'s $\}$ and $L_2 = \{ w \mid w$ has one or two *b*'s$\}$. Let *M* be the DFA(Deterministic Finite automata) that recognizes *L* and M_1 and M_2 be the DFAs that recognizes L_1 and L_2.

The state diagram of M_1 that recognizes L_1 is as follows



The state diagram of M_2 that recognizes L_2 is as follows

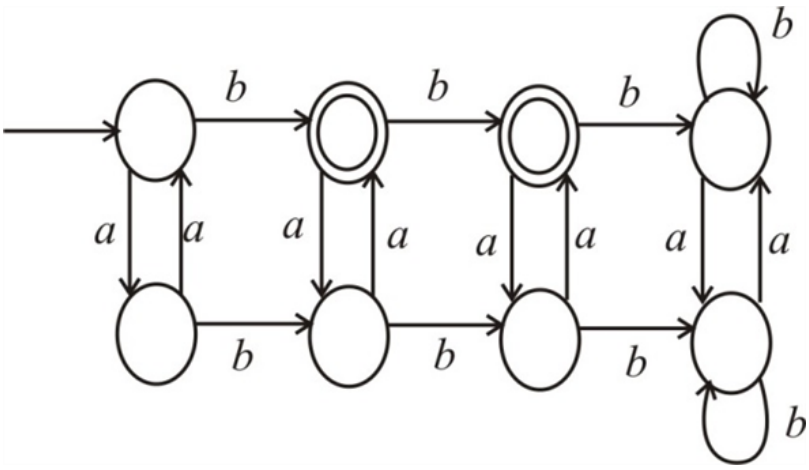


Comment

Step 7 of 18

The machine *M* will accept the input if and only if both M_1 and M_2 accept it. Because language *L* is the intersection of L_1 and L_2.

The state diagram of *M* which recognizes the language *L* is as follows



Comments (3)

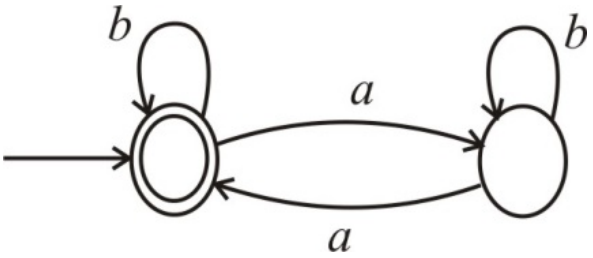
Step 8 of 18

d.

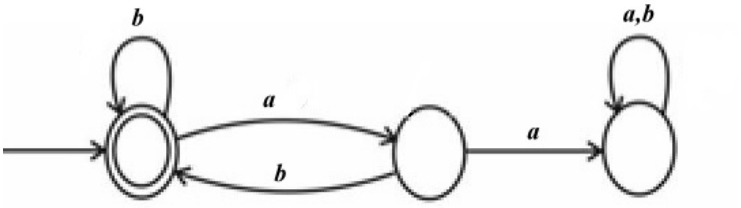
Consider the language $L = \{ w \mid w$ has an even number of *a*'s and each a is a followed by at least on *b*$\}$. The language *L* is the intersection of two simpler languages say L_1 and L_2.

Now $L_1 = \{ w \mid w$ has an even number of *a*'s $\}$ and $L_2 = \{ w \mid w$ has each a is a followed by at least on *b* $\}$. Let *M* be the DFA(Deterministic Finite automata) that recognizes *L* and M_1 and M_2 be the DFAs that recognizes L_1 and L_2.

The state diagram of M_1 that recognizes L_1 is as follows where $L_1 = \{ w \mid w$ has an even number of *a*'s$\}$.



The state diagram of M_2 that recognizes L_2 is as follows where $L_2 = \{ w \mid w$ has each *a* is followed by at least one *b*$\}$.



Comment

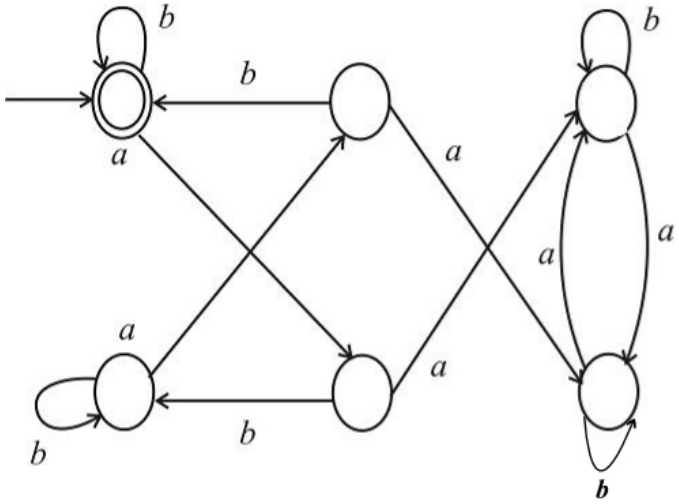
Step 9 of 18

The machine *M* will accept the input if and only if both M_1 and M_2 accept it. Because, the language *L* is the intersection of L_1 and L_2.

The state diagram of *M* that recognizes *L* is as follows:

Comment

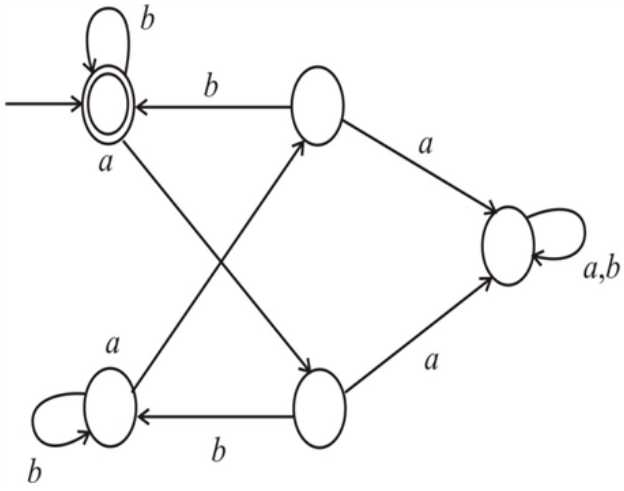
Step 10 of 18



Comments (4)

Step 11 of 18

Combine some states to get more simplified form as follows:



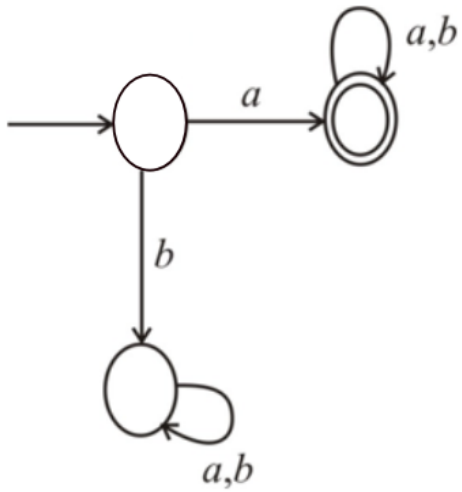
Comments (1)

Step 12 of 18

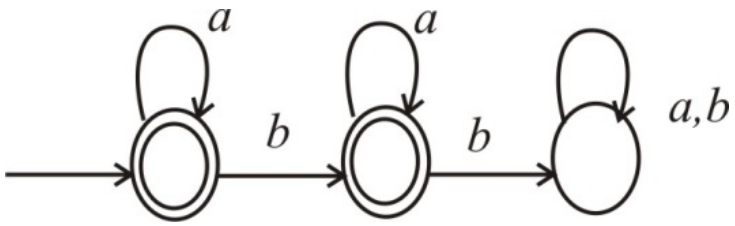
e.

Consider the language $L = \{ w \mid w$ Start with an *a* and has at most one *b*$\}$. The language *L* is the intersection of two simpler languages say L_1 and L_2. Now $L_1 = \{ w \mid w$ starts with *a*$\}$and $L_2 = \{ w \mid w$ has at most one *b*$\}$. Let *M* be the DFA(Deterministic Finite automata) that recognizes *L* and M_1 and M_2 be the DFAs that recognizes L_1 and L_2.

The state diagram of M_1 that recognizes L_1 is as follows:



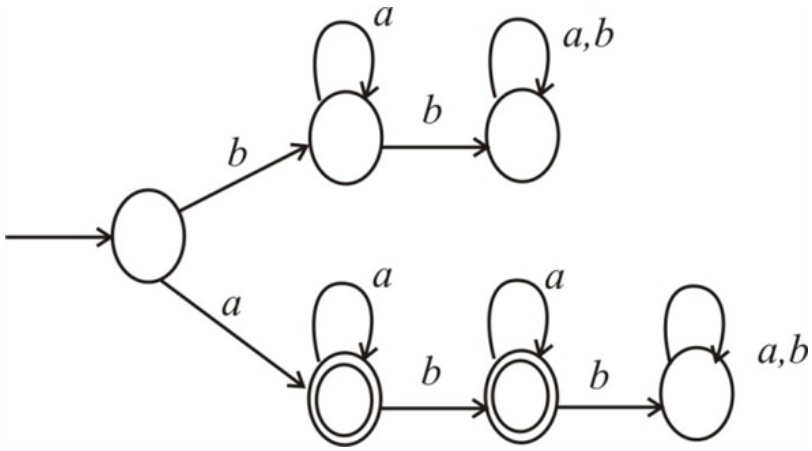
The state diagram of M_2 that recognizes L_2 is as follows.



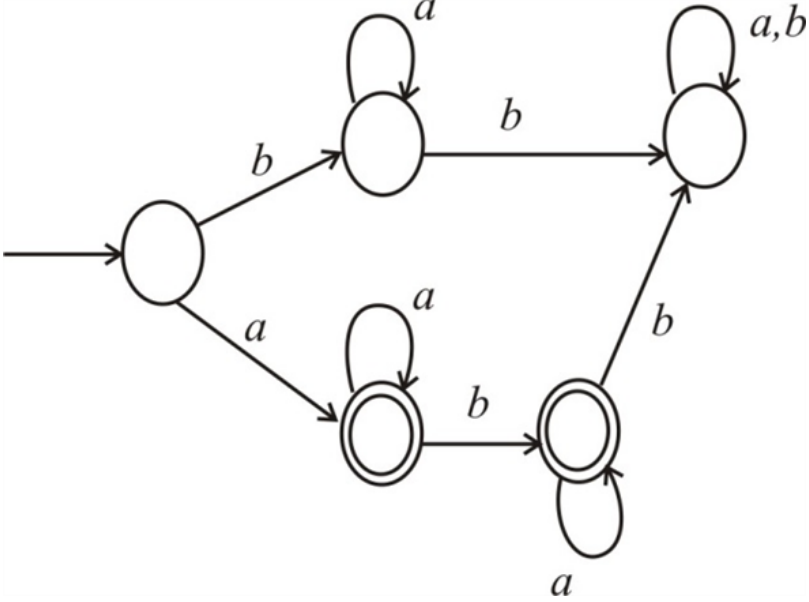
Comment

Step 13 of 18

The machine *M* will accept the input if and only if both M_1 and M_2 accept it. Because, the language *L* is the intersection of L_1 and L_2. The state diagram of *M* that recognizes *L* is as follows.



Combine some states to get more simplified form as follows:



Comments (3)

Step 14 of 18

f.

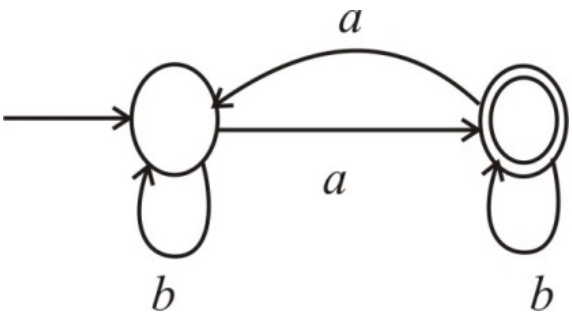
Consider the language $L = \{ w \mid w$ has an odd number of *a*'s and ends with *b*$\}$. The language *L* is the intersection of two simpler languages say L_1 and L_2. Now $L_1 = \{ w \mid w$ has an odd

Comment

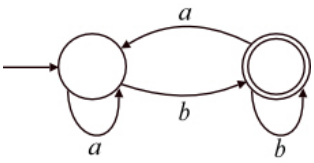
Step 15 of 18

number of *a*'s$\}$ and $L_2 = \{ w \mid w$ ends with a *b*$\}$. Let *M* be the DFA(Deterministic Finite automata) that recognizes *L* and M_1 and M_2 be the DFAs that recognizes L_1 and L_2.

The state diagram of M_1 that recognizes L_1 is as follows:



The language L_2 accepts the strings that ends with a symbol *b*. The state diagram of M_2 that recognizes L_2 is as follows:

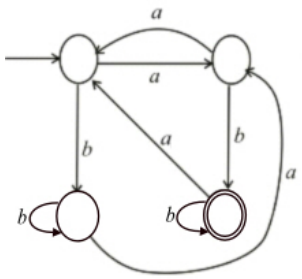


Comment

Step 16 of 18

The machine *M* will accept the input if and only if both M_1 and M_2 accept it. Because, the language *L* is the intersection of L_1 and L_2.

The state diagram of *M* that recognizes *L* is as follows:



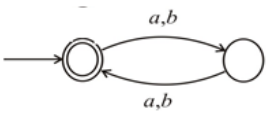
Comments (9)

Step 17 of 18

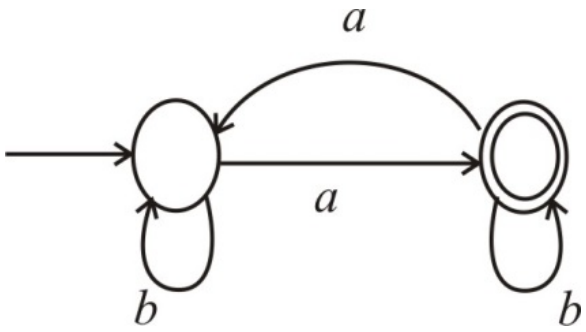
g.

Consider the language $L = \{ w \mid w$ has even length and an odd number of *a*'s $\}$. The language *L* is the intersection of two simpler languages say L_1 and L_2. Now $L_1 = \{ w \mid w$ has even length $\}$ and $L_2 = \{ w \mid w$ has odd number of *a*'s $\}$. Let *M* be the DFA(Deterministic Finite automata) that recognizes *L* and M_1 and M_2 be the DFAs that recognizes L_1 and L_2.

State diagram of M_1 that recognizes L_1 is as follows



State diagram of M_2 that recognizes L_2 is as follows

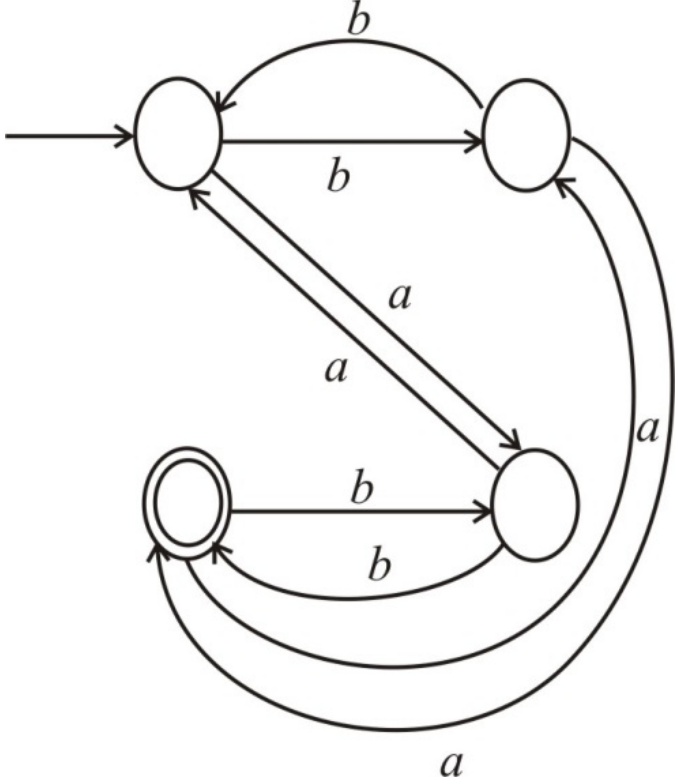


Comment

Step 18 of 18

The machine *M* will accept the input if and only if both M_1 and M_2 accept it. Because, the language *L* is the intersection of L_1 and L_2.

The state diagram of *M* that recognize *L* is as follows:



Comments (2)

Problem

Each of the following languages is the complement of a simpler language. In each part, construct a DFA for the simpler language, then use it to give the state diagram of a DFA for the language given. In all parts, ∑ = {a, b}.

A**a.** {wl w does not contain the substring ab}

A**b.** {wl w does not contain the substring baba}

c. {wl w contains neither the substrings ab nor ba}

d. {wl w is any string not in a*b*}

e. {wl w is any string not in (ab**+**)*}

f. {wl w is any string not in a* ∪ b*}

g. {wl w is any string that doesn't contain exactly two a's}

h. {wl w is any string except a and b}

Step-by-step solution

Step 1 of 16

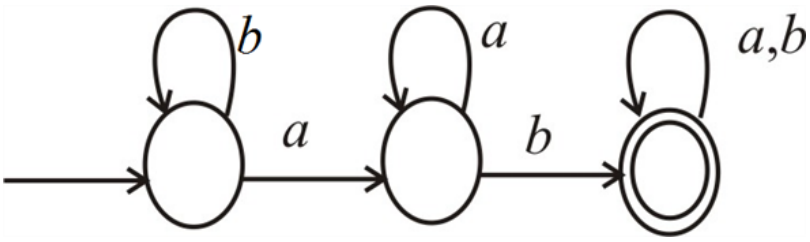
(a) The language is

$\overline{L} = \{ w \mid w$ does not contain the substring *ab*$\}$

$\overline{L}$ is the complement of a simpler language *L*.

Then the simple language is $L = \{ w \mid w$ contain the substring *ab*$\}$

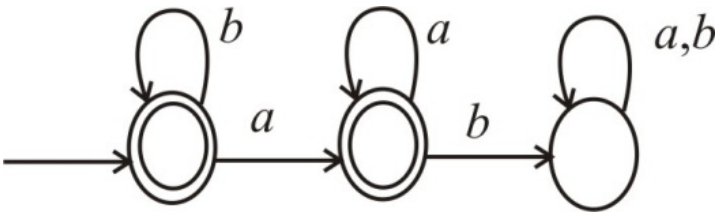
DFA recognizes the language *L* is a follows:



Comments (4)

Step 2 of 16

DFA that recognizes the language $\overline{L}$ is as follows:



Comment

Step 3 of 16

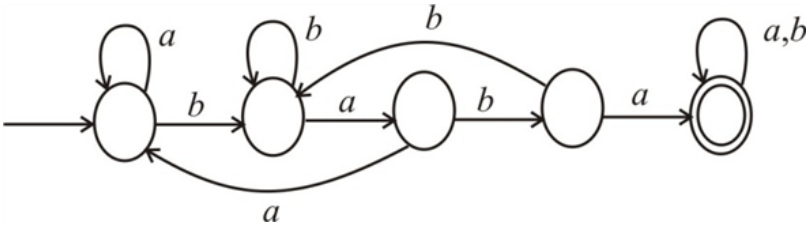
(b) The language is

$\overline{L} = \{ w \mid w$ does not contain the substring *baba*$\}$

$\overline{L}$ is the complement of a simpler language *L*.

Then the simple language is $L = \{ w \mid w$ contain the substring *baba*$\}$

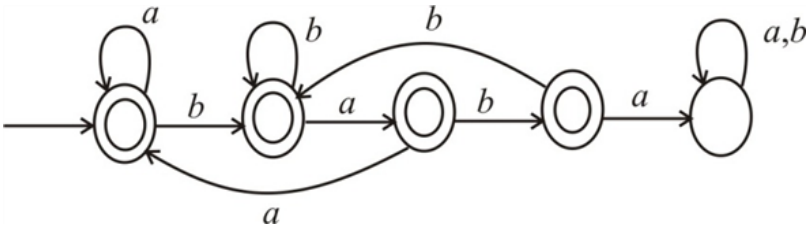
DFA that recognizes the language *L* is as follows:



Comment

Step 4 of 16

DFA that recognizes the language *L* is as follows:



Comment

Step 5 of 16

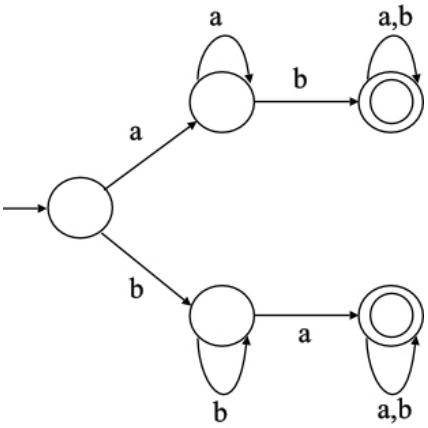
(c) The language is

$\overline{L} = \{ w \mid w$ contains neither the substrings *ab* nor *ba*$\}$

$\overline{L}$ is the complement of a simpler language *L*.

Then the simple language is $L = \{ w \mid w$ contains either the substring *ab* or *ba*$\}$

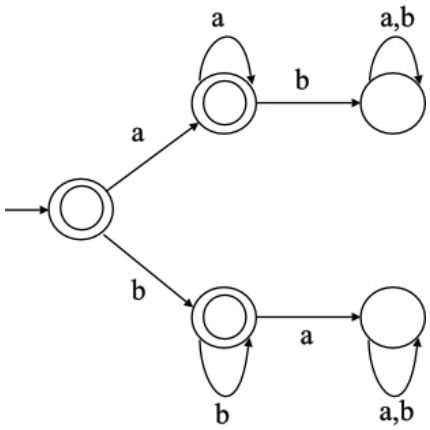
DFA that recognizes the language *L* is as follows



DFA that recognizes the language $\overline{L}$ is as follows:

Comment

Step 6 of 16



Comment

Step 7 of 16

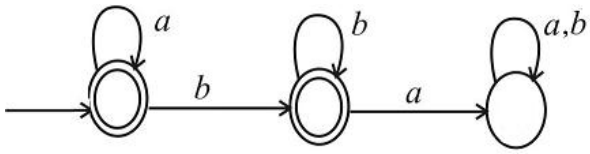
(d) The language is

$\overline{L} = \{ w \mid w$ is any string not in $a^*b^*\}$

$\overline{L}$ is the complement of a simpler language *L*.

Then the simple language is $L = \{ w \mid w$ is any string in $a^*b^*\}$

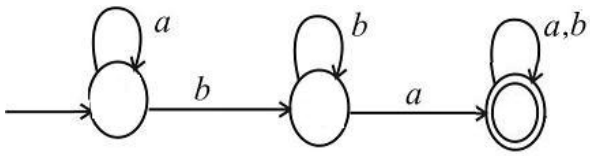
DFA that recognizes the language *L* as follows



Comments (7)

Step 8 of 16

DFA that recognizes the language $\overline{L}$ is as follows:



Comments (4)

Step 9 of 16

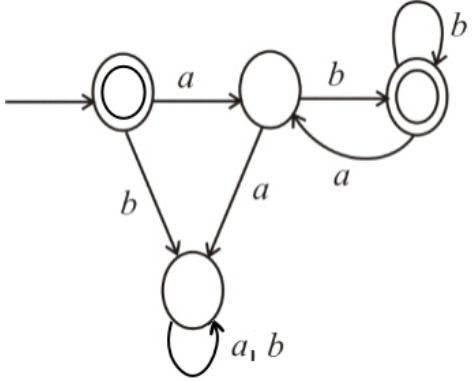
(e) The language is

$\overline{L} = \{ w \mid w$ is any string not in $\left(ab^+\right)^*\}$

$\overline{L}$ is the complement of a simpler language *L*.

Then the simple language is $L = \{ w \mid w$ is any string in $\left(ab^+\right)^*\}$

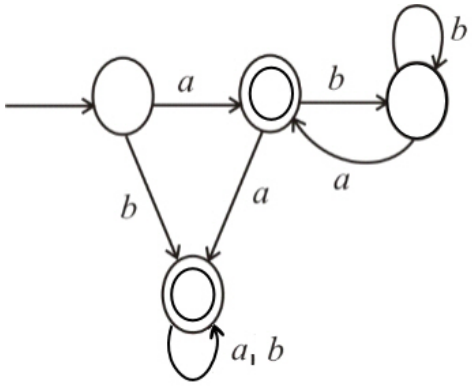
DFA that recognizes the language *L* is as follows:



Comments (7)

Step 10 of 16

DFA that recognizes the language $\overline{L}$ is as follows:



(f)

Comments (5)

Step 11 of 16

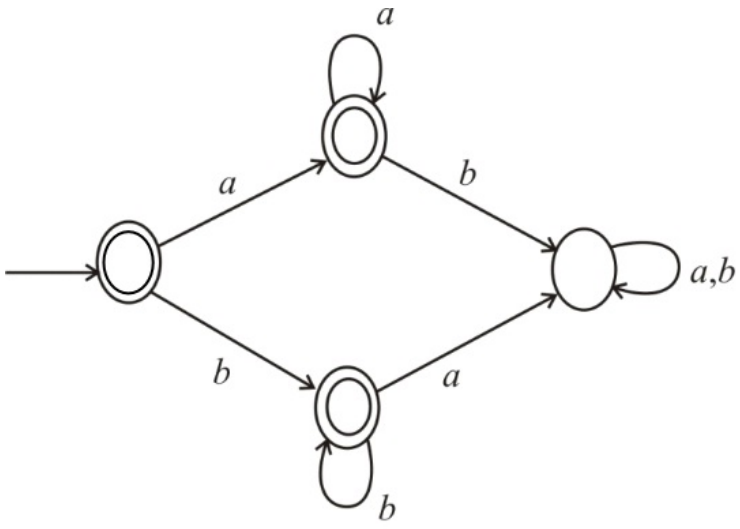
The language is

$\overline{L} = \{ w \mid w$ is any string not in $a^* \cup b^* \}$

$\overline{L}$ is the complement of a simpler language *L*.

Then the simple language is $L = \{ w \mid w$ is any string in $a^* \cup b^* \}$

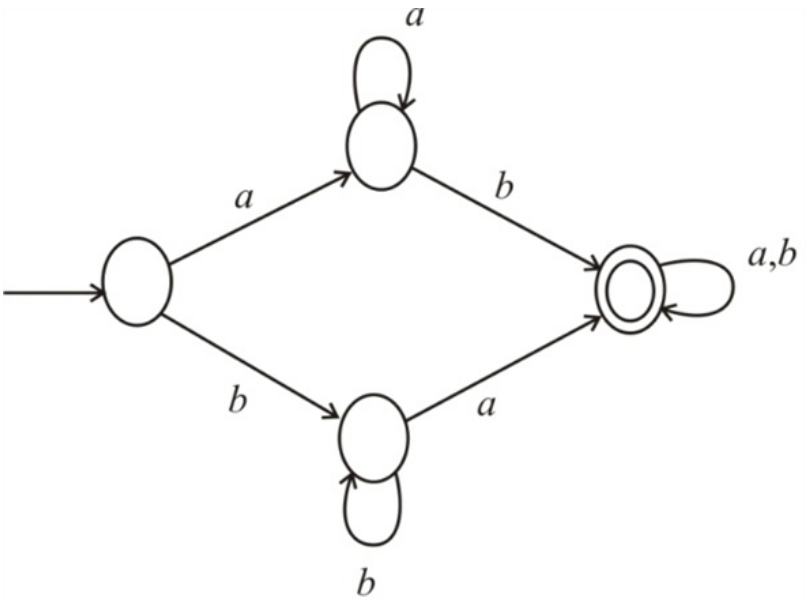
DFA that recognizes the language *L* is as follows:



Comments (3)

Step 12 of 16

DFA that recognizes the language $\overline{L}$ is as follows:



Comment

Step 13 of 16

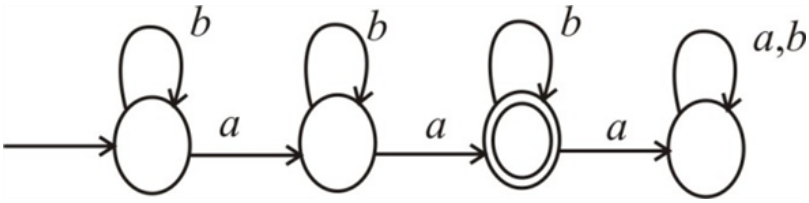
(g) The language is

$\overline{L} = \{ w \mid w$ is any string that doesn't contain exactly two a's$\}$

$\overline{L}$ is the complement of a simpler language *L*.

Then the simple language is $L = \{ w \mid w$ is any string contain exactly two *a*'s$\}$

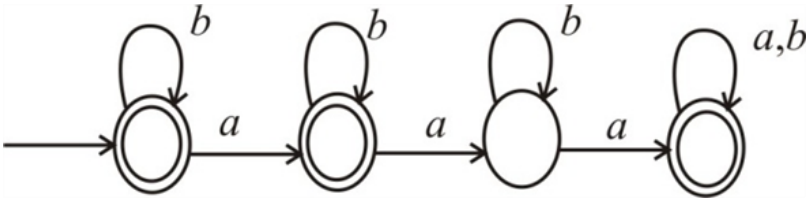
DFA that recognizes *L* is as follows:



Comment

Step 14 of 16

DFA that recognizes $\overline{L}$ is as follows:



Comment

Step 15 of 16

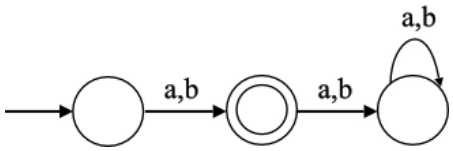
(h) The language is

$\overline{L} = \{ w \mid w$ is any string except *a* and *b*$\}$

$\overline{L}$ is the complement of a simpler language *L*.

Then the simple language is $L = \{ w \mid w$ is *a* and *b*$\}$

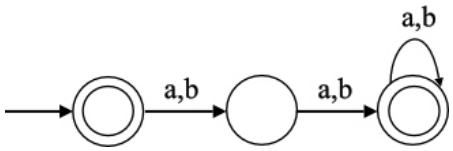
DFA that recognizes *L* is as follows:



Comments (6)

Step 16 of 16

DFA that recognizes $\overline{L}$ is as follows:



Comment

Problem

Give state diagrams of DFAs recognizing the following languages. In all parts, the alphabet is {0,1}.

a. {wl w begins with a 1 and ends with a 0}

b. {wl w contains at least three 1s}

c. {wl w contains the substring 0101 (i.e., w = x0101y for some x and y)}

d. {wl w has length at least 3 and its third symbol is a 0}

e. {wl w starts with 0 and has odd length, or starts with 1 and has even length}

f. {wl w doesn't contain the substring 110}

g. {wl the length of w is at most 5}

h. {wl w is any string except 11 and 111}

i. {wl every odd position of w is a 1}

j. {wl w contains at least two 0s and at most one 1}

k. { ε, 0}

l. {wl w contains an even number of 0s, or contains exactly two 1s}

m. The empty set

n. All strings except the empty string

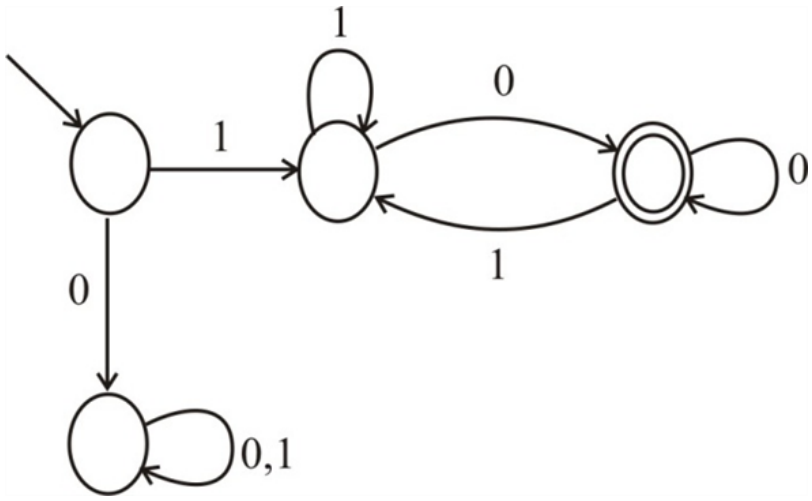
Step-by-step solution

Step 1 of 14

(a)

Language $L = \{ w \mid w$ begins with a 1 and ends with a 0$\}$

State diagram of *DFA* that recognizes *L* is given by:



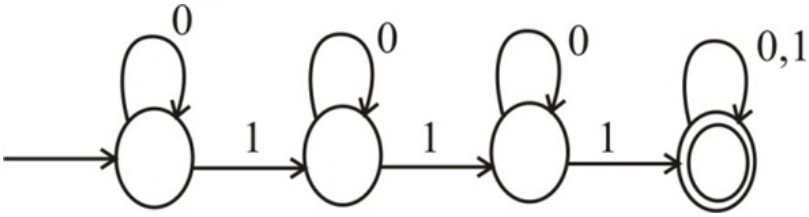
Comments (7)

Step 2 of 14

(b)

Language $L = \{ w \mid w$ contains at least three 1s$\}$

State diagram of *DFA* that recognizes *L* is given by:



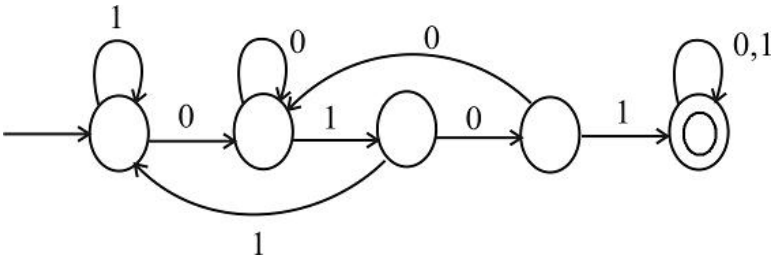
Comments (2)

Step 3 of 14

(c)

Language $L = \{ w \mid w$ contains the substring 0101, i.e. $w = x0101y$ for some x and $y \}$

State diagram of *DFA* that recognizes *L* is given by:



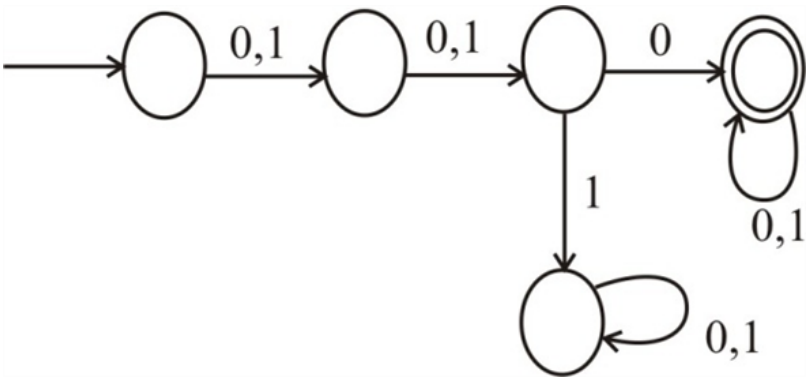
Comment

Step 4 of 14

(d)

Language $L = \{ w \mid w$ has length at least 3 and its third symbol is a 0$\}$

State diagram of *DFA* that recognizes *L* is given by:



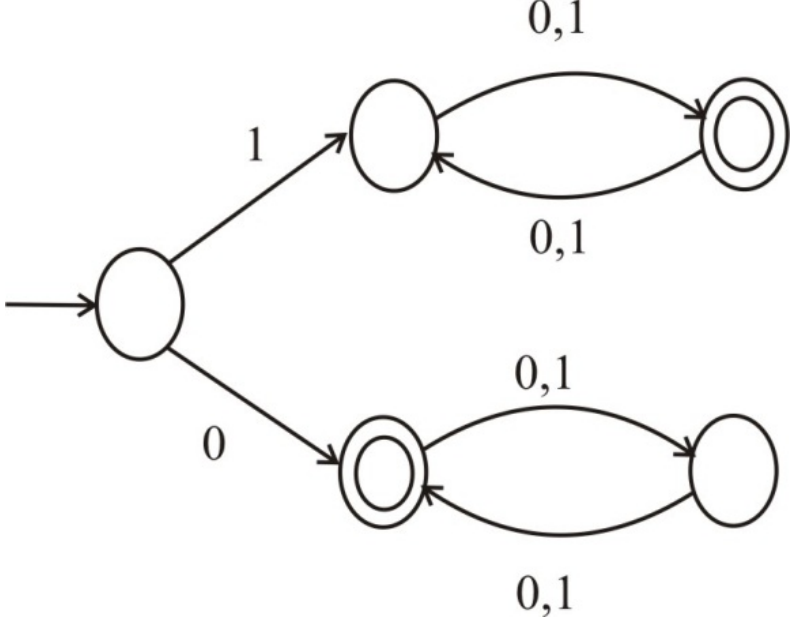
Comment

Step 5 of 14

(e)

Language $L = \{ w \mid w$ starts with 0 and has odd length, or starts with 1 and has even length$\}$

State diagram of *DFA* that recognizes *L* is given by:



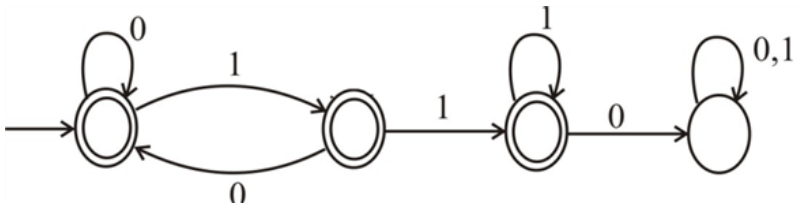
Comments (3)

Step 6 of 14

(f)

Language $L = \{ w \mid w$ doesn't contain the substring 110$\}$

State diagram of *DFA* that recognizes *L* is given by:



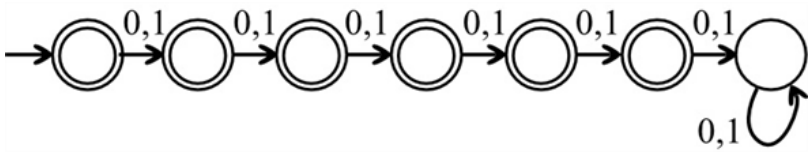
Comment

Step 7 of 14

(g)

Language $L = \{w$ l length of w is at most 5$\}$

State diagram of *DFA* that recognizes *L* is given by:



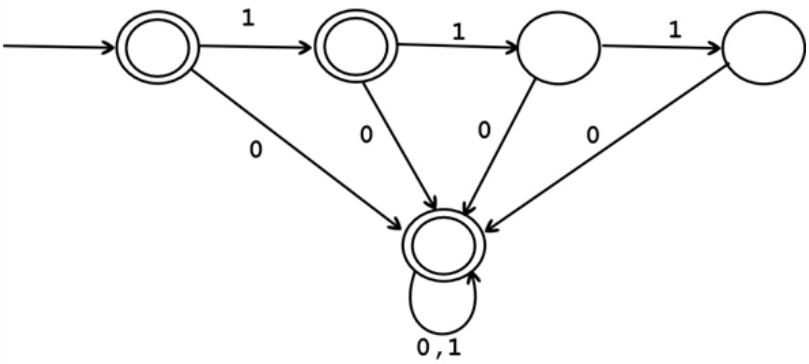
Comments (2)

Step 8 of 14

(h)

Language $L = \{ w \mid w$ is any string except 11 and 111$\}$

State diagram of *DFA* that recognizes *L* is given by:



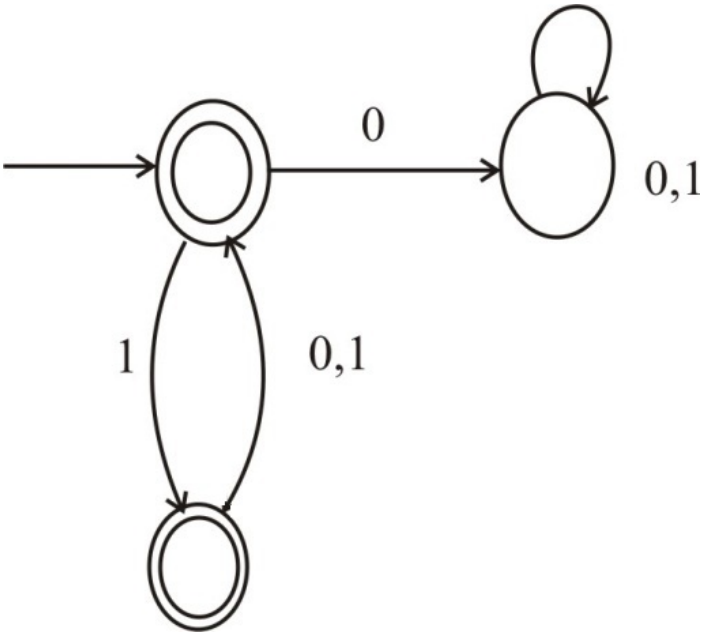
Comments (11)

Step 9 of 14

(i)

Language $L = \{ w \mid w$ every odd position of w is a 1$\}$

State diagram of *DFA* that recognizes *L* is given by:



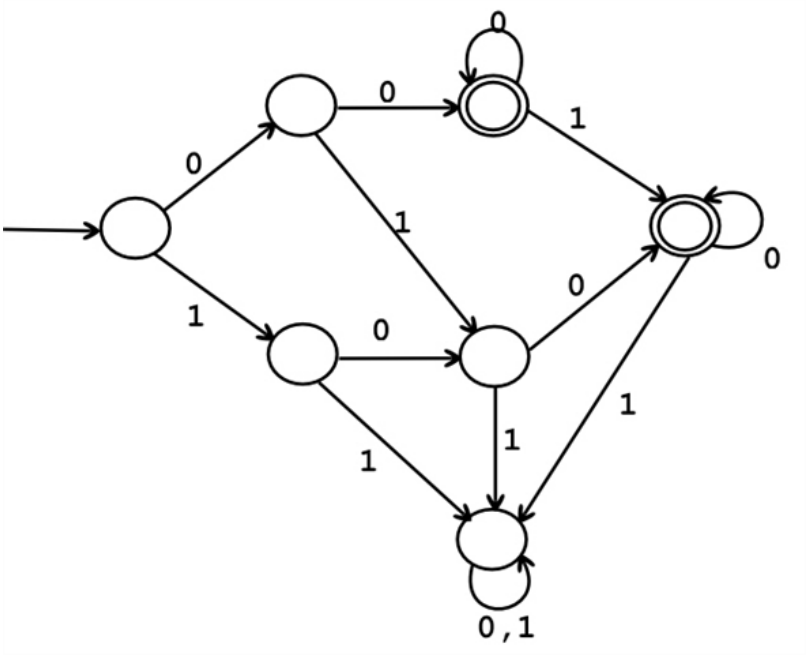
Comments (3)

Step 10 of 14

(j)

Language $L = \{ w \mid w$contain at least two 0s and at most one 1$\}$

State diagram of *DFA* that recognizes *L* is given by:



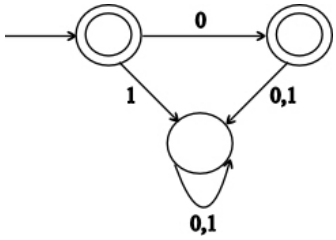
Comment

Step 11 of 14

(k)

Language $L = \{\in, 0\}$

State diagram of *DFA* that recognizes *L* is given by:



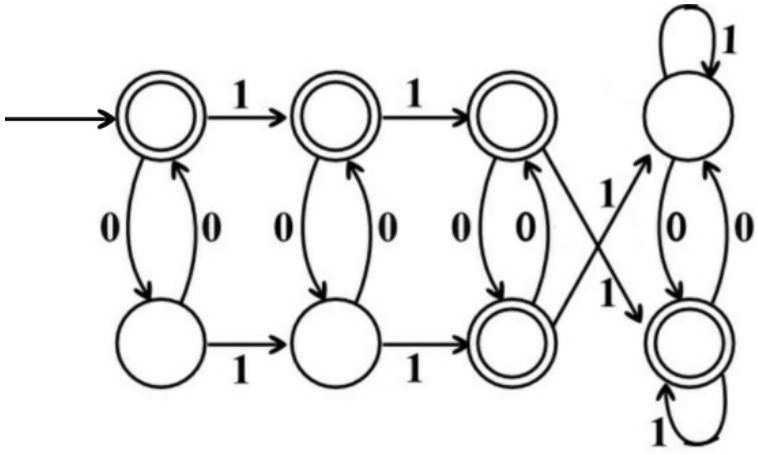
Comment

Step 12 of 14

(l)

Language $L = \{ w \mid w$ contains an even number of 0s, or contains exactly two 1s$\}$.

State diagram of *DFA* that recognizes *L* is given by:



The language L accepts the strings that contain even number of 0s or contains exactly two 1s. This language accepts the string if any one of the two conditions is satisfied.

Consider the string 11, the string 11 contains zero number of 0s which is even. In this case, the first condition is accepted. Thus, the language L accepts the string 11.

Comments (10)

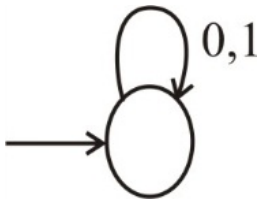
Step 13 of 14

(m)

Language *L*= {The empty set}

The empty set does not contain the null string. The language *L* does not accept any string even the null string.

State diagram of *DFA* that recognizes *L* is given by:



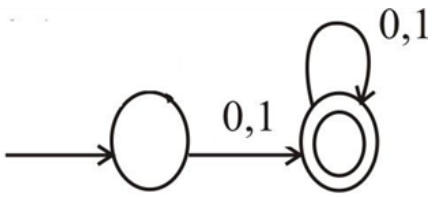
Comments (2)

Step 14 of 14

(n)

Language L ={all strings except the empty string}

State diagram of *DFA* that recognizes *L* is given by:



Comment

Problem

Give state diagrams of NFAs with the specified number of states recognizing each of the following languages. In all parts, the alphabet is {0,1}.

A**a.** The language {wl w ends with 00} with three states

b. The language of Exercise 1.6c with five states

c. The language of Exercise 1.6l with six states

d. The language {0} with two states

e. The language $0^*1^*0^+$ with three states

A**f.** The language $1^*(001^+)^*$ with three states

g. The language { ε } with one state

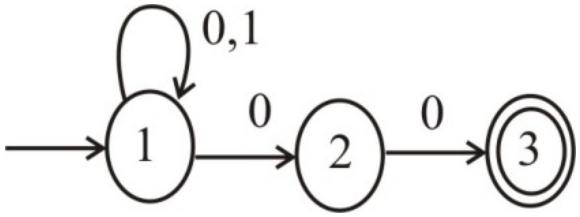
h. The language 0^* with one state

Step-by-step solution

Step 1 of 8

a.

Consider the Language $L = \{w \mid w \text{ends with } 00\}$ with three states over the alphabet $\Sigma = \{0,1\}$. The language states that the Finite automata should consist of three states that accept the strings over the alphabet $\Sigma = \{0,1\}$ and ends with 00. Let *M* be the NFA that recognizes L. The state diagram of *M* is as follows:



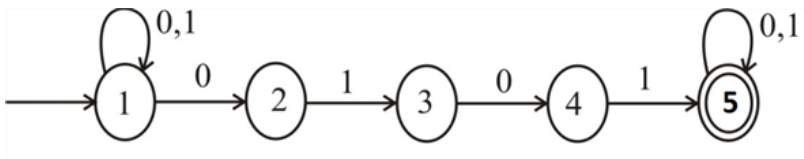
Comment

Step 2 of 8

b.

Consider the Language

$L = \{w \mid w \text{ contains the substring } 0101 \text{ i.e, } w = x0101y \text{ for some } x \text{ and } y\}$ with five states over the alphabet $\Sigma = \{0,1\}$. The language states that the Finite automata should consist of five states that accept the strings over the alphabet $\Sigma = \{0,1\}$ and contains the substring 0101. Let *M* be the NFA that recognizes L. The state diagram of *M* is as follows:



Comment

Step 3 of 8

c.

Consider the Language

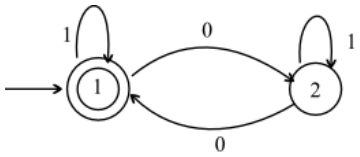
$L = \{w \mid w \text{ contains an even number of 0s, or contains exactly two 1s}\}$ with 6 states over the alphabet $\Sigma = \{0,1\}$. Let *M* be the NFA that recognizes L. Divide *L* into 2 languages L_1 and L_2.

$L_1 = \{w \mid w \text{ contains on even number of 0s}\}$

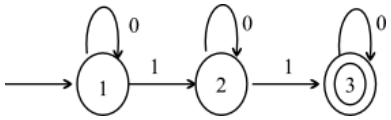
$L_2 = \{w \mid w \text{ contains exactly two 1s}\}$

Let M_1, M_2 be the NFAs recognizes L_1, L_2 respectively.

State diagram of M_1 is as follows:

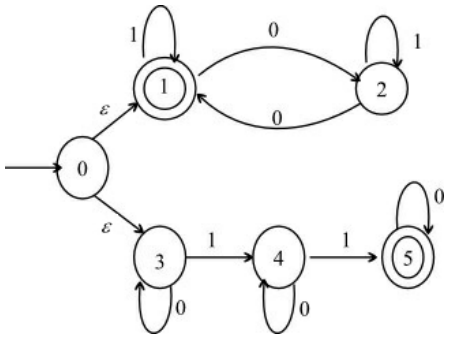


The state diagram of M_2 is as follows:



Now $L = L_1 \cup L_2$

The sate diagram of *M* that recognizes *L* is as follows:

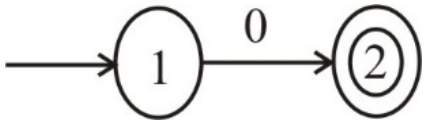


Comments (2)

Step 4 of 8

d.

Consider the Language $L_1 = \{w \mid w \text{ contains only } 0\}$ with 2 states over the alphabet $\Sigma = \{0,1\}$. Let *M* be the NFA that recognizes L_1. The state diagram of *M* is as follows:

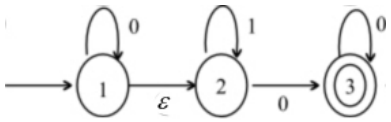


Comment

Step 5 of 8

e.

Consider the Language $L = \{w \mid w \text{ contains only } 0^*1^*0^+\}$ with 3 states over the alphabet $\Sigma = \{0,1\}$. The language states that the finite automata accept all the strings containing any number of zeroes and ones followed by at least one zero. Let *M* be the NFA that recognizes L. The state diagram of *M* is as follows:

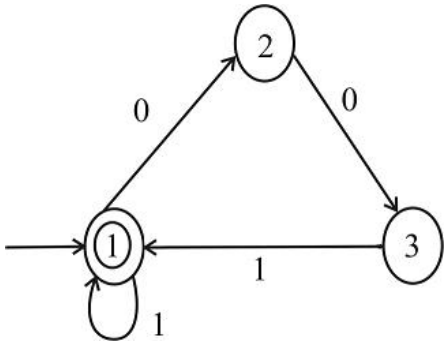


f.

Comments (21)

Step 6 of 8

Consider the Language L that accepts the strings of the form $1*(001^{+})*$ with 3 states over the alphabet $\Sigma = \{0,1\}$. Let *M* be the NFA that recognizes L. The state diagram of *M* is as follows:

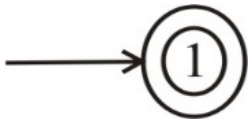


Comment

Step 7 of 8

g.

Consider the Language $L_1 = \{w \mid w \text{ the language results in empty string } \varepsilon\}$ with one state over the alphabet $\Sigma = \{0,1\}$. The language states that the finite automata accept a null string. Let *M* be the NFA that recognizes L_1. The state diagram of *M* is as follows:

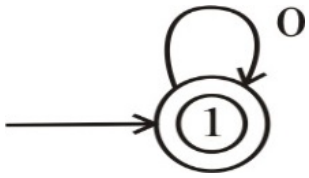


Comment

Step 8 of 8

h.

Consider the Language *L* that accepts the strings of the form $0*$with one state over the alphabet $\Sigma = \{0,1\}$. Let *M* be the NFA that recognizes *L*. The state diagram of *M* is as follows:



Comment

Problem

Use the construction in the proof of Theorem 1.45 to give the state diagrams of NFAs recognizing the union of the languages described in

a. Exercises 1.6a and 1.6b.

b. Exercises 1.6c and 1.6f.

THEOREM **1.45**

The class of regular languages is closed under the union operation.

Step-by-step solution

Step 1 of 9

a)

Consider the languages,

$L_1 = \{ w \mid w$ begins with 1 and ends with *a* 0 $\}$ and

$L_2 = \{ w \mid w$ contains at least three 1s$\}$ on $\Sigma = \{0,1\}$

M_1 be the NFA that recognizes L_1 and

M_2 be the NFA that recognizes L_2.

Comment

Step 2 of 9

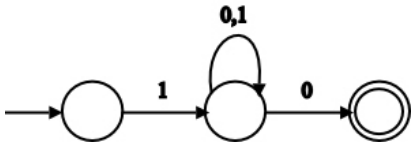
Let $L = L_1 \cup L_2$

Now *M* be the NFA that recognizes *L*.

- $L_1 = \{ w \mid w$ begins with *a* 1 and ends with *a* 0$\}$

$L_1 = 1(0,1)^* 0$

The state diagram of M_1 that recognizes L_1 is



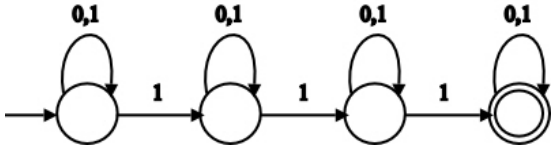
Comments (5)

Step 3 of 9

- $L_2 = \{ w \mid w$ contains at least three 1s$\}$

$L_2 = (0,1)^* 1(0,1)^* 1(0,1)^* 1(0,1)^*$

The state diagram of M_2 that recognizes L_2 is

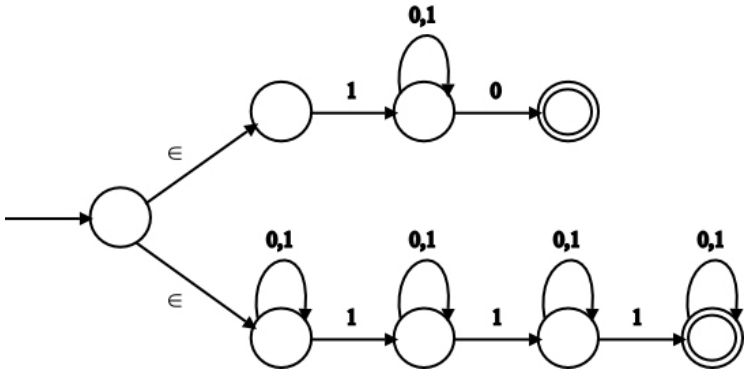


Comments (3)

Step 4 of 9

Now *L* is union of L_1 and L_2.

So the state diagram of *M* that recognizes *L* is described as follows.



Comments (1)

Step 5 of 9

(b) Languages are

$L_1 = \{w \mid w \text{ contains the substring } 0101 \text{ i.e, } w = x0101y \text{ for some } x \text{ and } y\} \text{on} \sum = \{0,1\}$

$L_1 = \{w \mid w \text{ doesn't contain the substring } 110\}$ on $\Sigma = \{0,1\}$

M_1 be NFA that recognizes L_1 and

M_2 be the NFA that recognizes L_2

Comment

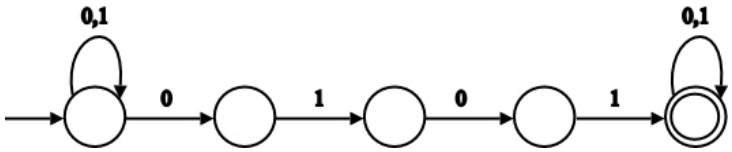
Step 6 of 9

Let $L = L_1 \cup L_2$

Now *M* be the NFA that recognizes *L*.

- $L_1 = \{w \mid w \text{ contains the substring } 0101\}$

The state diagram of M_1 that recognizes L_1 is

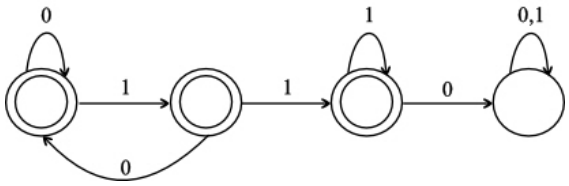


Comments (3)

Step 7 of 9

- $L_2 = \{w \mid w \text{ doesn't contain the substring } 110\}$

First we give the state diagram of the machine which recognize the language machine which recognize the language



Comments (4)

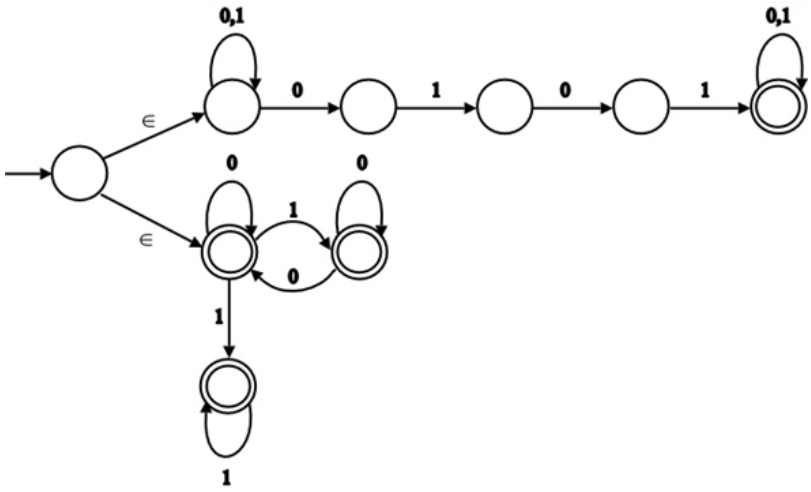
Step 8 of 9

Now *L* is the union of L_1 and L_2

So the state diagram of *M* that recognizes *L* is described as follows

Comment

Step 9 of 9



Comments (3)

Problem

Use the construction in the proof of Theorem 1.47 to give the state diagrams of NFAs recognizing the concatenation of the languages described in

a. Exercises 1.6g and 1.6i.

b. Exercises 1.6b and 1.6m.

THEOREM **1.47**

The class of regular languages is closed under the concatenation operation.

Step-by-step solution

Step 1 of 8

(a) Languages are

$L_1 = \{w \mid \text{the length of } w \text{ is at most } 5\}$ on $\Sigma = \{0,1\}$

And $L_2 = \{w \mid \text{every odd position of } w \text{ is a } 1\}$ on $\Sigma = \{0,1\}$

M_1 be the NFA that recognizes L_1 and

M_2 be the NFA that recognizes L_2.

Comment

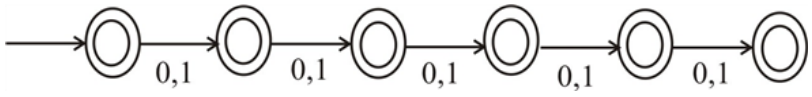
Step 2 of 8

Let $L = L_1 0 L_2$

M be the NFA that recognizes *L*.

- $L_1 = \{w \mid \text{ the length of } w \text{ is at most } 5\}$

The state diagram of M_1 that recognizes L_1 is



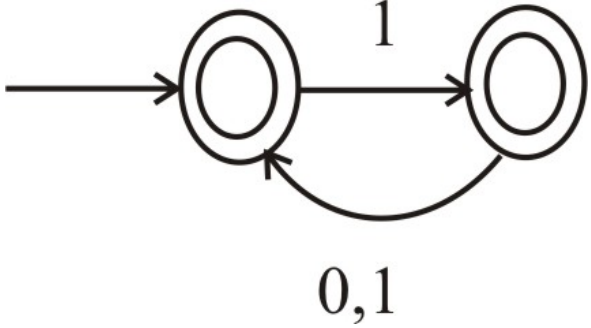
Comment

Step 3 of 8

- $L_2 = \{w\}$ every odd position of w is a $1\}$

$L_2 = (1\Sigma)^*$

The state diagram of M_2 that recognizes L_2 is

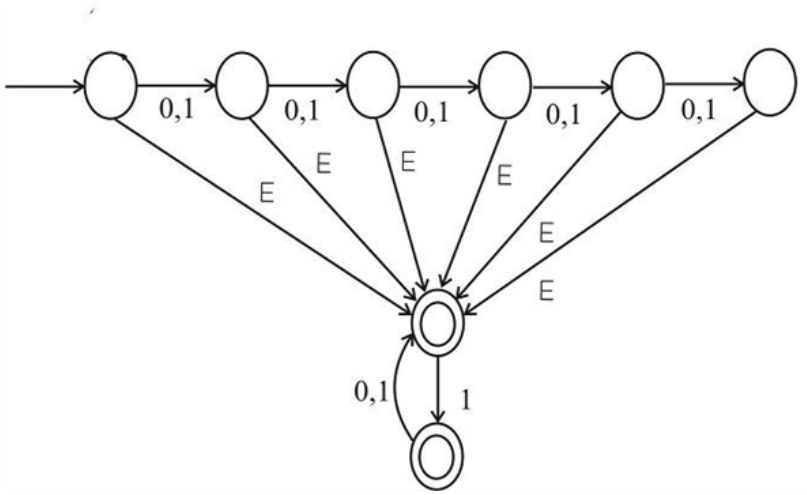


Comment

Step 4 of 8

L is concatenation of L_1 and L_2

So the state diagram of M that recognizes *L* is described as follows



Comment

Step 5 of 8

(b) Given Languages are

$L_1 = \{w \mid w \text{ contains at least three 1s}\}$ on $\Sigma = \{0,1\}$

And $L_2 = \{w \mid w \text{ is a empty set}\}$ on $\Sigma = \{0,1\}$

M_1 be the NFA that recognizes L_1 and

M_2 be the NFA that recognizes L_2.

Comment

Step 6 of 8

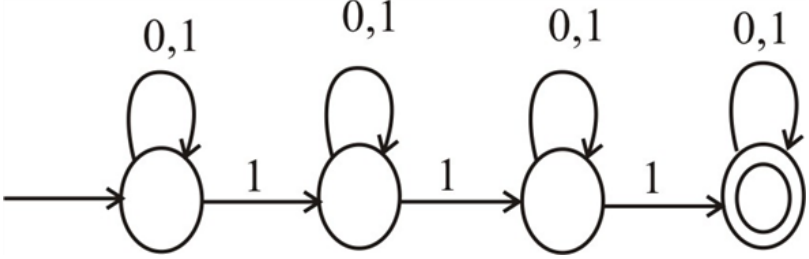
Let $L = L_1 0 L_2$

M be the NFA that recognizes *L*.

- $L_1 = \{w \mid w \text{ contains at least three 1s}\}$

$L_1 = (0,1)^* 1 (0,1)^* 1 (0,1)^* (0,1)^*$

The state diagram of M_1 that recognizes L_1 is



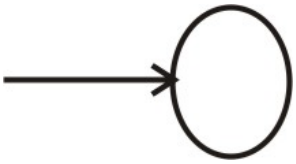
Comments (2)

Step 7 of 8

- $L_2 = \{w \mid w \text{ is a empty set}\}$

$L_2 = \phi = \{\ \}$

The state diagram of M_2 that recognizes L_2 is

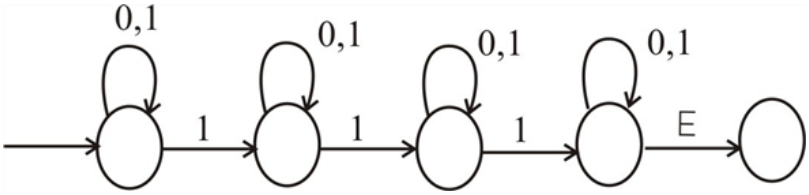


Comments (2)

Step 8 of 8

L is concatenation of L_1 and L_2

So the state diagram of M that recognizes *L* is described as follows



Comment

Problem

Use the construction in the proof of Theorem 1.49 to give the state diagrams of NFAs recognizing the star of the languages described in

a. Exercise 1.6b.

b. Exercise 1.6j.

c. Exercise 1.6m.

THEOREM **1.49**

The class of regular languages is closed under the star operation.

Step-by-step solution

Step 1 of 3

(a) Language $L_1 = \{w \mid w \text{contains at least three 1s}\}$

Let M_1 be the NFA that recognizes L_1.

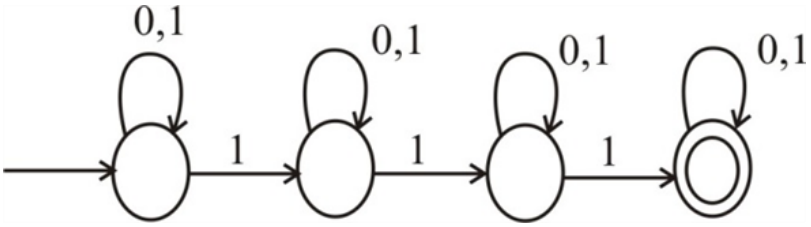
Let $L = L_1^*$

Let M be the NFA that recognizes L.

$L_1 = \{w \mid w \text{ contains at least three 1s}\}$

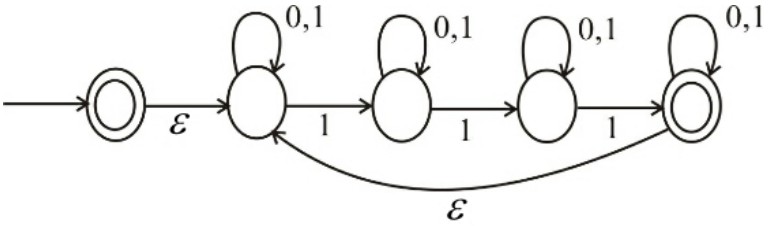
$L_1 = (0,1)^* 1 (0,1)^* 1 (0,1)^* 1 (0,1)^*$

The state diagram of M_1 that recognizes L_1 is as follows:



L is the language that recognizes star of L_1

The state diagram of M that recognizes L is as follows:



Comments (7)

Step 2 of 3

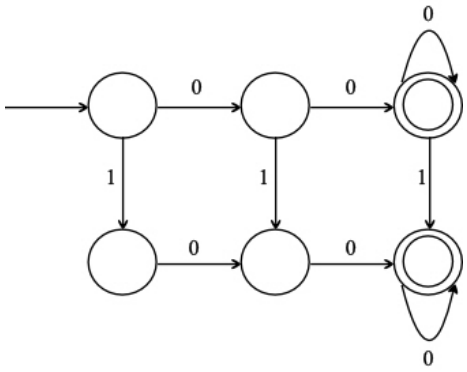
(b) Languages $L_1 = \{w \mid w \text{ contains at least two 0s and at most one 1}\}$

Let M_1 be the NFA that recognizes L_1.

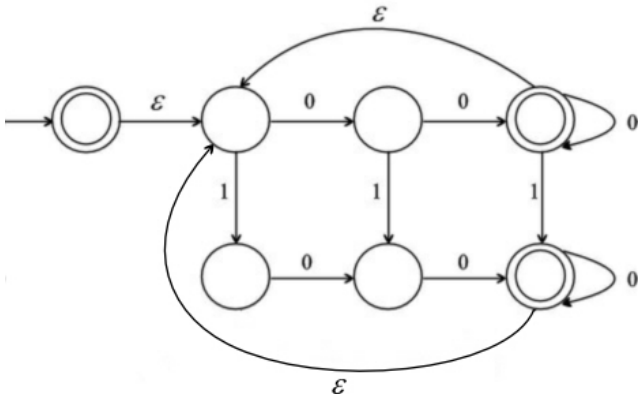
Let $L = L_1^*$

Let *M* be the NFA that recognizes *L*.

$L_1 = \{w \mid w \text{ contains atleast two 0s and at most one 1}\}$



The state diagram of *M* that recognizes *L* is as follows:



Comment

Step 3 of 3

(c) Languages $L_1 =$ The empty set.

Let M_1 be the NFA that recognizes L_1.

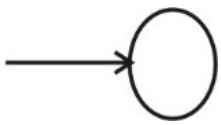
Let $L = L_1^*$

Let *M* be the NFA that recognizes *L*.

$L_1 =$ The empty set

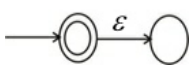
$L_1 = \phi = \{\ \}$

The state diagram of M_1 that recognizes L_1 is as follows:



L is the star of L_1.

The state diagram of *M* that recognizes *L* is as follows:



Comment

Problem

Prove that every NFA can be converted to an equivalent one that has a single accept state.

Step-by-step solution

Step 1 of 2

Let *M* be a NFA (non deterministic finite automatic)

Let *N* be the another NFA with single accept states q_{final}.

We go through every accept state *M* and do the following

(i) make it non – accepting state

(ii) add an $\in$-transition from that state to q_{final}

Then we will get NFA *N*.

If M has no accept states, then there will be no transitions coming into q_{final}.

Comment

Step 2 of 2

Now we will discuss this formally.

$M=(Q,\Sigma,\delta,q_0,F)$ then

$N=(Q\cup\{q_{\text{final}}\},\Sigma,\delta',q_0,\{q_{\text{final}}\})$ for any $q\in Q$ and $a\in\Sigma$

$$\delta'(q,a)=\begin{Bmatrix}\delta(q,a) \text{ if } a\neq\in \text{ or } q\notin F\\ \delta(q,a)\cup\{q_{\text{final}}\} \text{ if } a=\in \text{ and } q\in F\end{Bmatrix}$$

And $q'(q_{\text{final}},a)=\phi$

Thus we get *N* by simply making every accept state of *M* as non – accepting state and adding an $\in-$ transition from that state to q_{final}

Thus M is equivalent to N.

Thus every NFA is converted to an equivalent one that has single accept state.

Comments (3)

Problem

Let D = {wl w contains an even number of a's and an odd number of b's and does not contain the substring ab}. Give a DFA with five states that recognizes D and a regular expression that generates D. (Suggestion: Describe D more simply.)

Step-by-step solution

Step 1 of 4

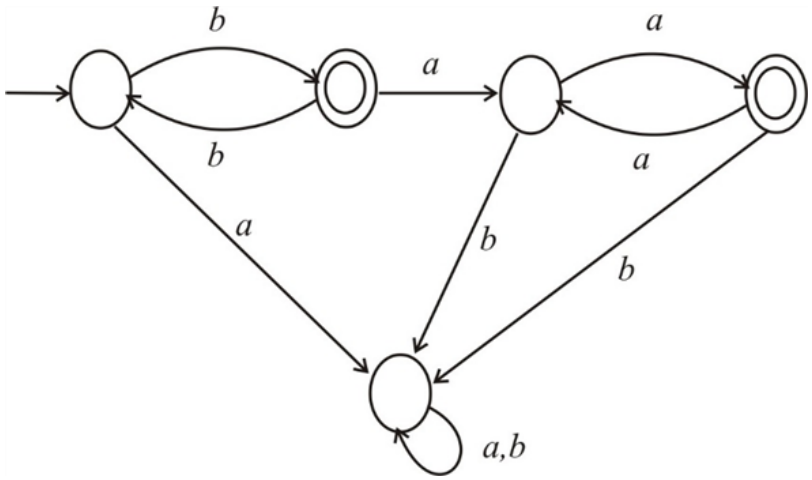
Consider the language $D = \{ w \mid w$ contains an even number of $a's$ and an odd number of $b's$ and does not contain the substring *ab*$\}$.

The language D can be described simply as follows $D = \{ w \mid w$ contains an odd number of *b*'s followed by even number of *a*'s$\}$.

Comment

Step 2 of 4

Let *M* be the DFA with five states that recognizes the language D. The state diagram of M is as follows:



Comments (6)

Step 3 of 4

The language accepts the strings like $\{b, baa, bbbaaaa, ...\}$. The string *b* is accepted by the language because, it contains the odd number of *b*'s (1) followed by even number of *a*'s (0).

Comment

Step 4 of 4

Now, the language D can be expressed as combination of following two languages D_1 and D_2.

$D_1 = \{w \mid w \text{ contain odd number of b's}\}$

$D_2 = \{w \mid w \text{ contains even number of a's}\}$

$D = D_1 o D_2$

R_1 be the regular expression that generates D_1

R_2 be the regular expression that generates D_2

R be the regular expression that generates D

$R = R_1 o R_2$

$R_1 = b(bb)^*$

$R_2 = (aa)^*$

$R = b(bb)^* o (aa)^*$

$R = b(bb)^*(aa)^*$

Therefore, the regular expression that generates the language D is $b(bb)^*(aa)^*$.

Comment

Problem

Let F be the language of all strings over {0,1} that do not contain a pair of 1s that are separated by an odd number of symbols. Give the state diagram of a DFA with five states that recognizes F. (You may find it helpful first to find a 4 state NFA for the complement of F.)

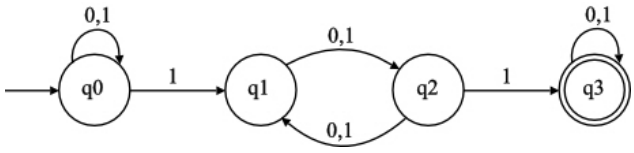
Step-by-step solution

Step 1 of 4

Consider the language *F* that accepts the strings over {0,1} that do not contain a pair of 1s that are separated by an odd number of symbols.

The complement of the language $\overline{F}$ accepts the strings over {0,1} that contain a pair of 1s that are separated by an odd number of symbols. The language accepts other strings as well.

The NFA for $\overline{F}$ is as follows:



The transition table for $\overline{F}$ is as follows:

State	0	1
q0	q0	{q0,q1}
q1	q2	q2
q2	q1	{q1,q3}
q3	q3	q3

Comment

Step 2 of 4

Now, Convert the NFA to DFA.

The following are the steps to convert NFA to DFA:

Step 1: Create an empty set for the set of states of DFA. Initially $Q' = \phi$.

Step 2: Identify the start state in the NFA and make it as a start sate for the DFA. Add the initial state to the set of states of DFA.

Step 3: For every new state find the possible set of states for each input symbol using transition function of NFA and add new states to Q'. Start the procedure with the initial state.

Step 4: If no new states are found then stop the procedure. The final state of DFA will be all states which contain final states of NFA.

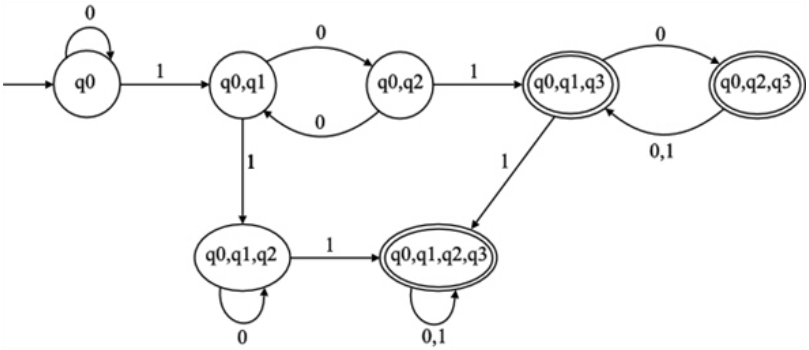
The transition table of the DFA is as follows:

State	0	1
q0	q0	{q0,q1}
{q0,q1}	{q0,q2}	{q0,q1,q2}
{q0,q2}	{q0,q1}	{q0,q1,q3}
{q0,q1,q2}	{q0,q1,q2}	{q0,q1,q2,q3}
{q0,q1,q3}	{q0,q2,q3}	{q0,q1,q2,q3}
{q0,q1,q2,q3}	{q0,q1,q2,q3}	{q0,q1,q2,q3}
{q0,q2,q3}	{q0,q1,q3}	{q0,q1,q3}

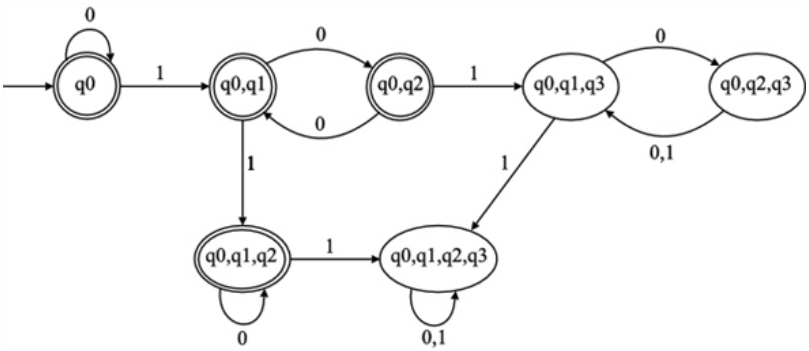
Comment

Step 3 of 4

Draw the DFA using the above transition table.



The above DFA should be complemented to get the DFA for the language *F*. Change the final states to non-final states and vice versa.

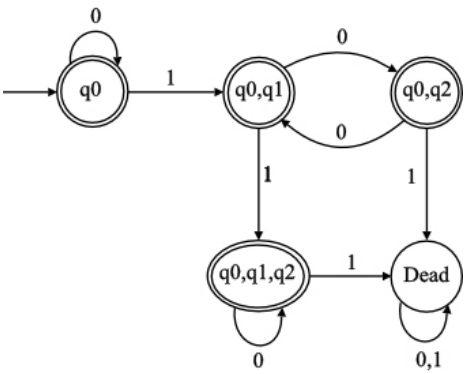


The states {q0,q1,q3}, {q0,q2,q3} and {q0,q1,q2,q3} are dead states. Combine all the three dead states into a single state.

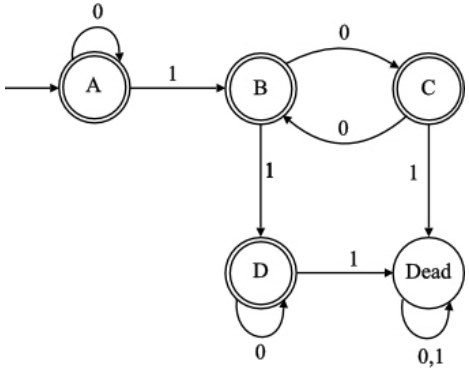
Comment

Step 4 of 4

The DFA after combining the 3 states is as follows:



Rename the states in the above DFA for simplicity. The minimized DFA that accepts the language *F* is as follows:



Comments (2)

Problem

a. Show that if M is a DFA that recognizes language B, swapping the accept and nonaccept states in M yields a new DFA recognizing the complement of B. Conclude that the class of regular languages is closed under complement.

b. Show by giving an example that if M is an NFA that recognizes language C, swapping the accept and nonaccept states in M doesn't necessarily yield a new NFA that recognizes the complement of C. Is the class of languages recognized by NFAs closed under complement? Explain your answer.

Step-by-step solution

Step 1 of 4

(a)

M is a DFA that recognizes the regular language *B*

Let M' be the new DFA that has swapped accept and non accept states in M.

• Consider M' accepts a string x.

• Run M' on x then M' surely enters into accept state.

• The machines M, M' have swapped accept and non-accept states.

• So if we run M on *x*, then M will end in a non-accept state.

• Thus if *x* is accepted by M' then *x* does not accepted by *M*.

• Similarly if *x* is accepted by *M* then *x* does not accepted by M'.

• One can say that if $x \in B$ then $x \notin \overline{B}$ (complement of B) and vice – versa

• So M' will accept the strings that are not accepted by *M*.

• Therefore M' recognizes the languages which are complement of B.

• As M recognizes a regular language *B*, there exists M' which recognizes complement of *B* which is also regular.

• Hence class of regular languages closed under complement.

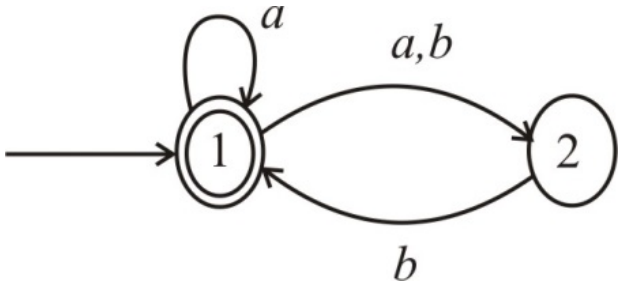
Comment

Step 2 of 4

(b)

Let *M* be a NFA that recognizes a languages *C*.

The state diagram of *M* is as follows

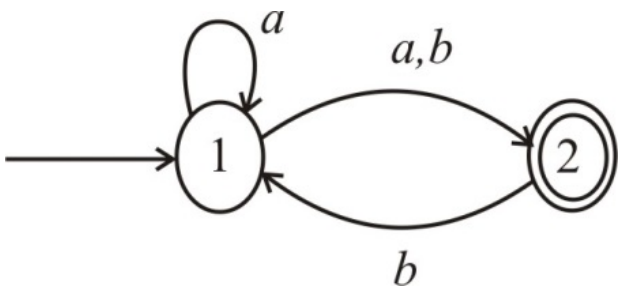


Comments (5)

Step 3 of 4

Clearly the string a is accepted by *M*.

If we swap accept and non accept states of *M* then we will get the following M'



Comment

Step 4 of 4

Clearly the string a is accepted by M'.

This shown that swapping accept and non - accept states of NFA doesn't necessarily yield a new NFA recognizing the complement of the original one.

Consider the facts,

"The class of languages recognized by NFAs is precisely the class of languages recognizes by DFAs" …… (a)

"The class of languages recognized by DFAs is closed under complement" …… (b)

Therefore from the facts (a) and (b)

Class of languages recognize by NFAs closed under complement.

Comment

Problem

Give a counterexample to show that the following construction fails to prove Theorem 1.49, the closure of the class of regular languages under the star operation.7

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1. Construct $N = (Q_1, \Sigma, \delta, q_1, F)$ as follows. N is supposed to recognize A_1^*.

a. The states of N are the states of N_1.

b. The start state of N is the same as the start state of N_1.

c. $F = \{q_1\} \cup F_1$.
The accept states F are the old accept states plus its start state.

d. Define δ so that for any $q \in Q_1$ and any $a \in \Sigma_\varepsilon$,

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \notin F_1 \text{ or } a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon. \end{cases}$$

(Suggestion: Show this construction graphically, as in Figure 1.50.)

Figure 1.50

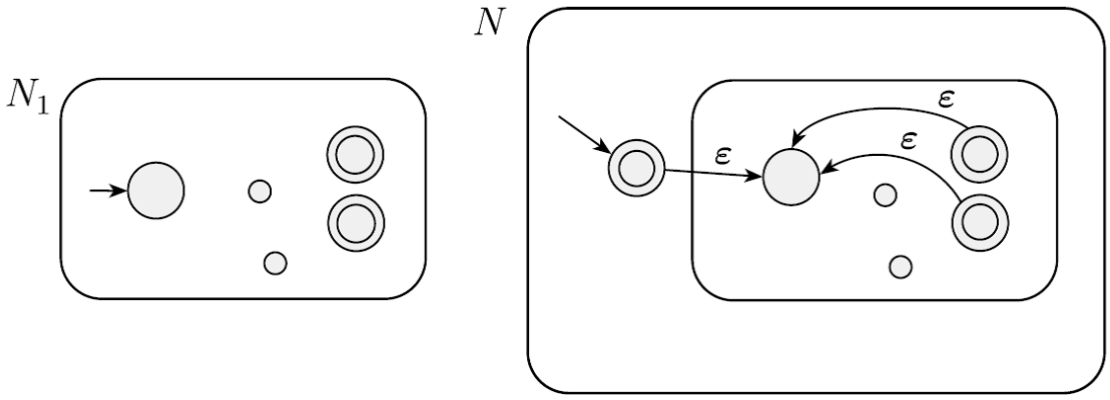


FIGURE 1.50
Construction of N to recognize A^*

Step-by-step solution

Step 1 of 1

Theorem 1.49 states that "*The class of regular languages closed under the star operation*".

Consider the data,

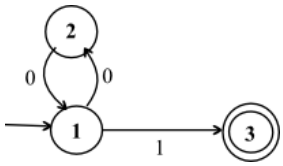
A language *A* is recognized by the automata $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$.

Assume *N* is the Non-deterministic finite automata which recognize the language A_1^*.

Example:

Assume a language $A_1 = \{(00)^* 1\}$.

The finite state automata N_1, which recognizes the language A_1 is as follows:



The following procedure is used to construct the finite state automata *N*, which recognizes the language A_1^*:

a. States of *N* are the states of N_1.

States of N_1 are {1,2,3}.

So, States of *N* are {1,2,3}.

b. The start state of *N* is same as start state of N_1.

Start state of N_1 is {1}.

So, start state of *N* is {1}.

c. $F = \{q_1\} \cup F_1$. The accept states for F are the accept states of F_1 including the start state. So, the accept state for F are **1** and **3.**

d. Define the transition δ for any $q \in Q_1$ and any $a \in \Sigma_\epsilon$ by using the following transition:

$$\delta(q,a) = \begin{cases} \delta^1(q,a) & q \notin F_1 \text{ and } a \neq \in \\ \delta^1(q,a) \cup \{q_1\} & q \in F_1 \text{ and } a = \in \end{cases}$$

$$\begin{aligned} \delta(1,\in) &= \delta_1(1,\in) \cup \{1\} \\ &= \phi \cup \{1\} \\ &= \{1\} \end{aligned}$$

$$\begin{aligned} \delta(3,\in) &= \delta_1(3,\in) \cup \{1\} \\ &= \phi \cup \{1\} \\ &= \{1\} \end{aligned}$$

$$\begin{aligned} \delta(1,0) &= \delta_1(1,0) \\ &= 2 \end{aligned}$$

$$\begin{aligned} \delta(1,1) &= \delta_1(1,1) \\ &= 3 \end{aligned}$$

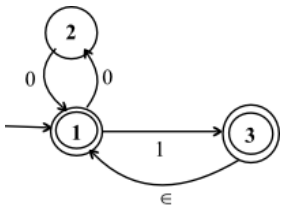
$$\begin{aligned} \delta(2,0) &= \delta_1(2,0) \\ &= 1 \end{aligned}$$

$$\begin{aligned} \delta(2,1) &= \delta_1(2,1) \\ &= \phi \end{aligned}$$

$$\begin{aligned} \delta(3,0) &= \delta_1(3,0) \\ &= \phi \end{aligned}$$

$$\begin{aligned} \delta(3,1) &= \delta_1(3,1) \\ &= \phi \end{aligned}$$

The state diagram for the Finite State automata is as follows:



• The above finite automata adds the start state to the set of accept states, which adds some other undesired strings and $\in$ to the recognized language .

• A new start state which is also an accept state is not added to the automata. Thus, the new state is not added to the automata and leads to different automata from original automata.

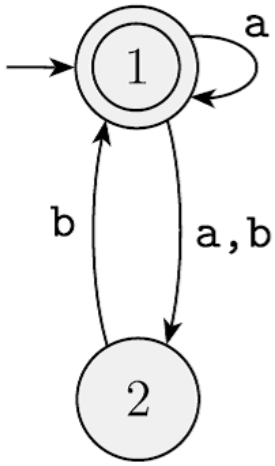
Comment

Problem

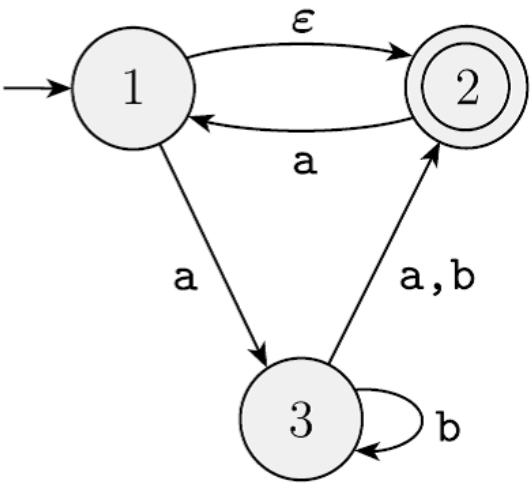
Use the construction given in Theorem 1.39 to convert the following two nondeterministic finite automata to equivalent deterministic finite automata.

THEOREM **1.39**

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.



(a)



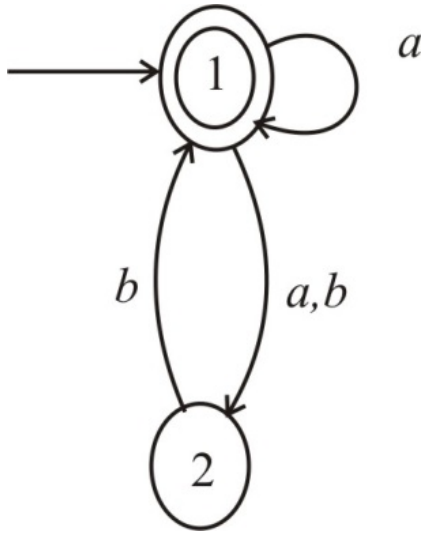
(b)

Step-by-step solution

Step 1 of 2

a.

Consider the Non-deterministic Finite Automata,



By using Theorem 1.39, "*For every non-deterministic finite automata, there is an equivalent Deterministic finite automation*".

Constructing equivalent DFA for the given NFA:

1. $Q^1 = P(Q)$ where Q^1 is the subset of all sets of *Q*.

So, $Q^1 = \{\phi, \{1\}, \{2\}, \{1,2\}\}$

2. For an element *R* in Q^1 and *a* in set of alphabets Σ, Calculate $\delta^1(R,a) = \{q \in Q \mid q \in \delta(r,a) \text{ for some } r \in R\}$. Here, δ^1 performs the transition on *r* for some value of *a*.

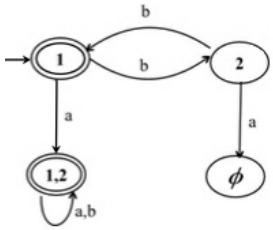
$$\begin{aligned}
\delta^1(\phi,a) &= \delta(\phi,a) \\
&= \phi \\
\delta^1(\phi,b) &= \delta^1(\phi,b) \\
&= \phi \\
\delta^1(\{1\},a) &= \delta(1,a) \\
&= \{1,2\} \\
\delta^1(\{1\},b) &= \delta(1,b) \\
&= \{2\} \\
\delta^1(\{2\},a) &= \delta(2,a) \\
&= \phi \\
\delta^1(\{2\},b) &= \delta(2,b) \\
&= \{1\} \\
\delta^1(\{1,2\},a) &= \delta(\{1,2\},a) \\
&= \delta(1,a) \cup \delta(2,a) \\
&= \{1,2\} \cup \phi \\
&= \{1,2\} \\
\delta^1(\{1,2\},b) &= \delta(\{1,2\},b) \\
&= \delta(1,b) \cup \delta(2,b) \\
&= \{2\} \cup \{1\} \\
&= \{1,2\}
\end{aligned}$$

3. $q_0' = \{q_0\}$ where q_0 is the start state in NFA.

Here, q_0'={1}.

4. $F' = \{R \in Q' \mid R \text{ contains an accept state of NFA}\}$. The machine M accepts the possible states where the NFA is present in the accept state.

5. The state diagram for the equivalent DFA is as follows:

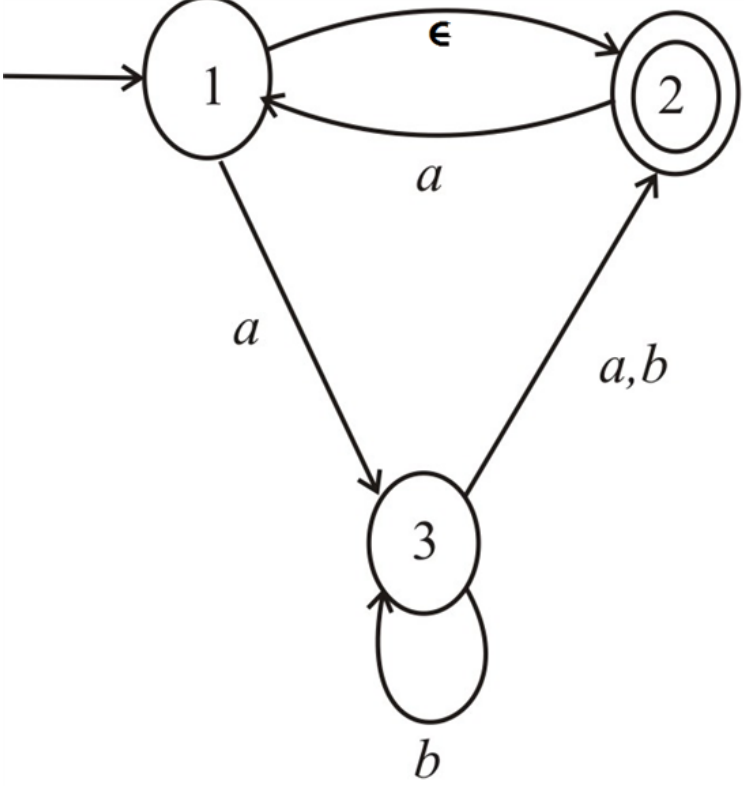


Comments (2)

Step 2 of 2

b.

Consider the Non-deterministic Finite Automata,



By using Theorem 1.39, "*For every non-deterministic finite automata, there is an equivalent Deterministic finite automation*".

Constructing equivalent DFA for the given NFA:

The initial state of DFA is 1 let x= (Q_x, $\sum$, δ_x, q_0, F_x).

1. $Q^1 = P(Q)$ where Q^1 is the subset of all sets of *Q*.

So, $Q^1 = \{\phi,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$

2. Considering $\in$ notations for each $R \subseteq Q$.

$$E(R) = \{q \mid q \text{ can be reached from } R \text{ by traveling along or more } \in \text{ arrows}\}$$

The collection of states reached from R by moving along the $\in$notations is,

$$E(\phi) = \phi$$
$$E(\{1\}) = \{1,2\}$$
$$E(\{2\}) = \{2\}$$
$$E(\{3\}) = \{3\}$$
$$E(\{1,2\}) = \{1,2\}$$
$$E(\{1,3\}) = \{1,2,3\}$$
$$E(\{2,3\}) = \{2,3\}$$
$$E(\{1,2,3\}) = \{1,2,3\}$$

3. Calculate $\delta^1(R,a) = \{q \in Q \mid q \in \delta(r,a) \text{ for some } r \in R\}$. Here, δ^1 performs the transition on *r* for some value of *a*.

$$\delta'(\phi,a) = \phi$$
$$\delta'(\phi,b) = \phi$$
$$\delta'(\{1\},a) = E(\delta(1,a))$$
$$= E(\{3\})$$
$$= \{3\}$$
$$\delta'(\{1\},b) = E(\delta(1,b))$$
$$= E(\phi)$$
$$= \phi$$
$$\delta'(\{2\},a) = E(\delta(1,a))$$
$$= E(\{1\})$$
$$= \{1,2\}$$
$$\delta'(\{2\},b) = E(\delta(2,b))$$
$$= E(\phi)$$
$$= \phi$$

$$\begin{aligned}\delta'(\{3\},a) &= E(\delta(3,a))\\ &= E(\{2\})\\ &= \{2\}\end{aligned}$$

$$\begin{aligned}\delta'(\{3\},b) &= E(\delta(3,b))\\ &= E(\{2,3\})\\ &= \{2,3\}\end{aligned}$$

$$\begin{aligned}\delta'(\{1,2\},a) &= E(\delta(1,a))\cup E(\delta(2,a))\\ &= E(\{3\})\cup E(\{1\})\\ &= \{3\}\cup\{1,2\}\\ &= \{1,2,3\}\end{aligned}$$

$$\begin{aligned}\delta'(\{1,2\},b) &= E(\delta(1,b))\cup E(\delta(2,b))\\ &= E(\{\phi\})\cup E(\{\phi\})\\ &= \phi\cup\phi\\ &= \phi\end{aligned}$$

$$\begin{aligned}\delta'(\{1,3\},a) &= E(\delta(1,a))\cup E(\delta(3,a))\\ &= E(\{3\})\cup E(\{2\})\\ &= \{3\}\cup\{2\}\\ &= \{3\}\cup\{2\}\\ &= \{2,3\}\end{aligned}$$

$$\begin{aligned}\delta'(\{1,3\},b) &= E(\delta(1,b))\cup E(\delta(3,b))\\ &= E(\{\phi\})\cup E(\{2,3\})\\ &= \phi\cup\{2,3\}\\ &= \{2,3\}\end{aligned}$$

$$\begin{aligned}\delta'(\{2,3\},a) &= E(\delta(2,a))\cup E(\delta(3,a))\\ &= E(\{1\})\cup E(\{2\})\\ &= \{1,2\}\cup\{2\}\\ &= \{1,2\}\end{aligned}$$

$$\begin{aligned}\delta'(\{2,3\},b) &= E(\delta(2,b))\cup E(\delta(3,b))\\ &= E(\phi)\cup E(\{2,3\})\\ &= \phi\cup\{2,3\}\\ &= \{2,3\}\end{aligned}$$

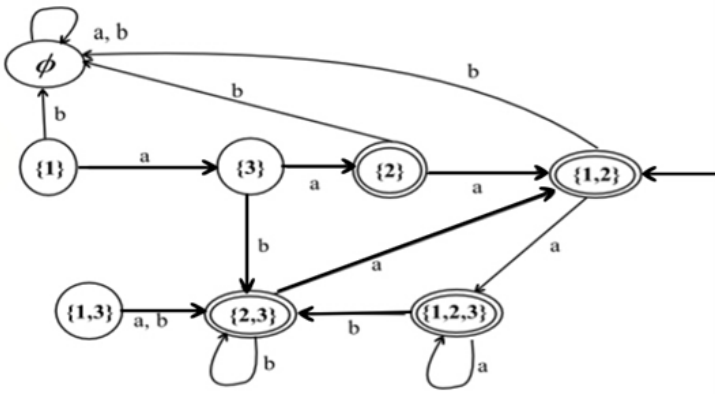
$$\begin{aligned}\delta'(\{1,2,3\},a) &= E(\delta(1,a))\cup E(\delta(2,a))\cup E(\delta(3,a))\\ &= E(\{3\})\cup E(\{1\})\cup E(\{2\})\\ &= \{3\}\cup\{1,2\}\cup\{2\}\\ &= \{1,2,3\}\end{aligned}$$

$$\begin{aligned}\delta'(\{1,2,3\},b) &= E(\delta(1,b))\cup E(\delta(2,b))\cup E(\delta(3,b))\\ &= E(\phi)\cup E(\phi)\cup E(\{2,3\})\\ &= \phi\cup\phi\cup\{2,3\}\\ &= \{2,3\}\end{aligned}$$

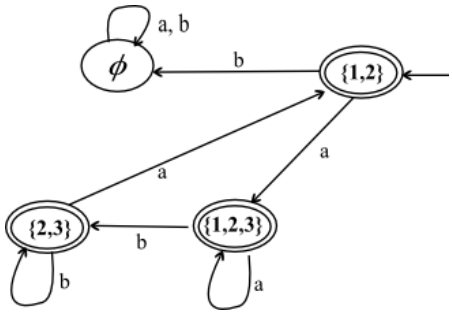
4. Changing q_0' to $E(q_0)$ the start state becomes,

$$\begin{aligned}q_0^1 &= E(q_0)\\ q_0^1 &= E(\{1\})\\ q_0^1 &= \{1,2\}\end{aligned}$$

5. The state diagram for the equivalent DFA is as follows:



Simplifying the machine by eliminating no arrow points. Here {1},{2},{1,3} and {3} do not contain any incoming arrows. Thus, the simplified machine is:



Comments (7)

Problem

a. Give an NFA recognizing the language $(01 \cup 001 \cup 010)^*$.

b. Convert this NFA to an equivalent DFA. Give only the portion of the DFA that is reachable from the start state.

Step-by-step solution

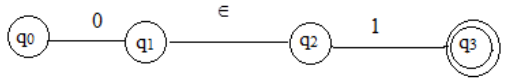
Step 1 of 2

(a) Given Language $L = (01 \cup 001 \cup 010)^*$

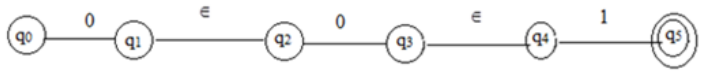
Assume that *M* as the NFA that recognizes language *L*.

The NFA *M* for the given language $L = (01 \cup 001 \cup 010)^*$ is as follows:

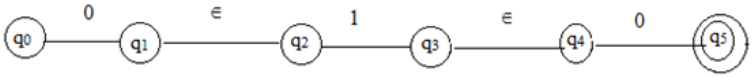
For string 01



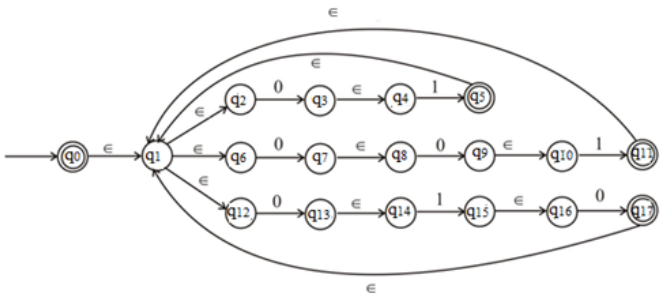
For string 001



For string 010



By joining all the above strings the final NFA for the $L = (01 \cup 001 \cup 010)^*$ is shown below:



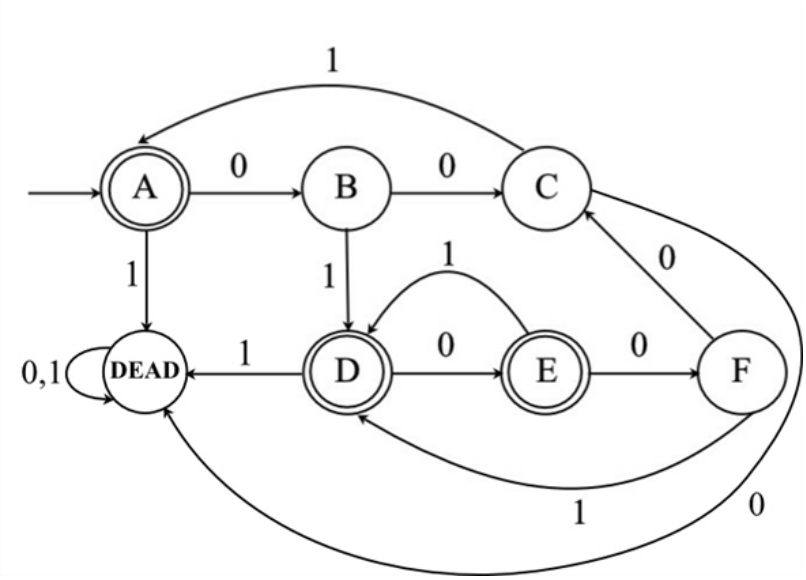
Comments (3)

Step 2 of 2

(b) Conversion of the NFA to DFA.

First remove all the $\in$ symbols in the NFA *M* and draw the transitions that are present in *M.*

The equivalent DFA for NFA for the Language $L = (01 \cup 001 \cup 010)^*$ is as follows:



Comments (2)

Problem

Give regular expressions generating the languages of Exercise 1.6.

Exercise 1.6.

Give state diagrams of DFAs recognizing the following languages. In all parts, the alphabet is {0,1}.

a. {wl w begins with a 1 and ends with a 0}

b. {wl w contains at least three 1s}

c. {wl w contains the substring 0101 (i.e., w = x0101y for some x and y)}

d. {wl w has length at least 3 and its third symbol is a 0}

e. {wl w starts with 0 and has odd length, or starts with 1 and has even length}

f. {wl w doesn't contain the substring 110}

g. {wl the length of w is at most 5}

h. {wl w is any string except 11 and 111}

i. {wl every odd position of w is a 1}

j. {wl w contains at least two 0s and at most one 1}

k. { ε , 0}

l. {wl w contains an even number of 0s, or contains exactly two 1s}

m. The empty set

n. All strings except the empty string

Step-by-step solution

Step 1 of 14

In the regular expressions, '*' indicates that the preceding regular expression may appear zero or more times and '+' indicates that the preceding regular expression may appear one or more times.

a.

Consider the language $L=\{w \mid w \text{ begins with a 1 and ends with a 0}\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

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

The strings accepted by the regular expression are $10,100,110,1010,1100,10100,...$

Therefore, the regular expression is $1(0+1)^*0$.

Comments (1)

Step 2 of 14

b.

Consider the language $L=\{w \mid w \text{ contains at least three 1s}\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

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

The strings accepted by the regular expression are $111,010101,01101,00001111,...$

Therefore, the regular expression is $(0+1)*1(0+1)*1(0+1)*1(0+1)*$.

Comments (4)

Step 3 of 14

c.

Consider the language $L=\{w \mid w \text{ contains the substring } 0101\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$$R=\Sigma*0101\Sigma*$$
$$=(0+1)*0101(0+1)*$$

The strings accepted by the regular expression are $0101,001011,101011,1101010,...$

Therefore, the regular expression is $(0+1)*0101(0+1)*$.

Comments (1)

Step 4 of 14

d.

Consider the language $L=\{w \mid w \text{ has length at least 3 and its third symbol is a } 0\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$$R=\Sigma\Sigma 0\Sigma*$$
$$=(0+1)(0+1)0(0+1)*$$

The strings accepted by the regular expression are $000,1101,...$

Therefore, the regular expression is $(0+1)(0+1)0(0+1)*$.

Comment

Step 5 of 14

e.

Consider the language,

$L=\{w \mid w \text{ starts with 0 and has odd length, or start with 1 and has even length}\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$$R=0(\Sigma\Sigma)*+1\Sigma(\Sigma\Sigma)*$$
$$=0((0+1)(0+1))*+1(0+1)((0+1)(0+1))*$$

The strings accepted by the regular expression are $0,011,010,00101,10,11,1001,...$

Therefore, the regular expression is $0((0+1)(0+1))*+1(0+1)((0+1)(0+1))*$.

Comment

Step 6 of 14

f.

Consider the language $L=\{w \mid w \text{ doesn't contain the substring } 110\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$$R=0*\left(10^{+}\right)*1*$$

The strings accepted by the regular expression are $010,011,0101,...$

Therefore, the regular expression is $0*\left(10^{+}\right)*1*$.

Comments (4)

Step 7 of 14

g.

Consider the language $L=\{w \mid \text{the length of } w \text{ is at most } 5\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$$\begin{aligned} R &= \varepsilon+\Sigma+\Sigma\Sigma+\Sigma\Sigma\Sigma+\Sigma\Sigma\Sigma\Sigma+\Sigma\Sigma\Sigma\Sigma\Sigma \\ &= \varepsilon+(0+1)+(0+1)(0+1)+(0+1)(0+1)(0+1)+(0+1)(0+1)(0+1)(0+1)+(0+1)(0+1)(0+1)(0+1)(0+1) \\ &= \varepsilon+(0+1)+(0+1)^2+(0+1)^3+(0+1)^4+(0+1)^5 \end{aligned}$$

The strings accepted by the regular expression are $\varepsilon,0,01,101,1010,00000,...$ The empty string is of length 0. The language accepts the strings of length from 0 to 5.

Therefore, the regular expression is $\varepsilon+(0+1)+(0+1)^2+(0+1)^3+(0+1)^4+(0+1)^5$.

Comments (1)

Step 8 of 14

h.

Consider the language $L=\{w \mid w \text{is any string except } 11 \text{ and } 111\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$$\begin{aligned} R &= \varepsilon+\Sigma+0\Sigma+10+0\Sigma\Sigma+10\Sigma+110+\Sigma^3\Sigma^+ \\ &= \varepsilon+(0+1)+0(0+1)+10+0(0+1)(0+1)+10(0+1)+110+(0+1)^3(0+1)^+ \end{aligned}$$

The strings accepted by the regular expression are $\varepsilon,101,110,1010,...$

Therefore, the regular expression is,

$\varepsilon+(0+1)+0(0+1)+10+0(0+1)(0+1)+10(0+1)+110+(0+1)^3(0+1)^+$.

Comments (4)

Step 9 of 14

i.

Consider the language $L=\{w \mid \text{every odd position of } w \text{ is a } 1\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

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

The strings accepted by the regular expression are $\varepsilon,101,111,1010,...$

Therefore, the regular expression is $(1(0+1))^*(\varepsilon+1)$.

Comments (3)

Step 10 of 14

j.

Consider the language $L=\{w \mid w \text{ contains at least two 0s and at most one } 1\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$$R=00^*00^*(\varepsilon+1)+00^*(\varepsilon+1)00^*+(\varepsilon+1)00^*00^*$$

The strings accepted by the regular expression are $001,010,100,...$ In the first part of the regular expression $00^*00^*(\varepsilon+1)$, there are two mandatory zeros and at most one 1. The optional 1 may appear at the start or middle or at the end. There are three parts in the regular expression to accept such strings.

Therefore, the regular expression is $00^*00^*(\varepsilon+1)+00^*(\varepsilon+1)00^*+(\varepsilon+1)00^*00^*$.

Comments (1)

Step 11 of 14

k.

Consider the language $L=\{\varepsilon,0\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$R=0+\varepsilon$

Therefore, the regular expression is $0+\varepsilon$ **.**

Comment

Step 12 of 14

l.

Consider the language,

$L=\{w \mid w \text{ contains an even number of 0s, or contains exactly two 1s}\}$ over the alphabet $\Sigma=\{0,1\}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$R=1*(01*01*)*+0*10*10*$

The strings accepted by the regular expression are $\varepsilon,00,11,0101,010100,...$

Therefore, the regular expression is $1*(01*01*)*+0*10*10*$ **.**

Comments (3)

Step 13 of 14

m.

Consider the language $L=\text{The empty set}$. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$R=\phi$

Therefore, the regular expression is ϕ **.**

Comment

Step 14 of 14

n.

Consider the language *L* accepts all the strings except the empty string. Let *R* be the regular expression that generates the language *L*. The regular expression is as follows:

$$R=\Sigma^+$$
$$=(0+1)^+$$

The language accepts all the strings except ε.

Therefore, the regular expression is $(0+1)^+$ **.**

Comment

Problem

Use the procedure described in Lemma 1.55 to convert the following regular expressions to nondeterministic finite automata.

a. $(0 \cup 1)^* 000 (0 \cup 1)^*$

b. $(((00)^*(11)) \cup 01)^*$

c. $\emptyset^*$

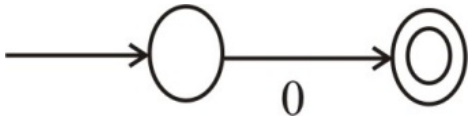
Step-by-step solution

Step 1 of 4

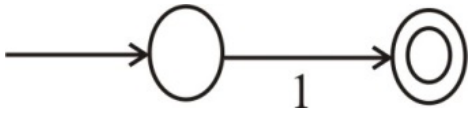
a. Consider the regular expression $R = (0 \cup 1)^* 000 (0 \cup 1)^*$.

Now, construct an NFA from this regular expression in the following procedure:

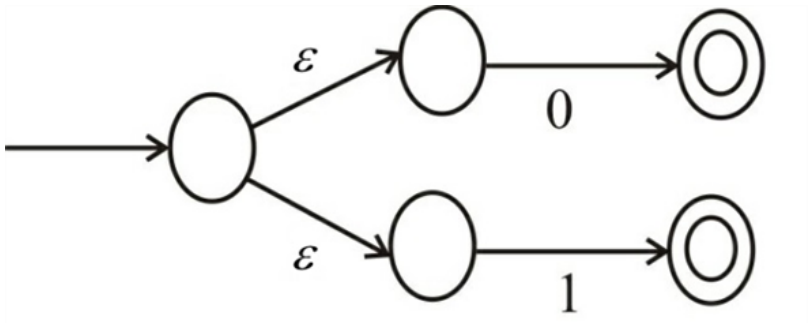
0



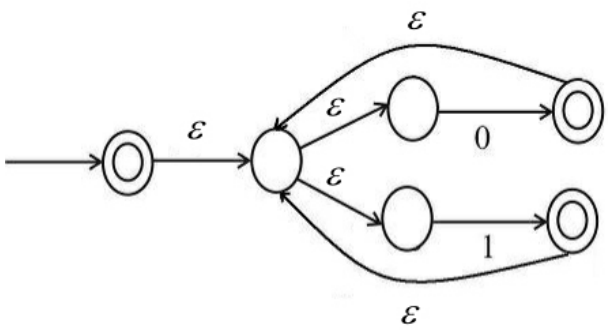
1



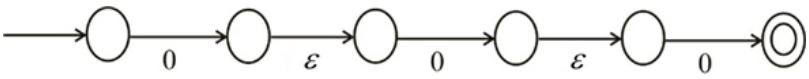
$0 \cup 1$



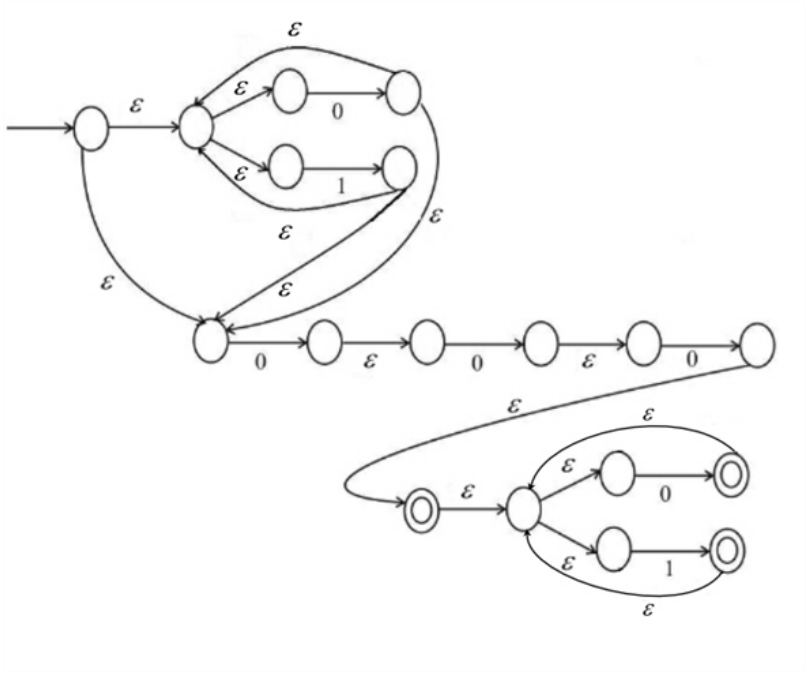
$(0 \cup 1)^*$



000



$(0\cup1)*000(0\cup1)*$



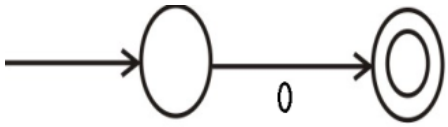
Comments (12)

Step 2 of 4

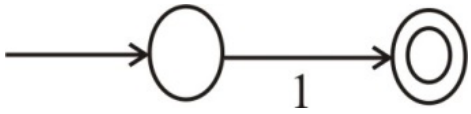
b. Consider the regular expression $R=\left(((00)*(11))\cup01\right)*$.

Now, construct the NFA from this regular expression in the following procedure:

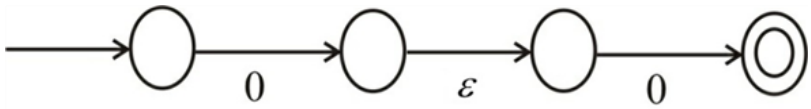
0



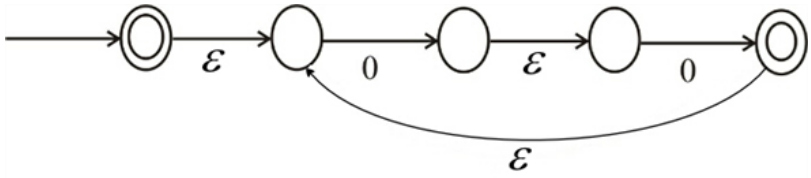
1



00



$(00)*$

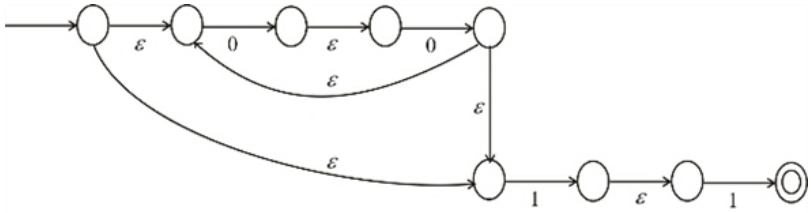


Comment

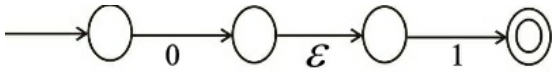
Step 3 of 4

11

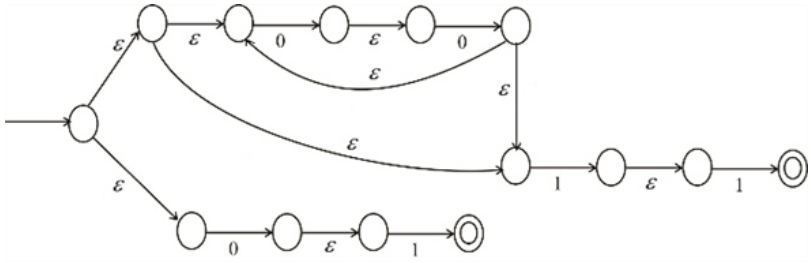
$(00)^*11$



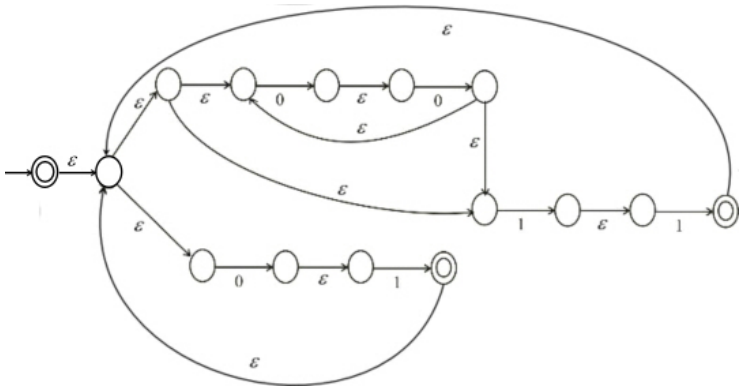
01



$(((00)^*(11))\cup(01))$



$(((00)^*(11))\cup(01))^*$



Comments (2)

Step 4 of 4

c. Consider the regular expression $R = \phi^*$.

The closure of an empty set is an empty string i.e., $\phi^* = \{\varepsilon\}$. The NFA for the regular expression is as follows:

Comment

Problem

For each of the following languages, give two strings that are members and two strings that are *not* members—a total of four strings for each part. Assume the alphabet $\sum = \{a,b\}$ in all parts.

a. a^*b^*

b. $a(ba)^*b$

c. $a^* \cup b^*$

d. $(aaa)^*$

e. $\sum^* a \sum^* b \sum^* a \sum^*$

f. $aba \cup bab$

g. $(\varepsilon \cup a)b$

h. $(a \cup ba \cup bb)\sum^*$

Step-by-step solution

Step 1 of 8

(a) Language is $L = a^*b^*$ over the alphabet $\Sigma = \{a,b\}$

• Strings that are member of *L*

(i) ab

(ii) abb

• Strings that are not members of *L*

(i) ba

(ii) bba

Comments (3)

Step 2 of 8

(b) Language is $L = a(ba)^*b$ over $\Sigma = \{a,b\}$

• Strings that are members of *L*

(i) $abab$

(ii) $ababab$

• Strings that are not members of *L*

(i) aba

(ii) bab

Comments (4)

Step 3 of 8

(c) Given language is $L = a^* \cup b^*$ over $\Sigma = \{a,b\}$

• Strings that are members of *L*

(i) aaa

(ii) bbb

• Strings that are not members of *L*

(i) $baab$

(ii) $bbaa$

Comments (4)

Step 4 of 8

(d) Given language is $L=(aaa)^*$ over alphabet $\Sigma=\{a,b\}$

• Strings that are members of *L*

(i) aaa

(ii) $aaaaaa$

• Strings that not members of *L*

(i) a

(ii) $aaaaa$

Comments (3)

Step 5 of 8

(e) Given language is $L=\Sigma^*a\,\Sigma^*b\Sigma^*a\Sigma^*$ over $\Sigma=\{a,b\}$

• Strings that are members of *L*

(i) aba

(ii) $aabbaa$

• Strings that over not members of L

(i) a

(ii) b

Comments (1)

Step 6 of 8

(f) Given language is $L=aba\cup bab$ over $\Sigma=\{a,b\}$

• Strings that are members of *L*

(i) aba

(ii) bab

• Strings that over not members of L

(i) abb

(ii) ba

Comments (5)

Step 7 of 8

(g) Given language is $L=(\in\cup a)b$ over $\Sigma=a\{a,b\}$

• Strings that are members of *L*

(i) b

(ii) ab

• Strings that are not members of *L*

(i) a

(ii) ba

Comments (1)

Step 8 of 8

(h) Given language is $L = (a \cup ba \cup bb)\Sigma^*$ over $\Sigma = \{a, b\}$

• Strings that are members of *L*

(i) a

(ii) $bbab$

• Strings that are not members of *L*

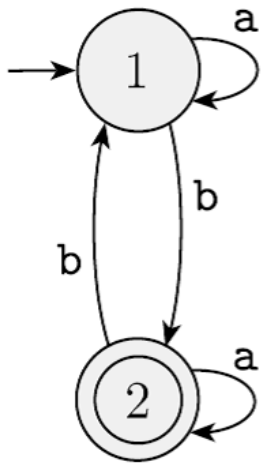
(i) b

(ii) $\in$

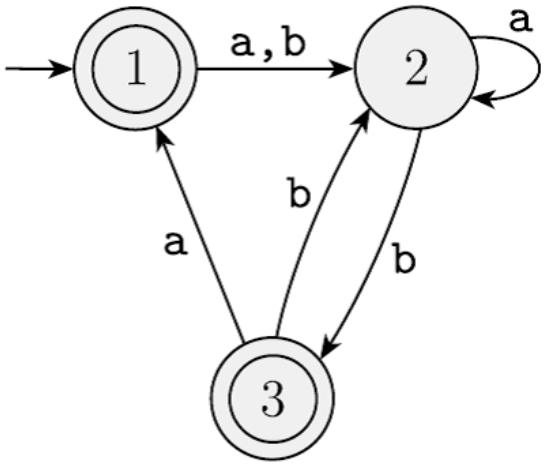
Comments (4)

Problem

Use the procedure described in Lemma 1.60 to convert the following finite automata to regular expressions.



(a)



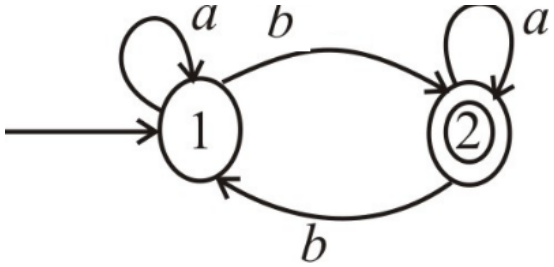
(b)

Step-by-step solution

Step 1 of 14

(a)

Consider the Finite Automata:



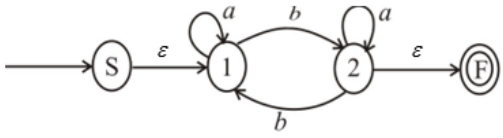
Now convert this finite automaton to a regular expression in the following steps as below:

Comment

Step 2 of 14

Step 1:

Add the start state(S) and new accept state (F) to make the original accept state as non – accepting state as:

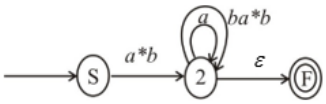


Comment

Step 3 of 14

Step 2:

In the second step eliminate the state(1), no need to add the loop for the state 1 and directly add loop to the state (2) and write the expression by passing state (S) to state (2)

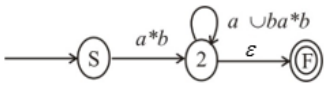


Comment

Step 4 of 14

Step 3:

From the above step one loop is represented a s union with the a as follows:

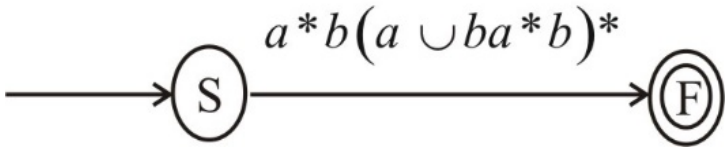


Comment

Step 5 of 14

Step 4:

Now remove the loop over the state (2) and eliminate it and write expression directly from state (S) to State(F)



Comment

Step 6 of 14

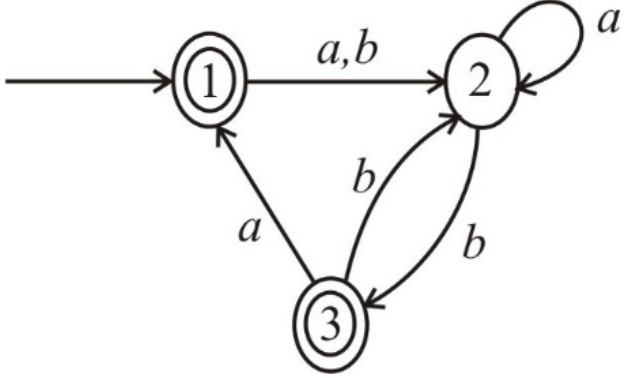
So, the regular expression for the given finite automata is $\boxed{a*b(a \cup ba*b)*}$

Comment

Step 7 of 14

(b)

Consider the second finite automata is



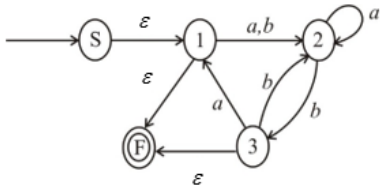
Now convert this finite automaton to a regular expression in the following steps as below:

Comment

Step 8 of 14

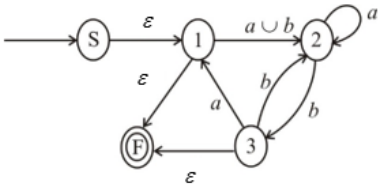
Step 1:

Add new start state (S) and new accept state (F). Make original accept states as non-accepting states then the Finite Automata becomes:



Step 2:

Perform union on the edge from state 1 to state2.

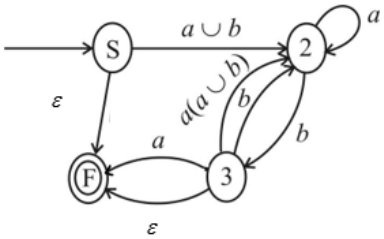


Comment

Step 9 of 14

Step 3:

From the above step 2, there are no unions or loops for the state 1, So eliminate the state 1 as follows:

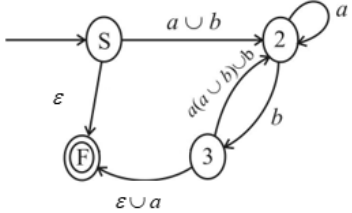


Comments (4)

Step 10 of 14

Step 4:

Perform unions on edges from state 3 to state 2 and from state 3 to the final state, Then the Automata becomes as below:

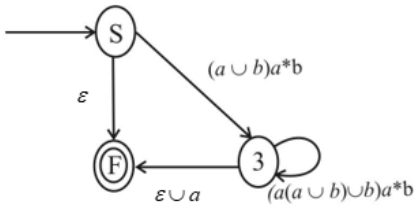


Comment

Step 11 of 14

Step 5:

Here to minimize the automata eliminate 2 and perform union on 3 and write expression for the state (S) to state(3), then apply loop on state (3) with the expression of state(2)

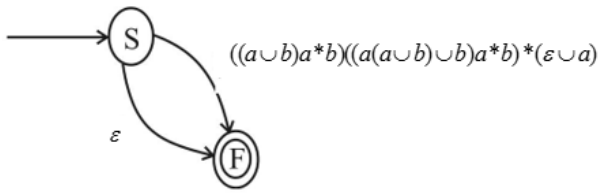


Comment

Step 12 of 14

Step 6:

Eliminate the state 3 and write the expression from state(S) to state (F), because there are no loops and unions.

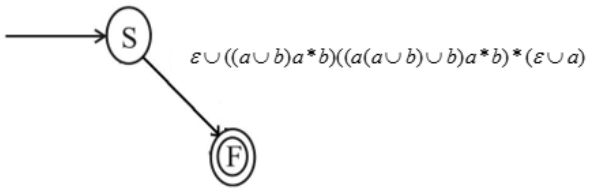


Comment

Step 13 of 14

Step 7:

Perform union on edge from state *S* to state *F*



Comment

Step 14 of 14

So, the regular expression for the given finite automata is

$$\varepsilon \cup ((a \cup b)a^*b)((a(a \cup b) \cup b)a^*b)^*(\varepsilon \cup a)$$

Comment

Problem

In certain programming languages, comments appear between delimiters such as /# and #/. Let C be the language of all valid delimited comment strings. A member of C must begin with /# and end with #/ but have no intervening #/. For simplicity, assume that the alphabet for C is ∑ = {a, b, /, #}.

a. Give a DFA that recognizes C.

b. Give a regular expression that generates C.

Step-by-step solution

Step 1 of 2

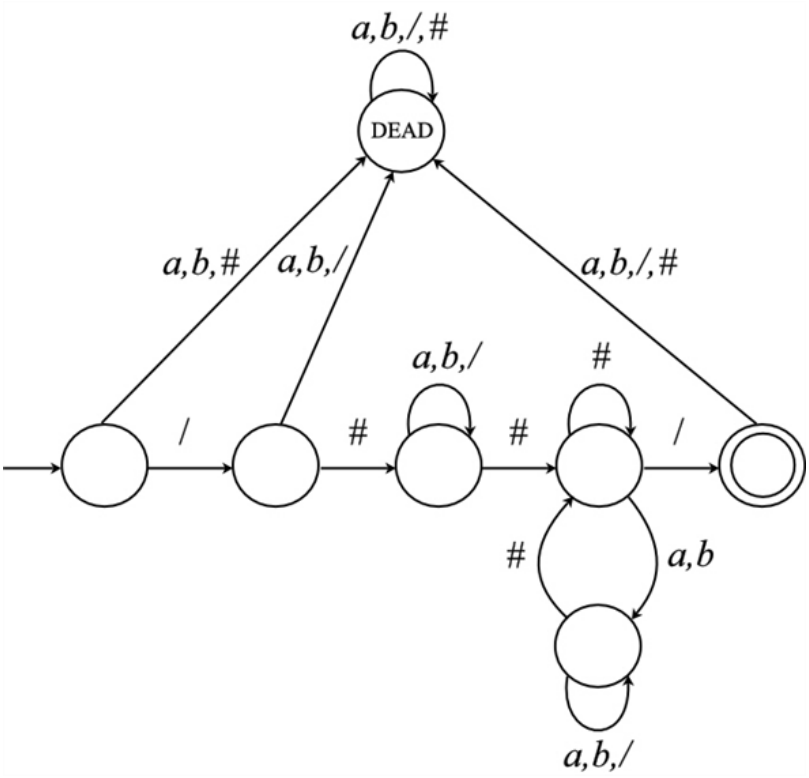
(a) Consider the language C of all valid delimited commented strings

• A member of C must begin with '/#' and end with '#/' but have no intervening '#/'.

• The alphabet of *C is* $\Sigma = \{a, b, /, \#\}$

Let *M* be the DFA that recognize the language C.

The state diagram of *M* is as follows



So this is the DFA that recognizes the languages of C.

Comments (2)

Step 2 of 2

(b) The regular expression that generates language C is,

$/\#\left(a+b+/\right)*\#(\#+(a+b)(a+b+/)*\#)*/$

Comments (3)

Problem

Let B be any language over the alphabet Σ. Prove that $B = B^+$ iff $BB \subseteq B$.

Step-by-step solution

Step 1 of 3

Let B be any language over the alphabet Σ.

To Prove: $B = B^+$ iff $BB \subseteq B$

The requirement is to prove both the directions of iff.

Comment

Step 2 of 3

One direction:

Assume: $B = B^+$ .....(1)

To show: $BB \subseteq B$

Since, for every language $BB \subseteq B^+$ .....(2)

By substituting (1) in (2), it can be obtained that $BB \subseteq B$

Hence, it has been proved that: $BB \subseteq B$ iff $B = B^+$

Comment

Step 3 of 3

Other direction:

Assume: $BB \subseteq B$

To prove: $B = B^+$

It is known that for every language $BB \subseteq B^+$ .....(3).

Let *w* be a string of elements, such that $w \in B^+$ then $w \in B$.

• If $w \in B^+$ then the string *w* can be split into elements $x_1, x_2, x_3, ...x_k$ such that $w = x_1, x_2, ...x_k$ for $x_i \in B$ and $k \geq 1$. Since, $x_1, x_2 \in B$ and it is assumed that $BB \in B$.

• Similarly, it holds for the element x_3, such that $x_3 \in B$ and $BB \subseteq B$ .Thus, $x_1, x_2, x_3 \in B$.

• If this procedure is continued, then $x_1, x_2, x_3, ...x_k \in B$ that is $w \in B$.

• It means that, if all the elements of the string ($x_1, x_2, ...x_k$) belong to B, then the strin $w \in B$.

Thus, $B^+ \subseteq B$ .....(4)

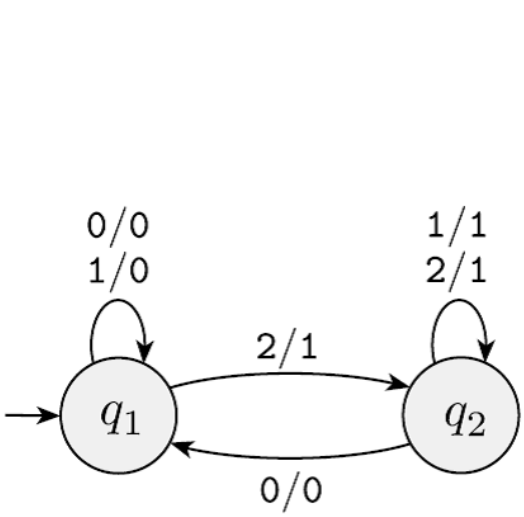
From, (3) and (4) it can be concluded that $B = B^+$.

Hence, it has been proved for both the directions that, for any language B, $B = B^+$ iff $BB \subseteq B$.

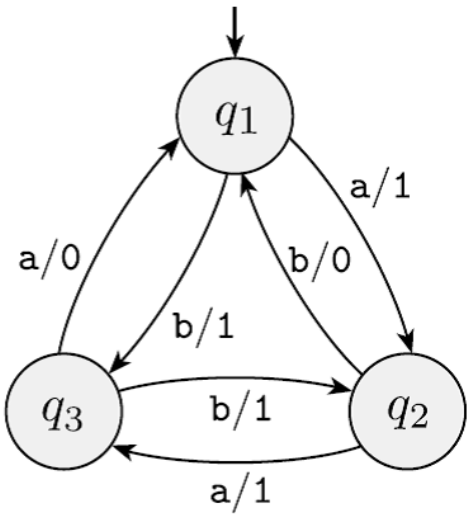
Comments (1)

Problem

A ***finite state transducer*** (FST) is a type of deterministic finite automaton whose output is a string and not just accept or reject . The following are state diagrams of finite state transducers T_1 and T_2.



T_1



T_2

Each transition of an FST is labeled with two symbols, one designating the input symbol for that transition and the other designating the output symbol. The two symbols are written with a slash, /, separating them. In T_1, the transition from q_1 to q_2 has input symbol 2 and output symbol 1. Some transitions may have multiple input–output pairs, such as the transition in T_1 from q_1 to itself. When an FST computes on an input string w, it takes the input symbols $w_1 \cdots w_n$ one by one and, starting at the start state, follows the transitions by matching the input labels with the sequence of symbols $w_1 \cdots w_n = w$. Every time it goes along a transition, it outputs the corresponding output symbol. For example, on input 2212011, machine T_1 enters the sequence of states $q_1, q_2, q_2, q_2, q_2, q_1, q_1, q_1$ and produces output 1111000. On input abbb, T_2 outputs 1011. Give the sequence of states entered and the output produced in each of the following parts.

a. T_1 on input 011

b. T_1 on input 211

c. T_1 on input 121

d. T_1 on input 0202

e. T_2 on input b

f. T_2 on input bbab

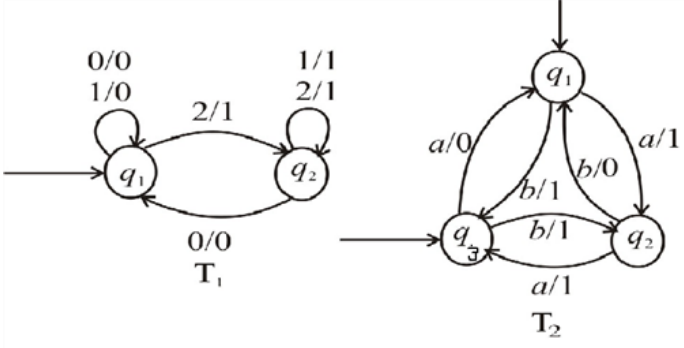
g. T_2 on input bbbbbb

h. T_2 on input ε

Step-by-step solution

Step 1 of 9

Given state diagram of finite state transducers (FST) T_1 and T_2 are as follows



Comment

Step 2 of 9

(a) Input 011

FST T_1 will enter the following sequence of states on input 011: q_1, q_1, q_1, q_1 and produces the output: 000

Comment

Step 3 of 9

(b) Input 211

FST T_1 will enter the following sequence of state on input 211: q_1, q_2, q_2, q_2 and produces the output: 111

Comment

Step 4 of 9

(c) Input 121

FST T_1 will enter the following sequence of states on input 121: q_1, q_1, q_2, q_2 and produces the output 011

Comment

Step 5 of 9

(d) Input 0202

FST T_1 will enter the following sequence of states on input 0202: q_1, q_1, q_2, q_1, q_2

and produces the output: 0101

Comment

Step 6 of 9

(e) Input *b*

FST T_2 will enter the following sequence of states on input *b*: q_1, q_3 and produces the output: 1

Comment

Step 7 of 9

(f) Input $bbab$

FST T_2 will enter the following sequence of states on input $bbab$: q_1, q_3, q_2, q_3, q_2

and produces the output 1111

Comment

Step 8 of 9

(g) Input $bbbbbb$

FST T_2 will enter the following sequene of states on input $bbbbbb$:

$q_1, q_3, q_2, q_1, q_3, q_2, q_1$ and produces the output: 110110

Comment

Step 9 of 9

(h) Input $\in$

FST T_2 will enter the following sequence of states on input $\in$: q_1

and produces the following output $\in$

Comment

Problem

Read the informal definition of the finite state transducer given in Exercise 1.24. Give a formal definition of this model, following the pattern in Definition 1.5 (page 35). Assume that an FST has an input alphabet ∑ and an output alphabet $\prod$ but not a set of accept states. Include a formal definition of the computation of an FST.

(Hint: An FST is a 5-tuple. Its transition function is of the form $\delta: Q \times \Sigma \longrightarrow Q \times \Gamma.)$

Step-by-step solution

Step 1 of 2

Finite State Transducer:

• A finite state transducer is a kind of deterministic finite state automaton which consists of both the input string and the output string.

• It converts the input string into an output string.

Comment

Step 2 of 2

The formal definition of Finite State Transducer (FST) is as follows:

A Finite State Transducer is a 6-tuple machine and is represented as $M = (Q, \Sigma, \Gamma, \delta, q_0)$ where

- Q is a finite set of states.
- Σ is a finite set of input alphabets.
- Γ is a finite set of output alphabets.
- $\delta : Q \times \Sigma \to Q \times \Gamma$ is the transition function that defines rules.
- $q_0 \in Q$, is the start state.

The formal definition of the computation of Finite State Transducer (FST) is as follows:

• The computation of the finite state machine is carried out by translating the input string into output string.

• Assume that *w* is an input string over the input alphabet Σ and *x* is an output string consisting of alphabet of Γ.

• The transition is carried out over a sequence of states $q_0', q_1', \ldots . q_n'$ in Q' such that

- $q_0' = q_0$

• The transition of $\delta(q_{i+1}', w_{i+1}) = (q_{i+1}', x_{i+1})$ for $0 \le i < n$.

Comments (1)

Problem

Using the solution you gave to Exercise 1.25, give a formal description of the machines T_1 and T_2 depicted in Exercise 1.24

Step-by-step solution

Step 1 of 3

Here, the formal description of Turing machine T_1 and T_2 need to be defined. A finite state transducer (FST) is formally defined by the $(Q,\Sigma,\Gamma,\delta,q_0)$ tuple, where:

• The finite set of states is Q.

• The input alphabet is Σ.

• The output alphabet is Γ.

• The transition function δ takes a state and an input symbol and returns a state and an output symbol.

$\delta: Q\times\Sigma\to Q\times\Gamma$

• The start state q_0.

Comment

Step 2 of 3

The finite state transducer T_1 is formally defined by the $(\{q_1,q_2\},\{0,1,2\},\{0,1\},\delta_1,q_1)$, where the transition function δ_1 is as follows:

Input / *State*	0	1	2
q_1	$\{q_1,0\}$	$\{q_1,0\}$	$\{q_2,1\}$
q_2	$\{q_1,0\}$	$\{q_2,1\}$	$\{q_2,1\}$

Comment

Step 3 of 3

The second FST is defined as $T_2=(\{q_1,q_2,q_3\},\{a,b\},\{0,1\},\delta_2,q_1)$. The transition function δ_2 is given by:

Input / *State*	a	b
q_1	$\{q_2,1\}$	$\{q_3,1\}$
q_2	$\{q_3,1\}$	$\{q_1,0\}$
q_3	$\{q_1,0\}$	$\{q_2,1\}$

Comment

Problem

Read the informal definition of the finite state transducer given in Exercise 1.24. Give the state diagram of an FST with the following behavior. Its input and output alphabets are {0,1}. Its output string is identical to the input string on the even positions but inverted on the odd positions. For example, on input 0000111 it should output 1010010.

Step-by-step solution

Step 1 of 2

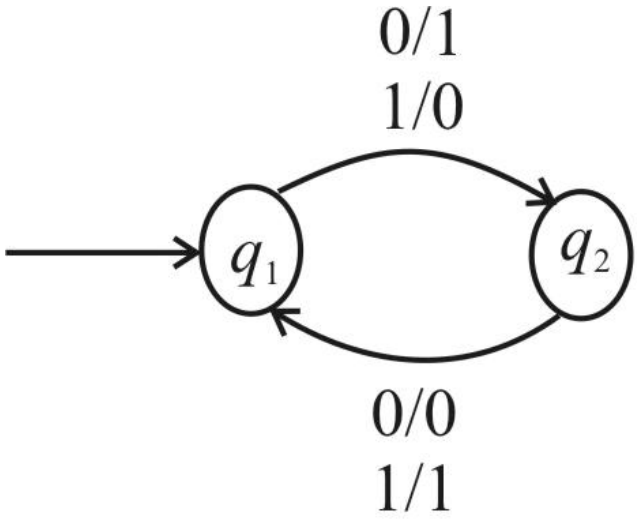
Given input and output alphabets for FST are $\{0,1\}$.

FST has the following behavior

• Its output string is identical to the input string on the even positions.

• Its output string is inverted on the odd positions.

So the state diagram of that FST is given as follows:



Comment

Step 2 of 2

So by this state diagram,

• In even positions output string is same as input string

i.e. for input $0 \rightarrow$ output is also 0

for input $1 \rightarrow$ output is also 1

• In odd positions, out string is inverted

i.e. for input $0 \rightarrow$ output is 1

for input $1 \rightarrow$ output is 0.

Comment

Problem

Convert the following regular expressions to NFAs using the procedure given in Theorem 1.54. In all parts, ∑ = {a, b}.

a. $a(abb)^* \cup b$

b. $a^+ \cup (ab)^+$

c. $(a \cup b^+)a^+b^+$

THEOREM 1.54

A language is regular if and only if some regular expression describes it.

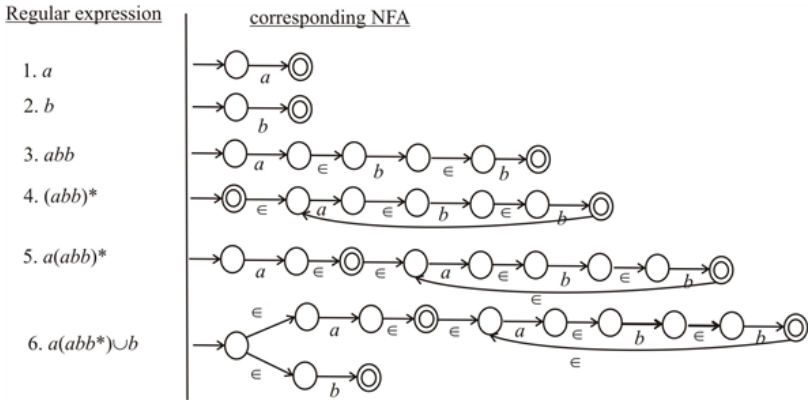
Step-by-step solution

Step 1 of 3

(a) Given regular expression

$R = a(abb)^* \cup b$ over $\Sigma = \{a, b\}$.

Now we have to convert this regular expression into NFA by the following steps.



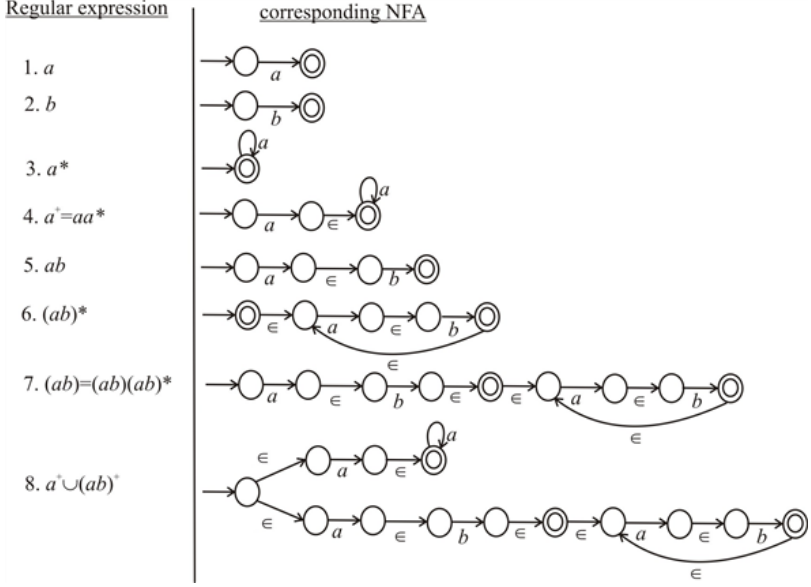
Comments (4)

Step 2 of 3

(b) Given regular expression is

$R = a^+ \cup (ab)^+$ over $\Sigma = \{a, b\}$

Now we have to convert this regular expression into NFA by the following steps.

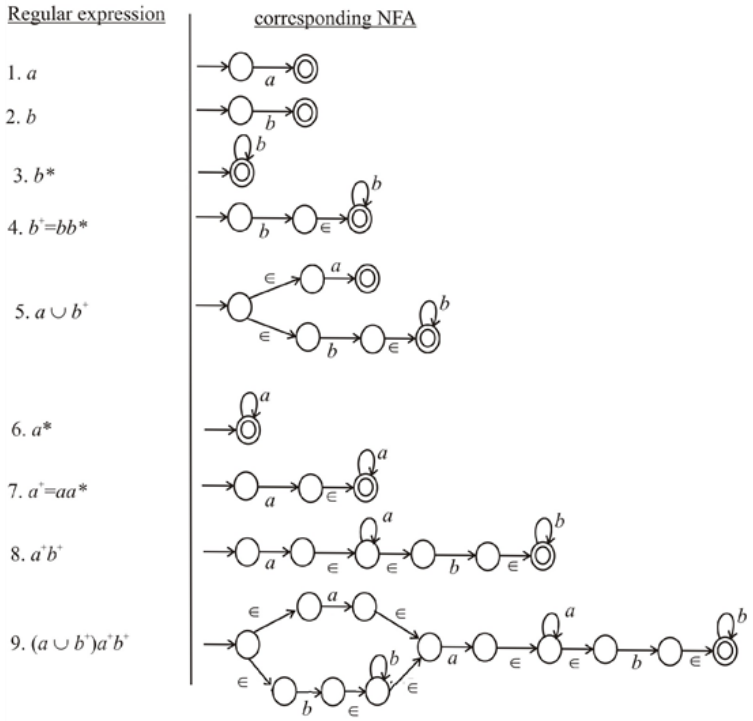


Comment

Step 3 of 3

(c) Given regular expression is $R=\left(a\cup b^{+}\right)a^{+}b^{+}$ over $\Sigma=\{a,b\}$.

Now we have to convert this regular expression R into NFA by the following steps.



Comments (2)

Problem

Use the pumping lemma to show that the following languages are not regular.

a. $A_1 = \{0^n 1^n 2^n \mid n \geq 0\}$

b. $A_2 = \{www \mid w \in \{a, b\}^*\}$

c. $A_3 = \{a^{2^n} \mid n \geq 0\}$ (Here, a^{2^n} means a string of 2^n a's.)

Step-by-step solution

Step 1 of 4

Pumping Lemma:

If A is regular language, there is a number p (the pumping length) where *S* is any string in A of length at least *P*, then *S* may be divided into three pieces, $S = xyz$, satisfying the following conditions.

1. For each $i \geq 0, xy^i z \in A$
2. $|y| > 0$, and
3. $|xy| \leq p$

Comment

Step 2 of 4

(a)

Consider the language, $A_1 = \{0^n 1^n 2^n \mid n \geq 0\}$.

Assume A_1 is a regular language.

Let p be the pumping length given by the pumping lemma consider a string $S = 0^p 1^p 2^p \in A_1$

$|S| > P$ so, by pumping lemma, take $S = 0^p 1^p 2^p = xyz$ such that $|xy| \leq p, |y| > 0$ consider the following 2 possibilities:

Let 001122 be the string that belongs to A_1. $S = 0^p 1^p 2^p = 001122$. The pumping length of the string is 2. To satisfy the conditions of the pumping lemma, x = 0, y = 0, z = 1122.

$$S = 001122$$
$$= \underset{x}{0}\ \underset{y}{0}\ \underset{z}{1122}$$

Pump the middle part such that $xy^i z\ (i \geq 0)$. For *i*=2, the *y* becomes 00. The string after pumping is 0001122.

$$S = (0)\ (0)^i\ (1122)$$
$$= \underset{x}{0}\ \underset{y}{00}\ \underset{z}{1122} \quad [when\ i = 2]$$

The string $0001122 \notin A_1$ because the string that is accepted by the language should have equal number of 0's, 1's and 2's. It is a contradiction. So, the pumping lemma is violated.

Therefore, A_1 is not a regular language.

Comment

Step 3 of 4

(b)

Consider the language, $A_2 = \{www \mid w \in \{a,b\}^*\}$.

Assume A_2 is a regular language.

Let p be the pumping length given by the pumping lemma.

Consider a string $S = a^p b a^p b a^p b \in A_2$.

By pumping lemma, this string can be divided into three pieces xyz such that $|xy| \le p, |y| > 0$ and $xy^i z \in A_2 \forall i \ge 0$

So $S = a^p b a^p b a^p b = xyz$.

Let *aabaabaab* be the string that belongs to A_2. The pumping length of the string is 2. To satisfy the conditions of the pumping lemma, x = *a*, y = *a*, z = *baabaab*.

$$S = aabaabaab$$
$$= \underset{x}{a}\ \underset{y}{a}\ \underset{z}{baabaab}$$

Pump the middle part such that $xy^i z \ (i \ge 0)$. For *i*=2, the *y* becomes *aa*. The string after pumping is *aaabaabaab*.

$$S = (a)\ (a)^i\ (baabaab)$$
$$= \underset{x}{a}\ \underset{y}{aa}\ \underset{z}{baabaab} \quad [when\ i = 2]$$

The string $aaabaabaab \notin A_2$.It is a contradiction. So, the pumping lemma is violated.

Therefore, A_2 is not a regular language.

Comments (8)

Step 4 of 4

(c)

Consider the language, $A_3 = \{a^{2^n} \mid n \ge 0\}$ (Here, a^{2^n} means a string of 2^n a's).

Assume that A_3 is regular language.

Let p be the pumping length given by pumping lemma consider a string $S = a^{2^p} \in A_3$. And $|S| > p$

By pumping lemma, this string can be divided into three pieces xyz such that $|xy| \le p, |y| > 0$ and $xy^i z \in A_2 \forall i \ge 0$

Let *aaaa* be the string that belongs to A_3. The pumping length of the string is 2. To satisfy the conditions of the pumping lemma, x = *a*, y = *a*, z = *aa*.

$$S = aaaa$$
$$= \underset{x}{a}\ \underset{y}{a}\ \underset{z}{aa}$$

Pump the middle part such that $xy^i z \ (i \ge 0)$. For *i*=2, the *y* becomes *aa*. The string after pumping is *aaaaa*.

$$S = (a)\ (a)^i\ (aa)$$
$$= \underset{x}{a}\ \underset{y}{aa}\ \underset{z}{aa} \quad [when\ i = 2]$$

The string $aaaaa \notin A_3$.It is a contradiction. So, the pumping lemma is violated.

Therefore, A_3 is not a regular language.

Comments (9)

Problem

Describe the error in the following “proof” that 0^*1^* is not a regular language. (An error must exist because 0^*1^* *is* regular.) The proof is by contradiction. Assume that 0^*1^* is regular. Let p be the pumping length for 0^*1^* given by the pumping lemma. Choose s to be the string 0^p1^p. You know that s is a member of 0^*1^*, but Example 1.73 shows that s cannot be pumped. Thus you have a contradiction. So 0^*1^* is not regular.

Step-by-step solution

Step 1 of 4

The pumping lemma is used as a negative test to prove that the given language is non-regular. The language that violates the any of the three conditions of the pumping lemma is classified as non-regular.

Since the pumping lemma starts by assuming that the given language is regular, the belongingness of the string, which is used as a counter example, is tested only for the given language and not for all the languages that accepts the string.

Comment

Step 2 of 4

The proof given in the example 1.73 (refer to the textbook example 1.73) is for a different language. The counter examples given to prove the language as non-regular does not hold true for the language given in the question as shown below:

Let A be the language $\{0^*1^*\}$ given in the question.

Let B be the language $\{0^n1^n \mid n>=0\}$ given in the example 1.73.

As per the example 1.73, s is 0^p1^p and hence as per the pumping lemma, s should be split into three pieces as s = xyz, where for any i>=0, the string xy^iz is in B. The cases are as follows:

• The string y contains all 0s: The string xyyz will result in more number of 0s than the number of 1s which does not belong to the language B but the string belongs to language A and hence, the pumping lemma is not violated.

• The same reason holds true for the case when the string y contains all 1s. The string xyyz will results in more number of 1s which is again accept by the given language A.

Comment

Step 3 of 4

Since the above conditions are satisfied, the given language cannot be classified as non-regular by pumping lemma.

Comment

Step 4 of 4

Therefore, the error in the given proof is that a string which is used as a counter example to prove a certain language as non-regular does not signifies that all the languages that accept that string are considered as non-regular.

The above problems arise when the pumping lemma is used on a language which is regular since violating a condition of the pumping lemma indicates that the language is non-regular but the vice versa is not always true.

Comment

Problem

For any string w = $w_1w_2 \cdots w_n$, the ***reverse*** of w, written w^R, is the string w in reverse order, $w_n \cdots w_2w_1$. For any language A, let $A^R = \{w^R | w \in A\}$. Show that if A is regular, so is A^R.

Step-by-step solution

Step 1 of 3

Let $M = (Q, \Sigma, \delta, q_0, F)$ be the DFA that recognizes *A*,

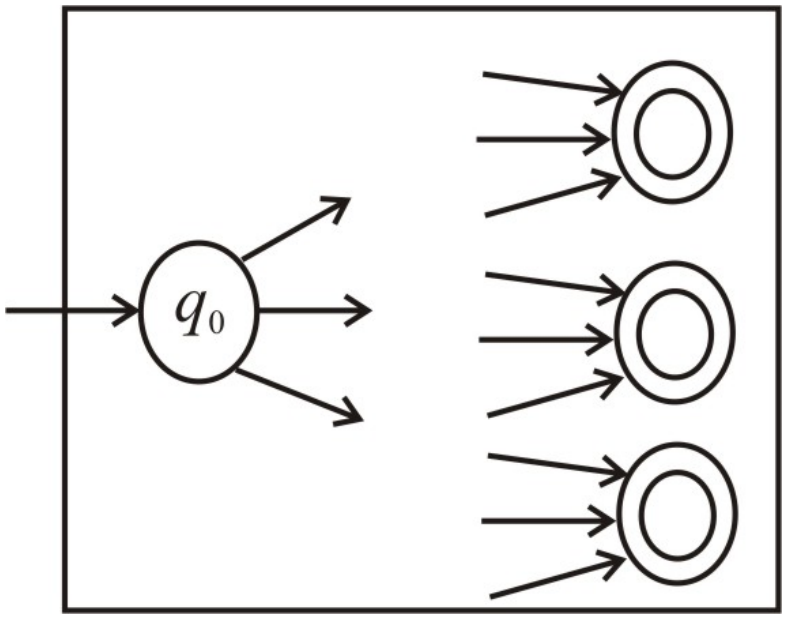
Now we build a NFA M' for A^R as follows:

• Reverse all the arrows of *M*

• Convert the start state for M as the only accept state q'_{accept} for M'.

• Add a new start state q_0' for M', and from q_0', add $\in -$transitions to each state of M' corresponding to accept states of *M*.

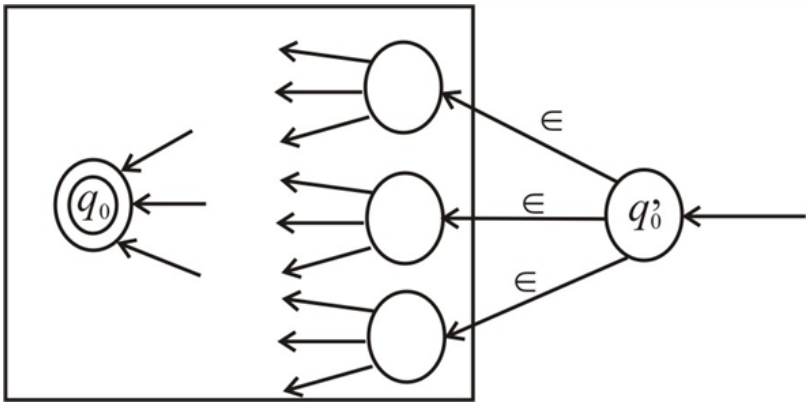
M



Comment

Step 2 of 3

M':



Comment

Step 3 of 3

Here $q_0' = q'_{accept}$

• For any $w \in \Sigma*$, there is a path following *w* from the start state to an accept sate in *M* iff there is a path following w^R from q_0' to q'_{accept} in M'

• That means that $w \in A$ iff $w^R \in A^R$.

Comments (1)

Problem

$$\Sigma_3 = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}.$$

Σ_3 contains all size 3 columns of 0s and 1s. A string of symbols in Σ_3 gives three rows of 0s and 1s. Consider each row to be a binary number and let

$$B = \{w \in \Sigma_3^* | \text{ the bottom row of } w \text{ is the sum of the top two rows}\}.$$

For example,

$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \in B, \quad \text{but} \quad \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \notin B.$$

Show that B is regular. (Hint: Working with $B^{\mathcal{R}}$ is easier. You may assume the result claimed in Problem 1.31.)

Step-by-step solution

Step 1 of 3

Consider the data

- $\Sigma_3 = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}$
- **A string of symbols in Σ_3 gives 3 rows of 0s and 1s.**
- **Each row to be a binary number**
- **$B = \{w \in \Sigma_3^*$ the bottom row of w is the same of the top two row $\}$ is the language over Σ_3.**

Comment

Step 2 of 3

Already know that "regular languages are closed under reversal".
Then, if prove that B^R is regular, then automatically B is regular and vice – versa.
So, first have to prove that B^R is regular.
A language is said to be regular if some automaton recognizes it.

Comment

Step 3 of 3

Let M be the automaton that recognizes B^R.

- M has 2 states.

(i) c_0, which denotes that the string that we have read so for leads to a carry 0.

(ii) c_1, that stands for carry 1.

Now $M = (Q, \Sigma, \delta, q_0, F)$

Where $Q = \{c_0, c_1\}$

= set of states

$$\Sigma = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \ldots \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}$$

= set of alphabets

$q_0 = c_0$

= start state

$F = \{c_0\}$

= set of final states.

δ is given as:

- $\delta(c_0, a) = c_0$ if $a = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$ or $\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$
- $\delta(c_0, a) = c_1$ if $a = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$
- $\delta(c_1, a) = c_1$ if $a = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$ or $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$
- $\delta(c_1, a) = c_0$ if $a = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

All other arrows go to trap state. Then, the defined a automation M to recognize B^R Therefore B^R is a regular language. As B^R is regular, B is also a regular language.

Comment

Problem

Let

$$\Sigma_2 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}.$$

Here, Σ_2 contains all columns of 0s and 1s of height two. A string of symbols in Σ_2 gives two rows of 0s and 1s. Consider each row to be a binary number and let

$$C = \{w \in \Sigma_2^* | \text{ the bottom row of } w \text{ is three times the top row}\}.$$

For example, $\begin{bmatrix} 0 \\ 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} \in C$, but $\begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} \notin C$. Show that C is regular. (You may assume the result claimed in Problem 1.31.)

Step-by-step solution

Step 1 of 5

Consider the language,

$C = \left\{ w \in \Sigma_2^* \mid \text{ the bottom row of } w \text{ is three times the top row} \right\}$

Over the alphabet $\Sigma_2 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$

Comment

Step 2 of 5

Here each row is binary number.

• The regular languages are closed under reversal. Use this property to prove the language *C* is regular.

• Scan the input in reverse order.

• Begin with the lower order bits and multiply it by 3 in binary format to add the multiplicand with the result of shifting the multiplicand itself by 1 bit.

• Consider a binary number 110 (6). Three times of 110(6) is 18 (10010) which is obtained by adding 110(6) with 1100(12).

• The top row can be represented as $X_n X_{n-1} ... X_1 X_0$.

• The bottom row can be represented as $P_n P_{n-1} ... P_1 P_0$.

• Add $X_{n-1} X_{n-2} ... X_0 \ 0$ to the top row to obtain the bottom row. It can be represented as follows:

$$\begin{array}{l} X_n X_{n-1} ..... X_1 X_0 \\ X_{n-1} X_{n-2} ... X_0 \ 0 \\ \hline P_n P_{n-1} ..... P_1 P_0 \end{array}$$

• The value of P_i depends on the values of X_i and X_{i-1}.

• The bit at the top of the column is the carry-in (C_i^{in}). For the first column the carry-in will be zero.

• If two 1 bits are added, the carry will be generated. It is called carry-out (C_{i-1}^{out}).

• The P_i can be obtained by performing the XOR operation on X_i, X_{i-1} and C_i^{in}. The carry-out of the currently working column will be given as the carry-in for the next column.

Comment

Step 3 of 5

The language is said to be regular if there exists a finite automaton (DFA or NFA) for that language.

• To prove that the language C is regular, construct a DFA.

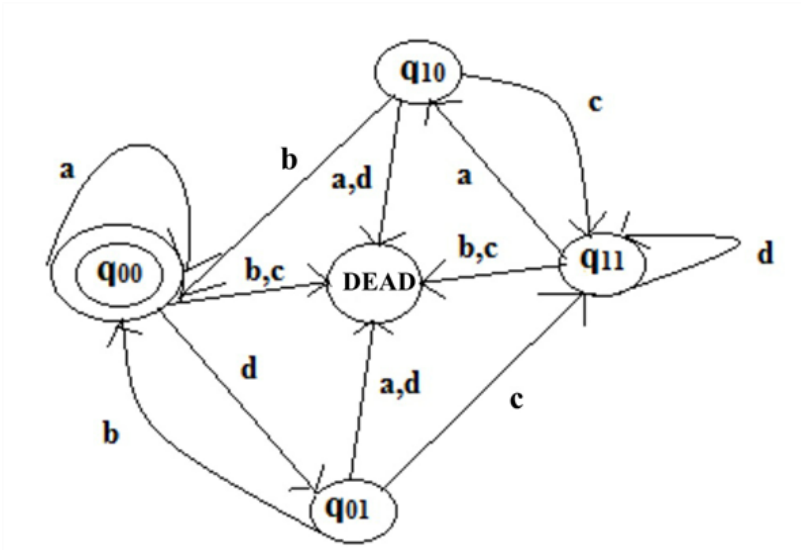
• It is necessary with 4 states without counting sink state to keep track of all the possible combinations of C_{i+1}^{in} and X_{i-1} and for each of these states, there will be exactly two possible valid output transitions depending on whether X_i is 0 or $1.$

• Assume that q_{ij} represents the state for which $C^{in}(Carry-in)$ is equal to *i* and the preceding symbol is observed at the top is equal to *j*.

Comment

Step 4 of 5

The state diagram of DFA is as follows:



Where $a=\begin{bmatrix}0\\0\end{bmatrix},b=\begin{bmatrix}0\\1\end{bmatrix},c=\begin{bmatrix}1\\0\end{bmatrix},d=\begin{bmatrix}1\\1\end{bmatrix}$

Consider an example string $\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}$.

Read the string in reverse order. Here, the initial state is $q00$. The input $\begin{bmatrix}0\\0\end{bmatrix}$ to the $q00$, stays in the same state. The input $\begin{bmatrix}1\\1\end{bmatrix}$ to $q00$, moves to the state $q01$. The input $\begin{bmatrix}0\\1\end{bmatrix}$ to $q01$, moves to the state $q00$. The input $\begin{bmatrix}0\\0\end{bmatrix}$ to the $q00$, stays in the same state. The string $\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix}$ is accepted in reverse order.

Comment

Step 5 of 5

Since a DFA that recognizes C^R is built. Therefore C^R is regular. Hence, it is proved that *C* is regular.

Comment

Problem

Let Σ_2 be the same as in Problem 1.33. Consider each row to be a binary number and let

$$D = \{w \in \Sigma_2^* | \text{ the top row of } w \text{ is a larger number than is the bottom row}\}.$$

For example, $\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} \in D$, but $\begin{bmatrix}0\\0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} \notin D$. Show that D is regular.

Step-by-step solution

Step 1 of 4

Given language is

$D = \left\{ w \in \Sigma_2^* \mid \text{ the top row of } w \text{ is the larger number than is the bottom row} \right\}$

Over the alphabet $\Sigma_2 = \left\{ \begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix} \right\}$

Language for given expression L = $\left\{ \varepsilon, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}, ... \right\}$

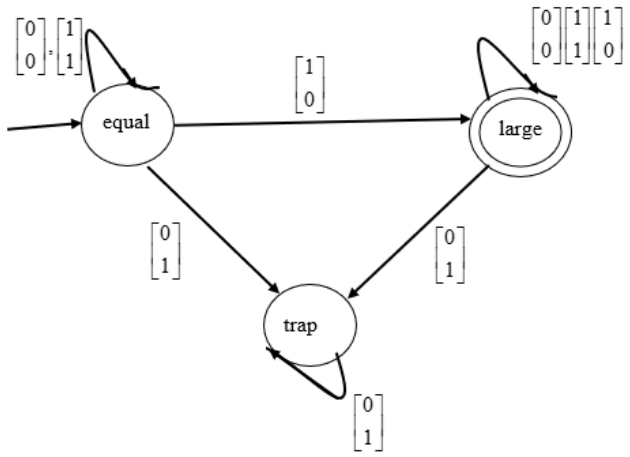
Here each row is binary number.

Comment

Step 2 of 4

Let **M** be the DFA, over the input alphabet $\Sigma_2 = \left\{ \begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix} \right\}$.

The state transition diagram of **M** is as follows:



Comments (1)

Step 3 of 4

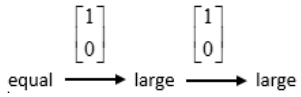
We must prove that *D* is a regular language.

A language is said to be regular if it recognizes by a DFA.

Let take string form language D, $w = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}$

Initial state of the above DFA is ‘equal’

Parse string



Here ‘large’ is final state , the string is accepted by the DFA.

Comment

Step 4 of 4

Thus, language of given D is accepted by the given DFA.

we defined a DFA to recognize the language D.

Therefore, D is a regular language.

Comment

Problem

Let $\sum_2$ be the same as in Problem 1.33. Consider the top and bottom rows to be strings of 0s and 1s, and let

$$E = \{w \in \Sigma_2^* | \text{ the bottom row of } w \text{ is the reverse of the top row of } w\}.$$

Show that E is not regular.

Step-by-step solution

Step 1 of 1

Expression is not regular

Consider that $\sum_2 = \left\{\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}\right\}$

Proof: The bottom row of the string is the reverse of the top row of the string.

In order to prove that bottom row of string is the reverse of the top row of the string user need to prove this with the help of contradiction.

First it is to be assumed that the expression *E* is a regular language.

Suppose, the constant variable *p* is associated with expression *E*

Choose the string *s*. This is because $s \in L$

$$s = \begin{bmatrix}1\\0\end{bmatrix}^p \begin{bmatrix}0\\1\end{bmatrix}^p$$

$$s = 1^p 0^p$$

$$= \left(0^p 1^p\right)^R, \text{ and}$$

$$|s| = 2p \geq p$$

Now, the string *s* is being partitioned into three pieces such that,

$s = xyz$, where $p \geq q$

$y = \begin{bmatrix}1\\0\end{bmatrix}^p$, and $z = \begin{bmatrix}0\\1\end{bmatrix}^p$

For any division of $y = uvw$, the value of $v = \begin{bmatrix}1\\0\end{bmatrix}^m$ but the condition is that $0 < m \leq p$

For any value of $i \geq 0$, suppose value of *i* is assumed to be 2.

$$s = xyz$$

Now, putting the value of *y* here,

$$s = xuv^i wz \quad s = xuv^2 wz$$

$$s = \begin{bmatrix}1\\0\end{bmatrix}^{p+m} \begin{bmatrix}0\\1\end{bmatrix}^p$$

Because $m > 0$ and $1^{p+m}0^p \neq \left(0^{p+m}1^p\right)^R$

By the above contradiction, it is being proved that the top row of the string is not the reverse of the bottom row of the string.

This implies, $xuv^2 wz \notin L$

Thus, by the pumping lemma of proof by contradiction, it is being proved that the given language *L* is not a regular language.

Comment

Problem

Let $B_n = \{\mathtt{a}^k \mid k \text{ is a multiple of } n\}$. Show that for each $n \geq 1$, the language B_n is regular.

Step-by-step solution

Step 1 of 5

Consider that:

$B_n = \{a^k \mid k \text{ is a multiple of } n\}$

In order to prove that the given expression is regular, the value of *n* is chosen as greater than or equal to 1.

Comment

Step 2 of 5

Suppose, $k = ni$, where *i* is any positive integer. In starting, suppose value of *i* is chosen to be 1.

When $i = 1$ and $n = 1$

$$\begin{aligned} B_1 &= \{a^k\} \\ &= \{a^{ni}\} \\ &= \{a^{1\times 1}\} \\ &= \{a\} \end{aligned}$$

Then, the String comes out to be $\{a\}$

Comment

Step 3 of 5

Increase the value of *n* keeping the value of *i* equal to 1. When $i = 1$ and $n = 2$

$$\begin{aligned} B_2 &= \{a^k\} \\ &= \{a^{ni}\} \\ &= \{a^{2\times 1}\} \\ &= \{aa\} \end{aligned}$$

Then, the String comes out to be $\{aa\}$

Comment

Step 4 of 5

Increase the value of *n* keeping the value of *i* equal to 1. When $i=1$ and $n=3$

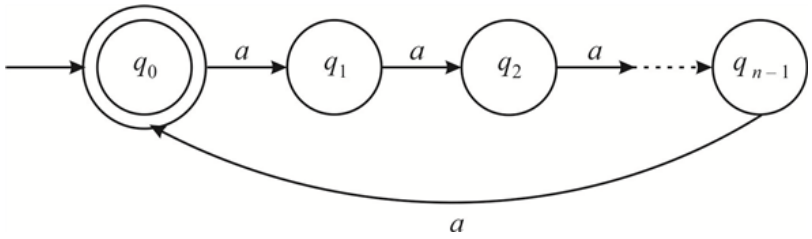
$$\begin{aligned} B_3 &= \{a^k\} \\ &= \{a^{ni}\} \\ &= \{a^{3\times 1}\} \\ &= \{aaa\} \end{aligned}$$

Then, the String comes out to be $\{aaa\}$ and so on.

Comment

Step 5 of 5

Finite automation of the regular expression is as shown:



In the above finite automaton, q_0 is the initial and final state and q_1, q_2, q_3 and q_{n-1} are the subsequent states.

The language B_n is the regular language. According to the closure property of the regular expression, it is clearly seen that the specific expression is a regular expression when the value of *n* is greater than and equal to 1.

Closure property includes various operations such as union, intersection, set complement, set reversal, set difference and many more. Assume that B_n is regular.

• Union of B_1 and B_2 results in the third string and it is also a regular expression.

• Similarly, if user applies any property of closure, then the result is the regular expression.

Hence, it is proved that the above expression is the regular expression.

Comments (2)

Problem

Let C_n = {xl x is a binary number that is a multiple of n}. Show that for each n ≥ 1, the language C_n is regular

Step-by-step solution

Step 1 of 3

Given language is

$$C_n = \{x \mid x \text{ is a binary number that is a multiple of } n\}$$

• A language is regular if it is recognized by a DFA.

• So construct a DFA to keep track of the remainder of the input when divided by *n*.

Comment

Step 2 of 3

Construction of DFA is as follows:

• Assume the value of n as 3.

• To determine whether a binary number is a multiple of 3, it is necessary to find its remainder modulo 3.

• If it ends up with remainder zero then accept.

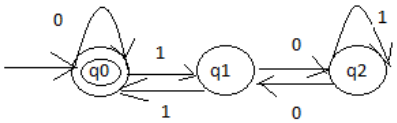
• Otherwise reject.

• Every time read a digit, the preceding string is shifted left one position thereby doubling its value x.

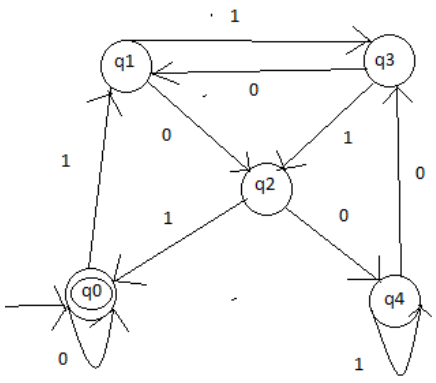
• If the current digit is 0, them the new value is $2x \pmod 3$.

• If the current digit is 1, them the new value is $2x+1 \pmod 3$.

• The DFA for the above example is as follows:



• Similarly for n=5 the DFA is as follows:



Comment

Step 3 of 3

The other case can be proved similarly.

For example:

$C_6 = C_3 0$

$C_{12} = C_6 00$

$C_{15} = C_3 \cap C_5$

....

Thus $M = \left(\{q_0, q_1, ...q_n\}, \{0,1\}, \delta, q_0, \{q_0\}\right)$

Since *M* recognizes language C_n. Hence, it is proved that C_n is regular.

Comment

Problem

An ***all***-NFA M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ if *every* possible state that M could be in after reading input x is a state from F. Note, in contrast, that an ordinary NFA accepts a string if *some* state among these possible states is an accept state. Prove that all-NFAs recognize the class of regular languages.

Step-by-step solution

Step 1 of 5

All NFA *M* is a 5-tuple (Q,Σ,δ,q_0,F) that accepts $x \in \Sigma^*$ if every possible state that M could be in after reading input *x* is a state from F.

There are two steps to prove that all – NFAs recognize the class of regular languages.

Step 1: Every regular language is recognized by some all – NFA

Step 2: Every all – NFA recognizes a regular language.

Comment

Step 2 of 5

Proof for the Step 1 as follows:

Clearly every DFA can be viewed as an all – NFA.

• Let *L* be the any regular language.

• Let $M=(Q,\Sigma,\delta,q_0,F)$ be the DFA that recognizes *L*.

• Clearly for each input string *x*,

There is exactly one possible state $q \in Q$ that M could be in after reading *x*.

• Hence, if M is viewed as an all-NFA, then it accept *x* if and only if $q \in F$, which happens if and only if $x \in L$.

• Therefore, when M is viewed as an all – NFA, it also recognizes the language L.

Thus, every regular language is recognized by some all – NFA.

Comment

Step 3 of 5

Proof for the Step 2 as follows:

• Let $N=(Q,\Sigma,\delta,q_0,F)$ be an all – NFA.

• Let *A* be the language recognized by N.

Now, prove that A is regular.

• Constructing a DFA $M=(Q',\Sigma,\delta',q_0',F)$ that recognizes *A*.

$M=(Q',\Sigma,\delta',q_0',F)$

• Where $Q'=P(Q)$, where $P(Q)$ is the set of subset of Q

• For $R \in Q'$ and $a \in \Sigma$

$$\delta'(R,a)=\bigcup_{r\in R}E(\delta(r,a))$$

• For any subset $S\subset Q, E(S)$ is the set of all sates $q\in Q$ that can reached from *S* by travelling along $\in$ arrows including the members of *S* themselves.

$$q_0'=E(\{q_0\})$$

$$F'=\{R\in Q'\mid R\subset F\}$$

Comment

Step 4 of 5

Now, consider an input string $x\in\Sigma^*$.

• Let *R* be the set of states that the all NFA *N* could be in after reading *x*.

• Then *x* is fed to the DFA *M* defined above, *M* will end at state *R*.

• By the definition of all-NFA , have $x\in A\Leftrightarrow R\subset F$.

• By the definition of *M*, *M* accepts *x* if and only if $R\in F'$, which is equivalent to $R\subset F$.

Hence M recognize A.

Comment

Step 5 of 5

Therefore, every all- NFA recognizes a regular language.

From step 1 and step 2 all- NFAs recognizes the class of regular language.

Comment

Problem

The construction inTheorem1.54 shows that every GNFA is equivalent to a GNFA with only two states. We can show that an opposite phenomenon occurs for DFAs. Prove that for every k > 1, a language $A_k \subseteq \{0,1\}^*$ exists that is recognized by a DFA with k states but not by one with only k – 1 states.

Step-by-step solution

Step 1 of 1

Given:

It is being given that if there is two states then GNFA is correspondingly equal to the GNFA. But, here user need to prove that in DFA subsequently opposite phenomenon occurs. It implies user need to prove that no DFA are equivalent to a DFA with lesser states.

Proof:

Assume A_k be the set of words of length at least $k-1$. Therefore it can be said that A_k has at least k equivalence classes of words length $0,1,2,\ldots,k-2,\text{and } k-1$ or more. So it is clear from this that A_k requires a DFA with k states.

For any DFA fewer than k states, by Pigeon Hole Principle, two of the k strings cause the machine to loop in same state results in a rejection from the DFA.

Consider $A_k = \{0,1\}^* 0^{k-1}0^*$ for $k>1$.

Now a DFA with exactly k states can recognize the language A_k. Starting from the start state there is a state in the DFA for each 0 it had read after the last 1.

After $k-1$ 0's it arrives at an accepting state whose further transitions are self-loops. Now based on the language let say A_k be the set consisting of the strings $10,100,\ldots,10^{k-1}$. If the DFA consist of fewer than k states then by the Pigeon Hole Principle two these strings cause a loop to a single state. Hence the machine fails to accept the strings.

Conclusion:

Hence, it can be said that no DFA's are equivalent to a DFA with lesser states.

Comment

Problem

Recall that string x is a ***prefix*** of string y if a string z exists where xz = y, and that x is a ***proper prefix*** of y if in addition x ≠ y. In each of the following parts, we define an operation on a language A. Show that the class of regular languages is closed under that operation.

A **a.** $NOPREFIX(A) = \{w \in A | \text{ no proper prefix of } w \text{ is a member of } A\}$.

b. $NOEXTEND(A) = \{w \in A | \, w \text{ is not the proper prefix of any string in } A\}$.

Step-by-step solution

Step 1 of 3

Consider the following information:

• String *x* is a prefix of string *y* if a string *z* exists such that *xz=y*.

• String *x* is a proper prefix of *y* if *xz=y* and $x \neq y$.

• The language *A* is regular language. Assume $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing *A*.

Comment

Step 2 of 3

a.

$NOPREFIX(A) = \{w \in A \,|\, no\ proper\ prefix\ of\ w\ is\ a\ member\ of\ A\}$

1. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing *A*.

2. Initially, find all words that have a proper prefix in *A*. The language *L* is represented as $L = \{w \in \Sigma^* : x \in A \text{ and } z \in \Sigma^* \text{ such that } xz = y\}$.

3. Now, construct the NFA $M^1 = (Q^1, \Sigma, \delta^1, q_0^1, F^1)$ for all its components such that:

• $Q^1 = Q \cup \{q_f\}$ and $q_f \notin Q$

• For $q \in Q^1$ and $a \in \Sigma$ define $\delta^1(q,a) = \begin{cases} \delta(r,a) & if\ r \notin F \\ \phi & if\ r \in F \end{cases}$

• $q_0^1 = q_0$

• $F^1 = q_f$

Proof:

• If *w* is a string in Language *L*, there is a string *y* in *A*. Here, *x* is a proper prefix of *y* such that *xz=y* and *x* is non-empty.

• If *w* is taken as input of M^1, the computation on *x* ends at an accepting state in *M* and some computation on *z* ends at state q_f.

• So *w* is accepted by M^1, which means that there is a computation that ends at q_f.

• From the construction of M^1, the computation arrives at one of the accepting states in *M* before it reaches q_f.

• If we conclude that String *x* is a proper prefix of *y*, *M* on input *x* ends in one of its accepting states. So, *w* is a member of *L*, and *x* is in A.

• As, *NOPREFIX*(*A*) is defined as $A \cap \overline{L}$ and class of regular languages are closed under intersection and complement, *NOPREFIX*(*A*) is also regular.

Comments (2)

b.

$$NOEXTEND(A) = \{w \in A \mid w \text{ is not proper prefix of any string in } A\}$$

• Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognizing *A*.

• Assume that the DFA for language M accepts that only the strings reaching the final state but not those strings that are added to reach a final state again.

• So, the strings exactly ending in final states are accepted.

• For a state $q \in F$, check whether there is a path from $q \in Q$ to any state in *F* (or a cycle involving *q*) using Depth First Search.

• Let $F^{\mid} \subseteq F$ be the set of all the states from which there is no such path.

• Now, changing the set of final states *F* to $F^{\mid}$ gives a DFA for *NOEXTEND* (*A*).

• Thus, *NOEXTEND(A)* is also regular.

Comment

Problem

For languages A and B, let the perfect shuffle of A and B be the language

$$\{w \mid w = a_1 b_1 \cdots a_k b_k, \text{ where } a_1 \cdots a_k \in A \text{ and } b_1 \cdots b_k \in B, \text{ each } a_i, b_i \in \Sigma\}.$$

Show that the class of regular languages is closed under perfect shuffle.

Step-by-step solution

Step 1 of 3

Consider the two languages *A* and *B*. The language *perfect shuffle* on *A* and *B* is as follows:
$\{w \mid w = a_1 b_1 ... a_k b_k, \text{ where } a_1 ... a_k \in A \text{ and } b_1 ... b_k \in B, \text{ each } a_i, b_i \in \Sigma\}$.

Assume, $DFA_A = (Q_A, \Sigma, \delta_A, S_A, F_A)$ and $DFA_B = (Q_B, \Sigma, \delta_B, S_B, F_B)$ be two DFAs that recognize *A* and *B* respectively.
$DFA_{Perfect-shuffle} = (Q, \Sigma, \delta, S, F)$ recognizes the language perfect shuffle on *A* and *B*.

Comment

Step 2 of 3

The DFA for perfect shuffle switches from DFA_A to DFA_B after each character is read and it tracks the current states of DFA_A and DFA_B. Each character should belong to DFA_A or DFA_B i.e., $a_i, b_i \in \Sigma$. For each character read, $DFA_{Perfect-shuffle}$ makes moves in the corresponding DFA (either DFA_A or DFA_B). After the whole string is read, if both DFA_A and DFA_B reaches to the final state, then the input string is accepted by $DFA_{Perfect-shuffle}$.

Comment

Step 3 of 3

The $DFA_{Perfect-shuffle}$ is defined as follows:

• $Q = Q_A \times Q_B \times \{A, B\}$: set of all possible states of DFA_A and DFA_B which should match with $DFA_{Perfect-shuffle}$.

• The input alphabet for $DFA_{Perfect-shuffle}$ is Σ.

• $q = (q_A, q_B, A)$: q_A and q_B are the initial states for DFA_A and DFA_B respectively. $DFA_{Perfect-shuffle}$ starts with q_A in DFA_A, q_B in DFA_B and the next character should be read from DFA_A.

• $F = F_A \times F_B \times \{A\}$: F_A and F_B are the final states for DFA_A and DFA_B respectively. $DFA_{Perfect-shuffle}$ accepts if both DFA_A and DFA_B reaches to the final states and the next character should be read from DFA_A.

• The transition function δ is,

1. $\delta((m, n, A), a) = (\delta_A(m, a), n, B)$
2. $\delta((m, n, B), b) = (m, \delta_B(n, b), A)$

Consider, the current state of DFA_A is *m* and the current state of DFA_B is *n*. Change the current state of *A* to $\delta_A(m, a)$ if the next character is to be read from DFA_A when *a* is the next character. After the character is read, read the next character from DFA_B. Change the current state of *B* to $\delta_B(n, b)$ if the next character is to be read from DFA_B when *b* is the next character.

The language *L* is said to be regular if there exist an FA that recognizes the language *L*. Here, the $DFA_{Perfect-shuffle}$ is defined for the language *perfect shuffle.*

Therefore, the class of regular languages is closed under perfect shuffle.

Comments (1)

Problem

For languages A and B, let the ***shuffle*** of A and B be the language

$$\{w \mid w = a_1 b_1 \cdots a_k b_k, \text{ where } a_1 \cdots a_k \in A \text{ and } b_1 \cdots b_k \in B, \text{ each } a_i, b_i \in \Sigma^*\}.$$

Show that the class of regular languages is closed under shuffle.

Step-by-step solution

Step 1 of 2

Consider the two languages *A* and *B*. The language *shuffle* on *A* and *B* is as follows:
$\{w \mid w = a_1 b_1 ... a_k b_k, \text{ where } a_1 ... a_k \in A \text{ and } b_1 ... b_k \in B, \text{ each } a_i, b_i \in \Sigma^*\}$.

Assume, $DFA_A = (Q_A, \Sigma, \delta_A, S_A, F_A)$ and $DFA_B = (Q_B, \Sigma, \delta_B, S_B, F_B)$ be two DFAs that recognize *A* and *B* respectively. $DFA_{shuffle} = (Q, \Sigma, \delta, S, F)$ recognizes the language perfect shuffle on *A* and *B*. For each character read, $DFA_{shuffle}$ may move from running DFA_A to running DFA_B. The *NFA* is more flexible when compared to the *DFA*. In this case, $NFA_{shuffle} = (Q, \Sigma, \delta, S, F)$ has to be constructed to allow more flexibility.

The $NFA_{shuffle}$ keeps track the current states of DFA_A and DFA_B. For each character read, $NFA_{shuffle}$ makes moves in the corresponding DFA (either DFA_A or DFA_B). After the whole string is read, if both DFA_A and DFA_B reaches to the final state, then the input string is accepted by $NFA_{shuffle}$.

Comment

Step 2 of 2

The $NFA_{shuffle}$ can be defined as follows:

- $Q = (Q_A \times Q_B) \cup \{q_0\}$ *:* The set of all possible states of DFA_A and DFA_B which should match with $NFA_{shuffle}$. Here, q_0 denotes the initial state.
- $q = q_0$
- $F = (F_A \times F_B) \cup \{q_0\}$: F_A and F_B are the final states for DFA_A and DFA_B respectively. The $NFA_{shuffle}$ accepts the string if both DFA_A and DFA_B are in accept states or $NFA_{shuffle}$ accepts the empty string.
- δ is as follows:

o $\delta(q_0, \varepsilon) = (q_A, q_B)$: At the start state q_0, the current state of DFA_A is q_A and the current state of DFA_B is q_B without reading anything.

o $(\delta_A(m,a), n) \in \delta((m,n), a)$: Change the current state of *A* to $\delta_A(m,a)$ when the character *a* is read. Here, the current state of D_A is *m* and the current state of D_B is *n*.

o $(m, \delta_B(n,a)) \in \delta((m,n), a)$: Change the current state of *B* to $\delta_B(n,a)$ when the character *a* is read. Here, the current state of D_A is *m* and the current state of D_B is *n*.

The language *L* is said to be regular if there exist an FA that recognizes the language *L*. Here, the $NFA_{shuffle}$ is defined for the language *shuffle*.

Therefore, the class of regular languages is closed under shuffle.

Comment

Problem

Let A be any language. Define *DROP-OUT*(A) to be the language containing all strings that can be obtained by removing one symbol from a string in A. Thus,

$$DROP\text{-}OUT(A) = \{xz \mid xyz \in A \text{ where } x, z \in \Sigma^*, y \in \Sigma\}.$$

Show that the class of regular languages is closed under the *DROP-OUT* operation. Give both a proof by picture and a more formal proof by construction as in Theorem 1.47.

THEOREM **1.47**

The class of regular languages is closed under the concatenation operation.

Step-by-step solution

Step 1 of 4

Given that

A is any language and

$DROP\_OUT(A) = \{xz \mid xyz \in A \text{ where } x, z \in \Sigma^*, y \in \Sigma\}$

We have to prove that class of regular languages closed under DROP_OUT operation.

i.e. if *A* is a regular language then $DROP\_OUT(A)$ is also regular.

We have to take that *A* is regular and we have to prove that $DROP\_OUT(A)$ is regular.

Since *A* is a regular language, it must be recognized by a DFA.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be the DFA recognizes *A*.

Now we will construct an NFA $N = (Q', \Sigma \cup \{\in\}, \delta', q_0', F')$ that we recognize $DROP\_OUT(A)$.

There are two copies of Machine *M*.

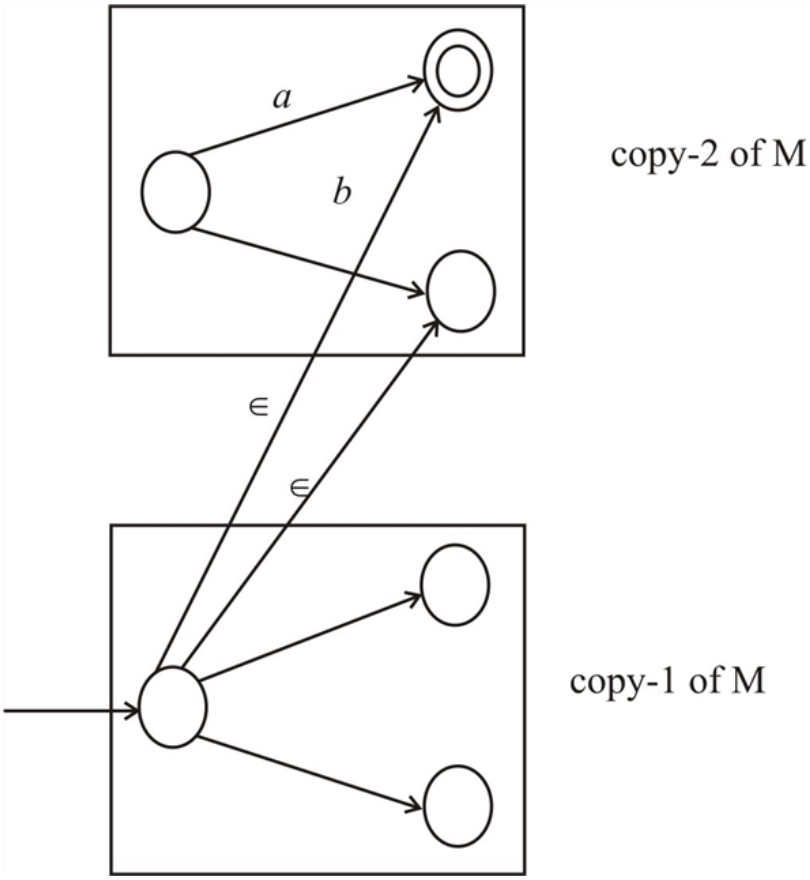
• Copy 1: Copy 1 corresponds to the state of having 'not yet skipped a symbol'

• Copy 2: Copy 2 corresponds to the state of having "already skipped a symbol".

Comment

Step 2 of 4

(i) Proof by picture:-



Comment

Step 3 of 4

$N=\left(Q',\Sigma\cup\{\in\},\delta',q_0',F'\right)$

- $Q'=\{(q,b)\mid q\in Q,b\in\{0,1\}\}$ = set of states
- $q_0'=$ start state

$=(q_0,0)$

- $F'=$ set of final states

$=\{(q,1)\mid q\in F\}$

- δ' is gives as follows:

$\rightarrow\ \delta'((q,b),a)=\{(\delta(q,a),b)\}\ \forall\, q\in Q,b\in\{0,1\},a\in\Sigma$

This means that both the copy1 and copy2 of the machine *M* do exactly as the original machine does on every symbol a of the alphabet

$\rightarrow\ \delta'((q,0),\in)=\{(\hat{q},1)\mid\exists a\in\Sigma,\delta(q,a)=\hat{q}\}$

Also at every stage, the machine has the option to skip a character. The only accepting sates are in copy 2. This means, the machine cannot accept a string without skipping a character.

Comment

Step 4 of 4

(ii) Formal proof:

→ The formal proof is given by induction on the length the string.

→ An appropriate inductive hypothesis is to assume that, for any string *w* of length *k*,

• The machine *M* stays in the copy -1 iff it has not yet skipped a symbol.

i.e. $\delta'^*((q_0,0),w)=(q_1,0)$ iff $\delta^*(q_0,w)=q_1$

• The machine *M* jumps to the copy-2 iff there is some symbol a that is skipped.

i.e. $\delta'^{*}((q_0,0),w)=(q_1,1)$ iff $\delta^{*}(q_0,w_1 a\, w_2)=q_1$.

So in both (i) and (ii) we constructed an NFA N that recognizes the language $DROP\_OUT(A)$.

Thus $DROP\_OUT(A)$ is regular.

Hence class of regular languages closed under $\mathrm{DROP\_OUT}$ operation.

Comment

Problem

Let B and C be languages over ∑ = {0, 1}. Define

$$B \overset{1}{\leftarrow} C = \{w \in B \mid \text{ for some } y \in C, \text{ strings } w \text{ and } y \text{ contain equal numbers of 1s}\}.$$

Show that the class of regular languages is closed under the $\overset{1}{\leftarrow}$ operation.

Step-by-step solution

Step 1 of 2

Given that *B* and *C* are two languages and $B \overset{1}{\leftarrow} C = \{w \in B \mid \text{ for some } y \in C, \text{srings } w \text{ and } y \text{ contain equal numbers of 1}\}$ over the alphabet $\Sigma = \{0,1\}$

We have to prove that class of regular languages closed under $\overset{1}{\leftarrow}$ operation

That means if *B* and *C* are regular languages than $B \overset{1}{\leftarrow} C$ is also a regular language.

So given that *B* and *C* are regular languages.

We know that

“A language is regular if it is recognized by an automation”

• Let M_B be the DFA that recognizes the language *B*

$M_B = (Q_B, \Sigma, \delta_B, q_B, F_B)$

• Let M_c be the DFA that recognizes the language *C* $M_C = (Q_C, \Sigma, \delta_C, q_C, F_C)$

Now we have to construct an NFA which recognizes $B \overset{1}{\leftarrow} C$.

Comment

Step 2 of 2

Construction of NFA to recognize $B \overset{1}{\leftarrow} C$:

Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA.

Now *N* has to decide whether a string $w \in B \overset{1}{\leftarrow} C$ or not.

• For that first machine M checks whether $w \in B$ or not.

• If $w \in B$, then non deterministically find out a string of that contains the same number of 1s as contained in *w* and checks that $y \in C$.

• That means for each string *B*, there are $C(\text{number ofstrings in } C)$ parallel machings will exist

Thus $Q =$ set of states

$= Q_B \times Q_C$

$\Sigma =$ set of alphabet

= same as B and C

δ is given by, for $(q,r) \in Q$ and $a \in \Sigma$

$$\delta(q,r),a=\begin{cases}\{(\delta_B(q,0),r)\} & \text{if } a=0\\ \{((\delta_B(q,1)),\delta_C(r,1))\} & \text{if } a=1\\ \{(q,\delta_C(r,0))\} & \text{if } a=\in\end{cases}$$

$q_0 =$ start state

$=(q_B,q_c)$

$F =$ set of final states

$=F_B \times F_C$

Thus we defined an NFA *N* to recognize $B \xleftarrow{1} C$.

Hence $B \xleftarrow{1} C$ is regular.

Therefore class of regular languages closed under $B \xleftarrow{1} C$ operation

---

Comments (3)

Problem

Let $A/B = \{w \mid wx \in A \text{ for some } x \in B\}$. Show that if A is regular and B is any language, then A/B is regular.

Step-by-step solution

Step 1 of 2

Given language is

$A/B = \{w \mid wx \in A \text{ for some } x \in B\}$

Here, *A* is a regular language and *B* is any language.

Now, the objective is to prove that A/B is regular.

Since *A* is a regular language, some DFA will recognize the language *A*.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be the DFA that recognizes *A*.

Here, *Q* is the set of states.

Σ is set of alphabets = of the alphabets for *A* and *B*.

δ is the transition function.

q_0 is the start state.

F is the set of final states.

Comment

Step 2 of 2

To prove A/B is a regular language, construct a DFA that recognizes the language A/B

Let $M' = (Q', \Sigma', \delta_0', F')$ be the DFA that recognizes A/B.

- Q' = set of states = *Q*
- Σ' = set of alphabets $= \Sigma$
- δ' = transition function $= \delta$
- q_0' = start state $= q_0$
- $F' = \{q \in Q \mid \exists x \in B \text{ such that } M \text{ goes from } q \text{ to some sate in } F \text{ on reading } x\}$

Thus, a DFA M' to recognize the language A/B has been constructed.

Hence A/B is a regular language.

Comment

Problem

Prove that the following languages are not regular. You may use the pumping lemma and the closure of the class of regular languages under union, intersection, and complement.

a. $\{0^n 1^m 0^n \mid m, n \geq 0\}$

A **b.** $\{0^m 1^n \mid m \neq n\}$

c. $\{w \mid w \in \{0,1\}^* \text{ is not a palindrome}\}^8$

* **d.** $\{wtw \mid w, t \in \{0,1\}^+\}$

Step-by-step solution

Step 1 of 5

Pumping lemma:

For a regular language *A* with the pumping length *P*, if S is any string of *A* with minimum length of *P*, then the string S can be divided into 3 pieces *x, y* and *z* represented as $S = xyz$ should satisfy the following conditions:

- for each
- $|y| > 0$, and
- $|xy| \leq P$

Comment

Step 2 of 5

a.

Consider the Language $L = \{0^n 1^m 0^n \mid m, n \geq 0\}$.

Assume that *L* is regular language and a string $S = 0^P 10^P$. Divide the string into three pieces *x, y* and *z*. So, $S = 0^P 10^P = xyz$ where, *P* is the pumping length.

Assume that $x = 0^{P-K}, y = 0^K$ and $z = 10^P$(where $K > 0$)

Now $xy^0 z = 0^{P-K}\left(0^K\right)^0 1\, 0^P$

$= 0^{P-K} 10^P \notin L \quad \left[\because y^0 = \in\right]$

The String $xy^0 z$ does not belong to *L* because *P-K < P.*

So, the assumption that *L* is regular is a contradiction. Thus, by using pumping lemma it is proved that *L* is not regular.

Comments (3)

Step 3 of 5

b.

Consider the Language $L=\{0^m1^n \mid m \neq n\}$.

Assume that *L* is regular language and string $S=0^P1^{P+P!}$. Divide the string into three pieces *x, y* and *z*. So, $S=0^P1^{P+P!}=xyz \in L$ and $|S| \geq P$ where, *P* is the pumping length.

Note: *P!* is divisible by all integers from 1 to *P*, where $P!=P\times(P-1)\times(P-2)\times...\times1.$

Assume that $x=0^a, y=0^b$ and $z=0^c1^{P+P!}$, where $b \geq 1$ and $a+b+c=P$.

Now take string $s'=xy^{i+1}z,$ where $i=\frac{P!}{b}$.

Then $y^i=0^{P!}$ so $y^{i+1}=0^{b+P!}$, and

So $xyz,=0^{a+b+c+P!}1^{P+P!}$.

That gives $xyz=0^{P+P!}1^{P+P!} \notin L,$ a contridiction.

Here, $m=P+P!, n=P+P!$ and $m=n$. This is a contradiction to the assumption because $m=n$. Thus, by using pumping lemma it is proved *L* is not regular.

Comment

Step 4 of 5

c.

Consider the Language $L=\{w \mid w \in \{0,1\}^* \text{ is not a palindrome}\}$.

Assume that *L* is regular language.

The compliment of the language L is $\overline{L}=\{w \mid w \in \{0,1\}^* \textit{ is palindrome}\}$ is also regular.

Assume a string $S=0^P1\,0^P$. Divide the string into three pieces *x, y* and *z*. So, $S=0^P1\,0^P=xyz \in L$ where, *P* is the pumping length.

Assume that $x=0^{P-K}, y=0^K$ and $z=10^P$ where $K>0$

Now $xy^0z=0^{P-K}\left(0^K\right)^0 10^P$

$=0^{P-K}10^P \notin L \quad [\because y^0=\in]$

The String xy^0z is not same from forward and backward direction because *P-K < P*.

So, the string xy^0z does not belong to $\overline{L}$. So, the assumption is a contradiction.

Thus, by using pumping lemma it is proved *L* is not regular.

Comment

Step 5 of 5

d.

Consider the Language $L=\{wtw \mid w,t \in \{0,1\}^+\}$.

Assume that *L* is regular language.

Assume a string $S=0^P1\,0^P$. Divide the string into three pieces *x, y* and *z*. So, $S=0^P1\,0^P=xyz \in L$ where, *P* is the pumping length.

Assume that $x=0^{P-K}, y=0^K$ and $z=10^P$ where $K>0$

Now $xy^0z=0^{P-K}\left(0^K\right)^0 10^P$

$=0^{P-K}10^P \notin L \quad [\because y^0=\in]$

The String xy^0z does not belong to *L* because *P-K < P* and not of the form *wtw* as in *L*.

So, the assumption is a contradiction. Thus, by using pumping lemma it is proved *L* is not regular.

Comments (3)

Problem

Let $\Sigma = \{1, \#\}$ and let

$$Y = \{w \mid w = x_1 \# x_2 \# \cdots \# x_k \text{ for } k \geq 0, \text{ each } x_i \in 1^*, \text{ and } x_i \neq x_j \text{ for } i \neq j\}.$$

Prove that Y is not regular.

Step-by-step solution

Step 1 of 4

Consider the following language over the alphabet $\Sigma = \{1, \#\}$.

$Y = \{w \mid w = x_1 \# x_2 \# ... \# x_k \text{ for } k \geq 0, \text{ each } x_i \in 1^*, \text{ and } x_i \neq x_j \text{ for } i \neq j\}$. The language *Y* accepts the words of the form $x_1 \# x_2 \# ... \# x_k$ where $x_1, x_2, \ldots x_k$ are the substrings that are formed with any number of 1s. Here, x_i can be any number of 1s.

The words that are accepted by the language *Y* contains only 1s and #s because $\{1, \#\}$ are the input alphabet. Any two substrings cannot be equal i.e., $x_i \neq x_j$. Every two substrings are separated by the input alphabet #.

The language is said to be regular if it is satisfied by the pumping lemma. Otherwise the language is not regular.

Comment

Step 2 of 4

Pumping lemma:

If *A* is a regular language, then there is a number *p* (the pumping length) where *S* is any string that belongs to *A* of length at least *p*, then *S* may be divided into three pieces, $S = uvw$, satisfying the following conditions.

1. For each $i \geq 0, uv^i w \in A$
2. $|v| > 0$, and
3. $|uv| \leq p$

Comment

Step 3 of 4

Assume that Y is a regular language.

Let p be the pumping length for Y. The strings of the language Y are of the form $w = x_1 \# x_2 \# ... \# x_k$.

Consider a string $S = x_1 \# x_2$ for $k = 2$ and $x_1 \neq x_2$. Here, x_1 and x_2 can be formed with only 1s but both cannot be equal. Any two different strings can be taken for x_1 and x_2.

Assume $x_1 = 1^p 1$ and $x_2 = 111^p$. Then the string $S = 1^p 1 \# 111^p$. Here, x_1 and x_2 are two different substrings and the value of x_1 and x_2 depends on the *p* value. For example, if *p*=2 then the values of x_1 and x_2 are 111 and 1111.

Clearly, the length of *S* is greater than *p* and $S \in Y$.

Comment

Step 4 of 4

Let $111\#1111$ be the string that belongs to Y. The pumping length of the string is 2.

To satisfy the conditions of the pumping lemma, divide the string $111\#1111$ into three parts *u*, *v* and *w*. Here *u* is equal to 1, *v* is equal to 1, *w* is equal to $1\#1111$ (the remaining part of the string).

$$S = 111\#1111$$
$$= \underbrace{1}_{u}\ \underbrace{1}_{v}\ \underbrace{1\#1111}_{w}$$

Pump the middle part such that $uv^iw\ (i \geq 0)$. For *i*=2, the *v* becomes 11.

$$S = (1)\ (1)^i\ (1\#1111)$$
$$= \underbrace{1}_{u}\ \underbrace{11}_{v}\ \underbrace{1\#1111}_{w} \qquad [when\ i{=}2]$$

The string after pumping is $1111\#1111$.

The string $1111\#1111 \notin Y$ because, the substring x_1 is equal to x_2 which violates the condition of the language *Y*. It is a contradiction. Thus, the pumping lemma is violated.

Therefore, Y is not a regular language.

Comment

Problem

Let ∑ = {0,1} and let

D = {wlw contains an equal number of occurrences of the substrings 01 and 10}

Thus 101 $\in$ D because 1010 contains two 10s and one 01. Show that D is a regular language.

Step-by-step solution

Step 1 of 2

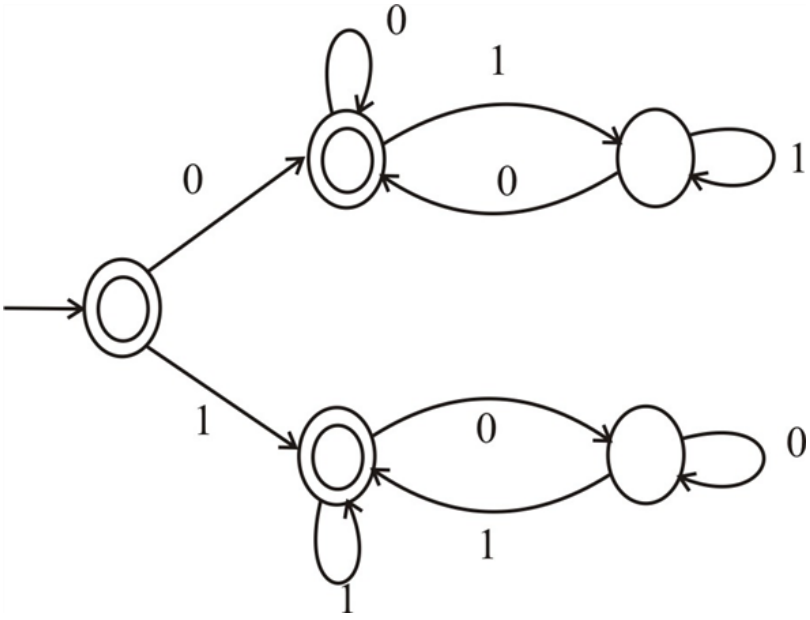
Given language is

$D = \{w \mid w \text{ contains an equal number of occurrences of the substring } 01 \text{ and } 10\}$ over the alphabet $\Sigma = \{0,1\}$

We have to prove that *D* is a regular language.

A language is regular if some DFA recognizes it.

The following DFA recognizes the language *D*.



Comment

Step 2 of 2

This DFA recognize the language *D*.

Thus *D* is regular

Comment

Problem

a. Let

$$B = \{1^k y \mid y \in \{0,1\}^* \text{ and } y \text{ contains at least } k \text{ 1s, for } k \geq 1\}.$$

Show that B is a regular language.

b. Let

$$C = \{1^k y \mid y \in \{0,1\}^* \text{ and } y \text{ contains at most } k \text{ 1s, for } k \geq 1\}.$$

Show that C isn't a regular language.

Step-by-step solution

Step 1 of 2

(a)

The language given is as follows:

$B = \{1^k y \mid y \in \{0,1\}^*\}$ and y contains at least k 1s, for $k \geq 1\}$

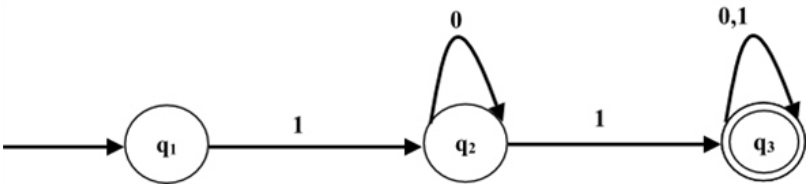
From the definition 1.16: A language is said to be Regular language if some finite automaton recognizes it.

String in language B must start with a 1 and contains at least one other 1, if k=1. So, if k is positive that, any string that start with a 1 and contains at least one other 1 matches in the y. B is defined by regular expression $10^*1(1 \cup 0)^*$ and therefore B is regular.

To show *B* is regular, the **definition 1.16** can be used.

Let *M* be the DFA recognizes the language *B*.

The state diagram of *M* is as follows:



Since there is a DFA recognizing the language *B*.

Hence it is proved that *B* is regular language.

Comments (4)

Step 2 of 2

(b)

The language given is as follows:

$C = \{1^k y \mid y \in \{0,1\}^*$ and y contains at most k 1s, for $k \geq 1\}$

The proof for to prove *C* is not regular language is as follows:

Assume that *C* is a regular language.

Assume *P* as the pumping length.

Consider a string $S = 1^P 0 1^P \in C$

Using the pumping lemma, *S* can be written as

$S = 1^P 01^P = uvw$ such that $|uv| \leq P, |y| > 0$ and $uv^i w \in C \ \ \forall i \geq 0$

When $i = 0, uw = 1^{P-t} 0 \ 1^P$ for some *k* where the number of 1's in *y* is less than or equal to *k*. $1^k y = 1^{P-t} 01^P$.

From this, *y* must contain the substring 01^P.

So $k \leq P - t$ with the number of 1s in $y \geq P$.

So, the number of 1's in *y* is always greater than *k* since $t > 0$.

Therefore, it is proved that uw does not belong to language *C*.

Since the above statement results a contradiction.

Hence, it is proved that the given language *C* is not a regular language.

Comment

Problem

Read the informal definition of the finite state transducer given in Exercise 1.24. Prove that no FST can output w^R for every input w if the input and output alphabets are {0,1}.

Step-by-step solution

Step 1 of 1

Let $w = w_1 w_2 ... w_n$ then reverse of w can be written as $w^R = w_n ... w_2 w_1$

We have to prove that no FST(Finite state transducer) can output w^R for every input w_1 over the alphabets $\{0,1\}$

Let us assume that FST *T* output w^R on input *w*.

Consider two input strings 00 and 01

(i) On input 00, FST *T* must produce output 00 because T outputs w^R on input *w*.

(ii) On input 01, FST *T* must produce output 10 because T outputs w^R on input *w*.

In these both case the first input bit is same i.e., 0.

But first output bits differ (0 and 1).

But from actual definition of FST, FST can produce its first output bit before it reads its second input.

Thus FST cannot operate to produce $w^R = 10$ on input $w = 01$.

Thus our assumption that FST *T* outputs w^R on input *w* is wrong.

Hence no FST outputs w^R on input *w*.

Comment

Problem

Let x and y be strings and let L be any language. We say that x and y are ***distinguishable by*** L if some string z exists whereby exactly one of the strings xz and yz is a member of L; otherwise, for every string z, we have xz $\in$ is an equivalence relation.

Step-by-step solution

Step 1 of 1

L be any language and x and y are the strings

Distinguishable by L :

Strings x and y are distinguishable by L if $\exists$ some z such that exactly one of the strings xz and yz belongs to L.

Indistinguishable by L :

Strings x and y are indistinguishable by L if for every string z , $xz \in L$ whenever $yz \in L$

If x and y are indistinguishable by L then we write $x \equiv_L y$

Now we have to show that $\equiv_L$ is an equivalence relation.

• To show that $\equiv_L$ is an equivalence relation, we have to show that $\equiv_L$ is

(i) Reflexive

(ii) Symmetric

(iii) Transitive

• According to the given data, $x \equiv_L y$ means "for every string *z*, *xz* is in L whenever *yz* is in L". That means, "for every string z, xz is in L iff yz is in L"

(i) Reflexivity: $x \equiv_L x$ is true

For all strings z, xz is in L iff xz is in L

Therefore $x \equiv_L x$ is true.

Hence $\equiv_L$ is reflexive.

(ii) Symmetry: $x \equiv_L y$ implies $y \equiv_L x$

If $x \equiv_L y$ is true then "for all *z*, *xz* is in L iff *yz* is in L"

Which is equivalent to "for all *z*, *yz* is in L iff *xz* is in L"

Therefore $y \equiv_L x$ is also true.

Hence $\equiv_L$ is symmetric.

(iii) Transitivity: If $a \equiv_L b$ and $b \equiv_L c$ then $a \equiv_L c$

This means that

"for all z, az is in L iff bz is in L and

For all z, bz is in L iff cz is in L".

Therefore, "for all z, az is in L iff cz is in L".

That is, $a \equiv_L c$ is true

Hence $\equiv_L$ is transitive.

From (i), (ii) and (iii)

$\equiv_L$ is equivalence relation.

Comment

Problem

Myhill–Nerode theorem. Refer to Problem 1.51. Let L be a language and let X be a set of strings. Say that X is ***pairwise distinguishable by*** L if every two distinct strings in X are distinguishable by L. Define the ***index of*** L to be the maximum number of elements in any set that is pairwise distinguishable by L. The index of L may be finite or infinite.

a. Show that if L is recognized by a DFA with k states, L has index at most k.

b. Show that if the index of L is a finite number k, it is recognized by a DFA with k states.

c. Conclude that L is regular iff it has finite index. Moreover, its index is the size of the smallest DFA recognizing it.

Step-by-step solution

Step 1 of 4

The definition of **Myhill-Nerode theorem** is as follows:

Myhill–Nerode theorem: for any language L

• ***Distinguishable by L:*** *x* and *y* are the strings ***distinguishable*** by L, for the string *z* in generating of the strings $xz \text{ or } yz$ is a member of L.

• ***Indistinguishable by L:*** *x* and *y* are ***indistinguishable*** by L for the string *z* we have $xz \in L$ every time $yz \in L$. We can write $x \equiv_L y$.

• ***Pair-wise distinguishable by L:*** set of strings contains in S, if every two separate strings are distinguishable in L.

• ***Index of L:*** It can count as finite or infinite. Language L contains max number of elements which are ***pair-wise distinguishable***.

Comment

Step 2 of 4

(a) Language L recognized by DFA (Deterministic Finite Automata) as M with number of states is k. We have to prove that L has an index at most k.

Take a contradiction assumption i.e., L has an index greater than k.

If L contains index more than k then $k+1$ strings are at least in any set S which is ***pair wise distinguishable by*** L.

Pigeonhole's principle:

We will find two distinct strings $x \text{ and } y$ from S, such that the state of DFA M after reading input x is the same as the state of DFA M after reading input y.

By applying **Pigeonhole's** principle both $xz \text{ and } yz$ are not in L. This is not satisfying the definition ***Distinguishable by L*** in **Myhill-Nerode theorem**

Hence contradiction occurs. Therefore our assumption that L has index greater than k is wrong. So, L has index at most k.

Comment

Step 3 of 4

(b) Index of Language L contains k finite states i.e., set $S = \{s_1, s_2, ...s_k\}$. We have to prove that L recognized by DFA with k states.

• Let $M = (Q, \Sigma, \delta, q_0, F)$ be *DFA* with *k* states that recognizes L

• The construction of M is as follows:

o Assume $Q = (q_1, q_2 ... q_k)$ is the set of states.

o Transition function is given as: $\delta(q_i, a) = q_j$ if $s_i a$ and s_j are not distinguishable.

o $F = \{q_i \mid s_i \in L\}$ be the setoff

o Start state q_0 be the state such that s_i and the empty string $\in$ are not distinguishable by L.

• We show that if string *t* and s_j are not distinguishable by L, the state of M will be q_j after reading *t* as input.

• By the definition of *F*, M accepts *t* if and only if *t* is in L.

• Hence M recognizes L.

Comment

Step 4 of 4

(c) Language L is regular if it contains finite index. Index is size of smallest DFA recognizing it.

(i) if L is regular then L has finite index:

• Let us assume that L is regular.

• M be *DFA* that recognizes L.

• Let k be the number of states in M.

• Then from part (a), L has index at most k

(ii) if L has finite index then L is regular:

• Let us assume that L has finite index k

• Then from part (b) we can contract a *DFA* with *k* states recognizing L

• We know that "A language is regular if and only if it is recognized by some *DFA*"

• Therefore L is regular language.

Therefore from (i) and (ii) *L* is regular if and only if it has finite index.

• The index k is size of the smallest *DFA* M recognizing it, suppose on the opposing that is not true. From part (a) we could terminate that L has indexed fewer than k, which contradicts fact that L has index equal to k.

Comment

Problem

Let ∑ = {0, 1, +, =} and

ADD = {x=y+zl x, y, z are binary integers, and x is the sum of y and z}.

Show that ADD is not regular.

Step-by-step solution

Step 1 of 3

Consider the following language over the alphabet $\Sigma = \{0,1,+,=\}$.

$ADD = \{x = y + z \mid x, y, z \text{ are binary integers, and } x \text{ is the sum of } y \text{ and } z\}$.

The language is said to be regular if it is satisfied by the pumping lemma, otherwise the language is not regular.

Comments (3)

Step 2 of 3

Pumping Lemma:

If *A* is any regular language then there is a number p (the pumping length) where *S* is any string that belongs to *A* of length at least *P*, then *S* may be divided into three pieces, $S = uvw$, satisfying the following conditions.

1. For each $i \geq 0, uv^i w \in A$
2. $|v| > 0$, and
3. $|uv| \leq p$

Comment

Step 3 of 3

Assume that ADD is a regular language.

Let p be the pumping length given by the pumping lemma. The strings of the language *ADD* are of the form $x = y + z$. Consider a string $1^{p+1} = 10^p + 1^p \in ADD$.

Let $111 = 100 + 11$ be the string that belongs to *ADD*. The pumping length of the string is 2. To satisfy the conditions of the pumping lemma, divide the considered string into three parts. Here, *u* is equal to 1, *v* is equal to 1, *w* is equal to 1=100+11 (the remaining part of the string).

$$S = 111 = 100 + 11$$
$$= \underset{u}{1}\ \underset{v}{1}\ \underbrace{1 = 100 + 11}_{w}$$

Pump the middle part such that $uv^i w\ (i \geq 0)$. For *i*=2, the *v* becomes 11.

$$S = (1)\ (1)^i\ (1 = 100 + 11)$$
$$= \underset{u}{1}\ \underset{v}{11}\ \underbrace{1 = 100 + 11}_{w} \qquad [when\ i{=}2]$$

The string after pumping is $1111 = 100 + 11$.

The string $1111 = 100 + 11 \notin ADD$ because, addition of two binary numbers 100, 11 is not equal to the binary number 1111. It is a contradiction. So, the pumping lemma is violated.

Therefore, ADD is not a regular language.

Comments (1)

Problem

Consider the language $F = \{\mathrm{a}^i\mathrm{b}^j\mathrm{c}^k \mid i, j, k \geq 0 \text{ and if } i = 1 \text{ then } j = k\}.$

a. Show that F is not regular.

b. Show that F acts like a regular language in the pumping lemma. In other words, give a pumping length p and demonstrate that F satisfies the three conditions of the pumping lemma for this value of p.

c. Explain why parts (a) and (b) do not contradict the pumping lemma.

Step-by-step solution

Step 1 of 4

Consider the language:

$F = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and if } i = 1 \text{ then } j = k\}$

The language F is the union of three disjoint languages.

$F_0 = \{b^j c^k \mid j, k \geq 0\}, F_1 \{ab^j c^j \mid j \geq 0\}, F_2 = \{a^i b^j c^k \mid i \geq j, k \geq 0\}$

Clearly F_0 and F_2 are regular languages.

The class of regular languages is closed under union and complement.

Thus $\overline{F_0 \cup F_2}$ is also regular.

We have $F_1 = F - (F_0 \cup F_2)$

$= F \cap \overline{F_0 \cup F_2}$

Since the class of regular languages is closed under intersection if *F* is regular, then so is F_1.

Comments (1)

Step 2 of 4

a.

Use pumping lemma to show that F_1 is not regular and hence neither is F.

Assume that F_1 is regular language.

Let *P* be the pumping length given by pumping lemma.

Consider a string $S = ab^P c^P \in F_1$

Clearly $|S| > P$.

By using the pumping lemma, *S* can be divided into three pieces.

i.e., $S = ab^P c^P = uvw$ such that $|uv| \leq P, |v| > 0$ and $uv^i w \in F_1 \quad \forall i \geq 0$.

Take $u = a \quad v = b^P \quad w = c^P$

$uv^0 w = a(b^P)^0 c^P \quad \therefore (i = 0)$

$= ac^P \notin F_1$

The string *w* consist of '*c*'s. In this case, string $a\left(b^{P}\right)^{0}c^{P}$ has more *c*s than '*b*'s and so is not a member of F_1, violating the condition of pumping lemma.

This is contradiction. The previous assumption that F_1 is regular is wrong. Thus F_1 is not regular.

Therefore, *F* is also not a regular language.

Comments (1)

Step 3 of 4

b.

Let pumping length $P=2$

• Show that every string $S \in F$ of length at least *P* can be divided into three pieces $S=uvw$ such that, $|uv| \leq P, |v|>0$ and $uv^{l}w \in F \; \forall l \geq 0$.

• Consider a string $a^{i}b^{j}c^{k} \in F$ of length at least 2.

• Choose *x* to be the empty string.

• If $i \neq 2$, then choose *y* to be the first symbol in $a^{i}b^{j}c^{k}$.

• If $i=2$, then choose $y=aa$

Clearly these chosen x, y satisfies the three conditions of pumping lemma.

Comment

Step 4 of 4

c.

part (a) and part(b) do not contradict the pumping lemma because the pumping lemma just says if a language is regular then there is a pumping length for that language. But, if the language satisfies the pumping lemma, the language may not be regular.

Comment

Problem

The pumping lemma says that every regular language has a pumping length p, such that every string in the language can be pumped if it has length p or more. If p is a pumping length for language A, so is any length p' ≥ p. The ***minimum pumping length*** for A is the smallest p that is a pumping length for A. For example, if A = 01^*, the minimum pumping length is 2. The reason is that the string s = 0 is in A and has length 1 yet s cannot be pumped; but any string in A of length 2 or more contains a 1 and hence can be pumped by dividing it so that x = 0, y = 1, and z is the rest. For each of the following languages, give the minimum pumping length and justify your answer.

a. 0001^*

b. 0^*1^*

c. $001 \cup 0^*1^*$

d. $0^*1^+0^+1^* \cup 10^*1$

e. $(01)^*$

f. ε

g. $1^*01^*01^*$

h. $10(11^*0)^*0$

i. 1011

j. Σ^*

Step-by-step solution

Step 1 of 10

a.

The pumping length could not be 3 as 000 being in the language it cannot be pumped. Consider the string length be 4 or more and divide in xyz as x being 000, y being the first 1 and z being everything after then it satisfies every condition of pumping lemma.

Hence the minimum pumping length is 4.

Comment

Step 2 of 10

b.

String ε is in the language but it could not be pumped so the pumping length could not be 0. According to pumping lemma's three condition arises which are as follows:

- If user divide the string in xyz as x is ε
- y is first symbol $(0|1)$
- z being the everything after then it holds.

Hence the minimum pumping length is 1.

Comment

Step 3 of 10

c.

The string 001 is in the language but if it is generated by **001** then it cannot be pumped. If string s is larger than 3 and in the language, then it is generated by 0^*1^*. Dividing the string according to Pumping Lemma's condition into a string xyz where x is ε, y be the first symbol and z be the remaining, user can pump the string.

Hence the minimum pumping length is 4.

Comment

Step 4 of 10

d.

The string 11 is in the language but it cannot be pumped so the length is not 2. Let s be the string in the language of length at least 3. If s is generated by $0^*1^+0^+1^*$ it can be divided in xyz as x is ε, y is the first string and z is everything else so it can be pumped.

Again if s is generated by 10^*1 then also user can write it as xyz where x is 1, y is 0, and z is the remainder so it could be pumped.

Hence the minimum pumping length is 3.

Comment

Step 5 of 10

e.

Let s be a string in the language.

Now s could be ε but it cannot be pumped so the length is not 0. Next s could be 01 which if divide in xyz as x is empty string ε, y is 01, and z is everything after then it satisfies the three conditions of pumping lemma.

Pumping length is not 1 because since there is no string of length 1 in the language.

Hence the minimum pumping length is 2.

Comment

Step 6 of 10

f.

Let s be a string in the language then s is ε and according to pumping lemma it cannot be pumped. As per the pumping lemma, pumping length should greater then equal to 1.

Hence the minimum pumping length is 0.

Comments (1)

Step 7 of 10

g.

The minimum pumping length of the language could not be 2 as 00 being in the language it could not be pumped. Let s be the string of length at least 3 in the language so minimally s could be 100 or 010 or 001.

Now dividing s in xyz in all the three cases user get, for 100 x is ε, y is 1, and z is the remainder, for 010 x is 0, y is 1, and z is the remainder and for 001 x is 00, y is 1, and z is the remainder. All satisfies Pumping Lemma's three conditions.

Hence the minimum pumping length is 3.

Comment

Step 8 of 10

h.

The minimum length string in the given language is 100 but it cannot be pumped. Next minimum length string is 10100. If divide it according to the pumping lemma in xyz then x be 10, y be 10, and z be the rest of the string then y can be pumped.

Hence the minimum pumping length is 4.

Comment

Step 9 of 10

i.

If s be a string in the language then s is 1011. If set p = 4, then claim s is pumpable (which it is not, as it is the only string in the language). This should be 5.

Hence then the minimum pumping length is 5.

Comment

Step 10 of 10

j.

Say s be a string in the language Σ^*. According to pumping lemma if divide s in xyz then x be the empty string, y is $(\varepsilon \mid 0 \mid 1)$ and z is empty string. Now ε could not be pumped.

Hence the minimum pumping length is 1.

Comment

Problem

If A is a set of natural numbers and k is a natural number greater than 1, let

$B_k(A)$ = {wl w is the representation in base k of some number in A}.

Here, we do not allow leading 0s in the representation of a number. For example, $B_2(\{3, 5\})$ = {11, 101} and $B_3(\{3, 5\})$ = {10, 12}. Give an example of a set A for which $B_2(A)$ is regular but $B_3(A)$ is not regular. Prove that your example works.

Step-by-step solution

Step 1 of 1

Regular and Non Regular Expression

Assume $A = \{2n \mid \text{n is a natural number}\}$
$= \{2,\ 4,\ 8,\ 16,\ 18,\ 20 \ldots\}$

So $B_2(A) = \{10, 100, 1000, 10000 \ldots\}$ should be regular, because $B_2(A)$ is recognized by regular expression $10*$.

Now $B_3(A) = \{2,\ 11,\ 22,\ 121 \ldots\}$

$B_3(A)$ is non regular.

It can be proved by contradiction that $B_3(A)$ is non regular.

Take on the contrary that $B_3(A)$ is regular.

Now p will be pumping length according to pumping lemma.

Choose u as element of $B_3(A)$ and the length of u should at least $p+1$.

Since $u \in B_3(A)$ and $i > 1$, by pumping lemma u can be divided into three part, $u = xyz$ where $\forall i \geq 0$ the string $xy^i z \in B_3(A)$

According to condition three of pumping lemma, $|xy| \leq p$, and $|z| > 0$.

If rightmost digit of z is 0, then u will be the power of 3. But u is power of 2, hence it is impossible. Hence the rightmost digit of 0 will must be 1 or may be 2.

For $i > 1$, $u' = xy^i z$ is power of 2 which is greater than u, so it is easy to generate it after u is added to itself few numbers of times. That is, if u is added at least 3 times to itself, there must be carry from the right to left column of z. When i will increase, the carries will affect more columns. For very large i, the carries will bleed on y, as pumping lemma condition two says that y should not be empty.

Power of 2 will be generated for very large value of i which cannot be generated by copying y to i times. The carries will force y as well as z to change. Hence, it is showing that $B_3(A)$ does not satisfy pumping lemma. Hence, the initial assumption that $B_3(A)$ is regular, is wrong.

Hence, $B_3(A)$ is non-regular.

Comment

Problem

If A is any language, let $A_{\frac{1}{2}\_}$ be the set of all first halves of strings in A so that

$$A_{\frac{1}{2}\_} = \{x \mid \text{ for some } y,\ |x| = |y| \text{ and } xy \in A\}.$$

Show that if A is regular, then so is $A_{\frac{1}{2}\_}$.

Step-by-step solution

Step 1 of 2

Consider the language *A* is regular. Let $A_{\frac{1}{2}\_}$ be the set of all first halves of strings in *A*.

$A_{\frac{1}{2}\_} = \{x \mid \text{ for some } y,\ |x| = |y| \text{ and } xy \in A\}$

Since *A* is a regular language, the DFA *M* recognizes the language *A*.

$M = (Q, \Sigma, \delta, q_0, F)$

where, Q is the set of states

Σ is the input alphabet

δ is the transition function

q_0 is the start state

F is the final state

The language is said to be regular if there exists an FA for it. In this case, construct an NFA *N* that recognizes $A_{\frac{1}{2}\_}$. Let *x* be the first part and choose *y* such that $|x| = |y|$. Here, $x \in A_{\frac{1}{2}\_}$. To prove the language $A_{\frac{1}{2}\_}$ is regular, run two DFAs at the same time one forward and the other backward. Run the DFA *M* on input *x* in forward direction and run the DFA *M* on input *y* in backward direction parallelly. The input string is accepted if both simulations reach the same state.

Comments (1)

Step 2 of 2

Construction of NFA N to recognize $A_{\frac{1}{2}\_}$:

Let $N = (Q', \Sigma, \delta', q_0', F')$ where,

(i) $Q' = Q \times Q \cup \{q_0'\}$ set of states contains the following:

• Special start state q_0' and

• A cross product $q \times q \times q$ where

• The first part tracks the performance of *M* on *x*

• The second part does the same thing for *y*.

• The third part tests whether the guess on M is consistent or not.

(ii) $\Sigma =$ input alphabet

(iii) q_0' is the start state

(iv) $F' =$ set of final states $= \left\{ \left\langle q_i, q_j, q_k \right\rangle \mid q_i, q_k, q_j \in Q \right\}$

(v) $\delta' =$ Rules of transition are as follows:

• There exists an ε move from the start state to the all the states in $\{(q_0, q_f) \mid q_f \in F\}$.

• Consider the states $q_i, q_j, q_k, q_l \in Q$. There exists a move from (q_i, q_j) to (q_k, q_l) on input symbol $a \in \sum$ if and only if $\delta(q_i, a) = q_k$ and $\delta(q_l, b) = q_j$.

The NFA N is constructed to recognize $A_{\frac{1}{2}}$. Thus, $A_{\frac{1}{2}}$ is regular.

Therefore, if A is regular then $A_{\frac{1}{2}}$ is also regular.

Comment

Problem

If A is any language, let $A_{\frac{1}{3}-\frac{1}{3}}$ be the set of all strings in A with their middle thirds removed so that

$$A_{\frac{1}{3}-\frac{1}{3}} = \{xz | \text{ for some } y,\ |x| = |y| = |z| \text{ and } xyz \in A\}.$$

Show that if A is regular, then $A_{\frac{1}{3}-\frac{1}{3}}$ is not necessarily regular.

Step-by-step solution

Step 1 of 2

Constraints:

A is any language and $A_{\frac{1}{3}-\frac{1}{3}} = \{x | \text{ for some } y, |x| = |y| = |z| \text{ and } xyz \in A\}$ is the set of all strings in A with their middle thirds removed.

• Let $A = \{a*\#b*\}$ is regular language

• We know that $\{a*b*\}$ is a regular language.

• Also we know that "Regular languages are closed under intersection"

• Now $A_{\frac{1}{3}-\frac{1}{3}} \cap \{a*b*\} = \{a^n b^n \mid n \geq 0\}$

• Clearly $\{a^n b^n \mid n \geq 0\}$ is not regular, because if p is the pumping length and

$S = xyz = aabb$ is $p = 2$ Here $x = a\ y = a\ z = bb$, obtain $xy^2z = aaabb$

Comment

Step 2 of 2

Pumping lemma : If A is a regular language, then there is a pumping length p where, if s is any string in A of length at least P, then s may be divided into three pieces, $s = xyz$, satisfying following conditions

(i) For each $i \geq 0, xy^iz \in A$

(ii) $|y| > 0$ and

(iii) $|xy| \leq p$

So according to pumping lemma $xy^2z = a^3b^2 \notin \{a*b*\}$

Hence $\{0*1*\}$ is not regular.

As regular languages are closed under intersection and $\{0*1*\}$ is not regular, $A_{\frac{1}{3}-\frac{1}{3}}$ is not regular. If A is regular, then $A_{\frac{1}{3}-\frac{1}{3}}$ is not necessarily regular is proved.

Comment

Problem

Let $M = (Q, \Sigma, \delta, q_0, F)$ (q, s) equals the state where M ends up when M starts at state q and reads input s.) Say that M is ***synchronizable*** if it has a synchronizing sequence for some state h. Prove that if M is a k-state synchronizable DFA, then it has a synchronizing sequence of length at most k^3. Can you improve upon this bound?

Step-by-step solution

Step 1 of 1

Given:

A DFA $M = (Q, \Sigma, \delta, q_0, F)$ and suppose h is a state of DFA M known as its home state.

Proof:

The at most length of synchronizing sequences is $k^3 0$ for a k-state synchronizable DFA. In the year 1964, a Slovak scientist named Jan Cerny first tried to solve the problem of synchronizing automata in real time. This problem is sometimes referred as *Cerny's Conjecture.*

To prove the upper bound on the synchronizing sequence we try to device a greedy algorithm.

Algorithm:

1. Let a synchronizing DFA be $M = (Q, \Sigma, \delta, q_0, F)$. Initialize the synchronizing sequence $\omega \leftarrow 1$ (empty word) and a set of states $P \leftarrow Q$.

2. ***while*** $|P| > 1$

a. Find a word υ which is belongs to Σ^* and it has minimum length $|\delta(P, \upsilon)| < |P|$.

b. If none exists, ***return*** failure.

c. $\omega \leftarrow \omega\upsilon$

$P \leftarrow \delta(P, \upsilon)$

3. ***return*** ω

• Now suppose that M is a k-state DFA, that is, $|Q| = k$ then clearly the main loop of the algorithm runs at most $k-1$ times. In order to get the length of the output word ω user has to estimate the length of each word υ derived at each loop.

• Consider a generic step at which $|P| = n > 1$ and let $\upsilon = a_1 \cdots a_l$ with $a_i \in \Sigma,\ i = 1 \cdots l$. Then it is quite simple to see that the sets, $P_1 = P,\ P_2 = \delta(P_1, a_1),\ \ldots,\ P_l = \delta(P_{l-1}, a_{l-1})$ are n-element subsets of Q.

• Furthermore, since $|\delta(P_l, a_l)| < |P_l|$, there exists two states $q_l, q_l' \in P_l$ such that $\delta(q_l, a_l) = \delta(q_l', a_l)$.

• Now define two element subsets $R_i = \{q_i, q_i'\} \subseteq P_i,\ i = 1, \cdots, l-1$. then the condition that υ is a word that has minimum length $|\delta(P, \upsilon)| < |P|$ which implies that $R_i \not\subset P_j$ for $1 \le j < i < l$. Now by the Peter Frankl inequality, the tight bound over l be $\binom{k-n+2}{2}$.

• Summing up these inequalities from $n = k$ to $n = 2$, user can get the upper bound over the synchronizing sequence $|\omega| \le \frac{k^3 - k}{6}$.

Conclusion:

Therefore, at most length of synchronizing sequences is $k^3 0$ for a k-state synchronizable DFA.

Comment

Problem

Let $\sum = \{a, b\}$. For each $k \geq 1$, let C_k be the language consisting of all strings that contain an a exactly k places from the right-hand end. Thus $C_k = \sum^* a \sum^{k-1}$. Describe an NFA with k + 1 states that recognizes C_k in terms of both a state diagram and a formal description.

Step-by-step solution

Step 1 of 2

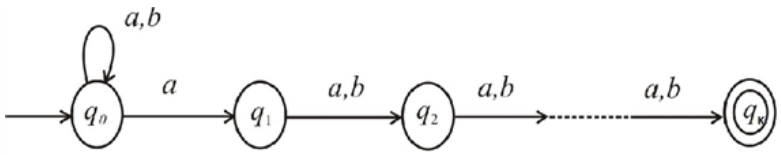
Given language is

$C_k = \Sigma^* a\, \Sigma^{k-1}$ for each $K \geq 1$, over the alphabet $\Sigma = \{a, b\}$

C_k is the language consisting of all strings that contains an '*a*' exactly *K* places from the right – hand end.

Let *N* be the NFA with $K+1$ states that recognizes C_K

(i) The state diagram of NFA *N* is follows:



Comment

Step 2 of 2

(ii) The formal description of NFA *N* is as follows:

$N = (Q, \Sigma, \delta, q_0, F)$

$Q =$ set of sates $= \{q_0, q_1 \ldots, q_K\}$

$\Sigma =$ set of alphabet $= \{a, b\}$

$q_0 =$ start state $= \{q_0\}$

$F =$ set of final states $= \{q_K\}$

$\delta =$ The transition function is given as follows:

$$\delta(q_i, a) = \begin{cases} \{q_0, q_1\} & \text{if } i = 0 \\ \{q_{i+1}\} & \text{if } 0 < i < k \\ \phi & \text{if } i = k \end{cases}$$

$$\delta(q_i, b) = \begin{cases} \{q_0\} & \text{if } i = 0 \\ \{q_{i+1}\} & \text{if } 0 < i < k \\ \phi & \text{if } i = k \end{cases}$$

$\delta(q_i, \in) = \phi \, \forall i.$

Comment

Problem

Consider the languages C_k defined in Problem 1.60. Prove that for each k, no DFA can recognize C_k with fewer than 2_k states.

Step-by-step solution

Step 1 of 2

Consider the language $C_k = \Sigma^* a \Sigma^{k-1}$ for each $k \geq 1$, over the alphabet $\Sigma = \{a, b\}$. C_k be the language consisting of all strings that contain an '*a*' exactly k places from the right-hand end.

Comment

Step 2 of 2

Now it is required to prove that, no DFA (Deterministic finite automation) can recognize C_k with fewer than 2^k states.

If a DFA enters two different states after reading two different strings *lz* and *mz* with same suffix *z*, then the DFA enters two different states after reading the strings *l* and *m*. Take two different strings of length *k* such that $l = l_1 \ldots . l_k$ and $m = m_1 \ldots . m_k$. Let *i* be some position such that $l_i \neq m_i$. One of the strings contains *a* in its i^{th} position and the other string contains *b* in its i^{th} position.

Consider the suffix string $z = b^{i-1}$. In this case, either the string *lz* or *mz* has the k^{th} bit from the end as *a*. The total number of strings of length *k* over the input alphabet $\{a, b\}$ is 2^k. Thus, the DFA needs 2^k states in order to distinguish 2^k strings.

Therefore, the DFA that recognizes the language C_k has at least 2^k states.

Comment

Problem

Let ∑ = {a, b}. For each k ≥ 1, let D_k be the language consisting of all strings that have at least one a among the last k symbols. Thus $D_k = \Sigma^* a(\Sigma \cup \varepsilon)^{k-1}$. Describe a DFA with at most k+1 states that recognizesD_k in terms of both a state diagram and a formal description.

Step-by-step solution

Step 1 of 2

The Language $D_k = \Sigma^* a(\Sigma \cup \varepsilon)^{k-1}$ for each $k \geq 1$, over the alphabet $\Sigma = \{a, b\}$. Here, D_k is the language consisting of all strings that have at least one a among the last k symbols. The DFA (deterministic finite automata) that recognizes the language D_k.

The construction of M where $M = (Q_k, \Sigma, \delta_k, q_0, F)$ is as follows:

- $Q_k = \{q_0, q_1, q_2, ...q_k\}$ = set of states
- $\Sigma = \{a, b\}$ = set of alphabets
- $q_0 = \{q_0\}$ = start state.
- $F = \{q_1, q_2 ... q_k\}$ = set of final states
- δ_k = Transition function is given as follows

$$\delta_k(q, l) = \begin{cases} q_1 & i = 0 \wedge l = a \\ q_0 & i = 0 \wedge l = b \\ q_1 & i \neq 0 \wedge l = a \\ q_{(i+1) \bmod k} & i \neq 0 \wedge l = b \end{cases}$$

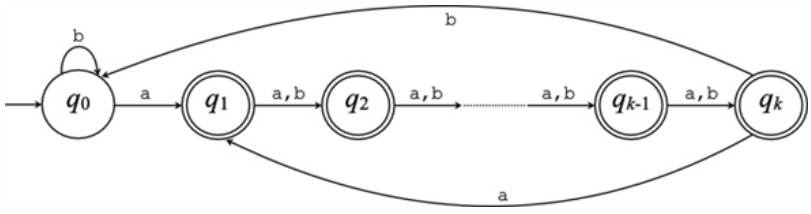
Comment

Step 2 of 2

In other words,

- M is in state q_0 if it has not seen and a within the past k letters then it will reject.
- It is in state q_i if it saw a from i letters ago.

The complete DFA is as follows:



Comment

Problem

a. Let A be an infinite regular language. Prove that A can be split into two infinite disjoint regular subsets.

b. Let B and D be two languages. Write $B \in D$ if $B \subseteq D$ and D

Step-by-step solution

Step 1 of 2

a.

The proof that an infinite regular language, say A, can be split into two infinite disjoint regular subsets is as follows:

• Let there be a string, say 's', such that $s \in A$ and s = xyz, where x, y and, z, represent the sub-strings of the string s.

• Since s belongs to the language A, and the language A is regular, xy^iz must belong to A, where $i \geq 0$. (As per the condition 1 of pumping lemma).

• Let A_1 be a language such that $A_1 = \{xy^{2i}z, \text{where } i \geq 0\}$.

• Since all the strings of the form xy^iz belong to A, the strings of the form $xy^{2i}z$ must also belong to A.

• Hence, the language A_1 is a subset of the language A, i.e:

$A_1 \subset A$

• The strings of the language A_1 can be represented by the following regular expression:

$x(yy)^*z$

Hence, the language A_1 is a regular language

• Since in the expression, $A_1 = \{xy^{2i}z, \text{where } i \geq 0\}$, there is no upper limit for the value of i, the language A_1 is infinite.

• Since the regular languages are closed under the operation of complement, the language $\overline{A_1}$ is a regular language.

• Let A_2 be a language such that, $A_2 = \overline{A_1} \cap A$.

• Since the regular languages are closed under the operation of intersection, the language A_2 is a regular language.

• Since the languages, A_1 and A are infinite, the language A2 is also infinite

• Clearly A_2 and A_1 are two disjoint sets.

• Also, $A = A_1 \cup A_2$

Thus, the language A can be split into two infinite disjoint regular subsets.

Hence, proved.

Comment

Step 2 of 2

b.

The steps required to prove the given statement are as follows:

• Divide the regular language D into two regular disjoint subsets and let one of those subsets be B.

• Let the other subset be A, such that A = D – B.

• Since D contains infinitely many strings that are not in B, A also contains infinitely many strings that are not in B.

• Further divide the language A into two disjoint subsets, A_1 and A_2, such that A_2 contains infinitely many strings that are not in A_1 and vice versa.

• Since A contains infinitely many strings that are not present in B, A_1 also contains infinitely many strings that are not in B.

• Create a set C such that $C = A_1 \cup B$.

• Since A_1 contains infinitely many strings that are not in B, C also contains infinitely many strings that are not in B.

• Clearly, B is a subset of C.

• Hence, the following statement is true:

$B \Subset C$

• Since A_2 contains infinitely many strings that are not in A_1, D contains infinitely many strings that are not present in A_1.

• Since D contains infinitely many strings that are not present in A_1, D contains infinitely many strings that are not present in C.

• Hence, the following statement is true:

$C \Subset D$.

• Since $B \Subset C$ and $C \Subset D$, the following statement is true:

$B \Subset C \Subset D$

Hence, proved.

Comment

Problem

Let N be an NFA with k states that recognizes some language A.

a. Show that if A is nonempty, A contains some string of length at most k.

b. Show, by giving an example, that part (a) is not necessarily true if you replace both A's by $\overline{A}.$

c. Show that if $\overline{A}$ contains some string of length at most 2^k.

d. Show that the bound given in part (c) is nearly tight; that is, for each k, demonstrate an NFA recognizing a language A_k where $\overline{A_k}$ shortest member strings are of length exponential in k. Come as close to the bound in (c) as you can.

Step-by-step solution

Step 1 of 5

Let $N=(Q,\Sigma,\delta,q_0,F)$ be an NFA with *K* states that recognizes some language *A*

(a) Suppose *A* is non empty

• Then there must be an accept state $q\in F$ that can be reached from the start state q_0.

• Let *w* be the string that can be accepted by *N* when traveling along the shortest path q_0 to q.

• Let *n* be the length of *w*.

• Then, the sequence of state $q_0,q_1...q$ in the shortest path from q_0 to *q* has length $n+1$.

• Note that all the states from q_1 to q in this sequence must be distinct; otherwise we would find a shorter path from q_0 to q by removing the repeated states.

• Since there are only *K* states in N and there are *n* distinct states in the shortest path from q_0 to q, we have $n\le K$.

• Clearly, *w* is accepted by *N* because *q* is an accept state

• So *A* contains a string of length at most *K*.

Comment

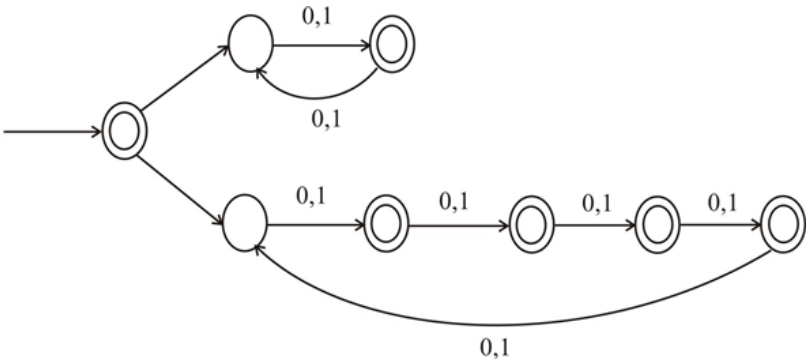
Step 2 of 5

(b) Example:

Suppose $\Sigma=\{0,1\}$ and N be the NFA with $K=\delta$ states.

Let *A* be the language recognized by *N*.

The State diagram of *N* is as follows



Comment

Step 3 of 5

• Clearly *N* accepts the empty string.

• For any nonempty string *w*, *N* will reject *w* if and only if the length of *w* is divisible by 2 and 5.

• Thus $\overline{A}$ consists of all non empty strings of length divisible by 10

• So $\overline{A}$ is non-empty and the shortest string in $\overline{A}$ has length $10 > K$

• Hence we got the contradiction of part (a) when we replace *A* by $\overline{A}$.

Comment

Step 4 of 5

(c) We know that " Every non deterministic finite automaton has an equivalent

deterministic finite automaton"

• So we convert *N* into a DFA *M* that also recognizes *A*, where the set of states in *M* is the set of subsets of *Q*.

• Then we swap the accept and non accept states of *M* to obtain a DFA $\overline{M}$ that recognizes $\overline{A}$.

• Note that $\overline{M}$ has 2^K states.

• Applying part (a) by replacing *N* with $\overline{M}$, we can conclude that if $\overline{A}$ is non-empties, then $\overline{A}$ contains a string of length at most 2^K.

Comment

Step 5 of 5

(d) The idea used in part (b) can be generalized to obtain a bound close to an exponential form.

• Let $2 \le P_1 < P_2 < ... < P_n \le \sqrt{K}$ be all the primes in $\left[1, \sqrt{K}\right]$

• For each P_i, construct a DFA M_i of P_i states that rejects only strings of length divisible of P_i.

• Finally, construct an NFA *M* by union all these M_i machines in the following ways:

→ create a separate starting state q_0 add an $\in$ transition from q_0 to the starting states of each M_i, and also designate q_0 as an accepting state.

→ This machine M will have $1 + P_1 + P_2 + ... + P_m \le K$ states.

→ On the other hand, *M* rejects a string *w* if and only if a is non empty and $|w|$ is divisible by $P_1 P_2 ... P_m$.

→Hence, the shortest string is rejected by *M* has length $P_1 P_2 ... P_m$.

→ Now coming to the analysis of the bound,

By the prime Number theorem, there are approximately $\ln$ *n* prime numbers in $[0, n]$ for *n* sufficiently large.

→ Hence, $m \approx \frac{1}{2} \ln K$.

→ Since there are about $\frac{1}{4} \ln K$ primes in $\left[0, \sqrt[4]{K}\right]$, there are about $m - \frac{1}{4} \ln K \approx \frac{1}{4} \ln K$ primes in $\left[\sqrt[4]{K}, \sqrt{K}\right]$

Thus $P_1 P_2 ... P_m \ge \left(\sqrt[4]{K}\right)^{\frac{1}{4} \ln K} = K^{\frac{1}{16} \ln K}$

Problem

Prove that for each n > 0, a language B_n exists where

a. B_n is recognizable by an NFA that has n states, and

b. if $B_n = A_1 \cup \cdots \cup Ak$, for regular languages A_i, then at least one of the A_i requires a DFA with exponentially many states.

Step-by-step solution

Step 1 of 2

a.

Proving that a language is recognizable by an NFA

Suppose B_n be a language where $n > 0$. User has to prove that the language is recognizable by an NFA with n states.

BASIS: Let $n = 1$ hence $B_n = \{\varepsilon, 0, 1\}$. Therefore formally we can design an NFA $N = (\{q_o\}, \Sigma, \delta, q_0, \{q_0\})$ with a single state that accepts all the given language as $\delta(q_0, \varepsilon \mid 0 \mid 1) = q_0$.

Proof by induction: suppose one can divide B_n in two regular expressions say $E \text{ and } F$ of length $n_1, n_2 < n$ and $n_1 + n_2 = n$.

Now by inductive hypothesis it can easily concluded that the NFA's accepting $E \text{ and } F$ are consisting of at least $n_1 \text{ and } n_2$ states.

But it is already known to us that the set of regular expression is closure under Union, Concatenation and Star operation.

Therefore the language B_n is recognizable by an NFA with n states.

Comment

Step 2 of 2

From the above part it is proved that a language B_n where $n > 0$ is recognizable by an NFA with n states.

Now for $B_n = A_1 \bigcup A_2 \bigcup \cdots \bigcup A_k$ where A_i's are regular.

If a DFA is constructed which is equivalent to the DFA of the given NFA.

There could be at least n and at most 2^n states in the resultant equivalent DFA. Every regular language is recognized by a DFA so there is a corresponding DFA for all the A_is. Now, by the pigeon hole principle, one can state that there is at least one DFA which requires 2^i states to recognize a language among all the A_i.

Comment

Problem

A ***homomorphism*** is a function $f:\ \Sigma \longrightarrow \Gamma^*$ for any language A.

a. Show, by giving a formal construction, that the class of regular languages is closed under homomorphism. In other words, given a DFA M that recognizes B and a homomorphism f, construct a finite automaton M` that recognizes f(B). Consider the machine M` that you constructed. Is it a DFA in every case?

b. Show, by giving an example, that the class of non-regular languages is not closed under homomorphism.

Step-by-step solution

Step 1 of 2

a.

Class of Regular Languages is closed under Homomorphism:

In general by the definition of homomorphism if R be a regular expression where each symbol a in Σ, Assume, $f(R)$ be the expression this is obtained by replacing each symbol a in R by $f(a)$.

Now assume a language $B = B(R)$ for some regular expression R. Here $f(R)$ defines the language $f(B)$.

Therefore it is needed to proof that if user take a sub-expression E of R and apply homomorphism f to it and get $f(E)$, the language of $f(E)$ is the same language if user apply f to the language $L(E)$. It implies $L(f(E)) = f(L(E))$.

BASIS: Suppose, $E = \Phi$ that is $L(E)$ does not contain any strings, then $f(L(E)) = L(f(E))$. Again suppose $E = \{\varepsilon\}$ that is $L(E)$ contains a string with no symbols, then also $f(L(E)) = L(f(E))$

Now assume $E = \{a\}$ for some symbol $a \in \Sigma$. Here, in this case $L(E) = \{a\}$ then $f(L(E)) = \{f(a)\}$. Again, $f(E)$ is the regular expression that is the string of symbols $f(a)$,

Therefore, it can be said that $L(f(E)) = \{f(a)\}$.

It implies $L(f(E)) = f(L(E))$.

INDUCTION: By the rule of union over regular expression. Assume, $E = E_1 + E_2$.

By the application of homomorphism over regular expression user may say that, $f(E) = f(E_1) + f(E_2)$. By the grammar of regular language

$L(E) = L(E_1) \cup L(E_2)$.

Therefore,

$$\begin{aligned} L(f(E)) &= L(f(E_1) + f(E_2)) \\ &= L(f(E_1)) \cup L(f(E_2)) \end{aligned}$$

Again, though f is applied to all the strings of a language separately and individually,

$$\begin{aligned} f(L(E)) &= f(L(E_1) \cup L(E_2)) \\ &= L(f(E_1)) + L(f(E_2)) \end{aligned}$$

Here by the Inductive Hypothesis we may assert that, $L(f(E_1)) = f(L(E_1))$ and $L(f(E_2)) = f(L(E_2))$. It implies $L(f(E)) = f(L(E))$.

Hence, it can be said that the class of regular languages is closed under homomorphism.

Comment

Class of non-regular language is not assumed to be closed under homomorphism:

Consider a given language $B=\left\{0^n1^n\middle|n\geq 0\right\}$ is defined over the alphabet $\Sigma=\{0,1\}$. Now by pumping lemma of regular languages it can easily be said that the language B is non-regular.

Here another alphabet set is $\Gamma=\{a,b\}$ also defined.

The homomorphism function $f:\Sigma\to\Gamma^*$ is defined as $f(0)=ab$ and $f(1)=\varepsilon$. Therefore the language B' over alphabet Γ could be described as $B'=\left\{(ab)^n\middle|n\geq 0\right\}$.

If user apply pumping lemma of regular languages over B' and divide it in xyz where x is ε, $y=ab$ and z is also ε then we can say that B' is a regular language.

Therefore user can see from this that applying homomorphism over a non-regular language yields a regular language.

Hence homomorphism is not closed over the class of non-regular languages.

Comment

Problem

Let the ***rotational closure*** of language A be $RC(A) = \{yx \mid xy \in A\}$.

a. Show that for any language A, we have *RC*(A) = *RC*(*RC*(A)).

b. Show that the class of regular languages is closed under rotational closure.

Step-by-step solution

Step 1 of 4

a) The Rotational Closure of a language A is defined as $RC(A) = \{yx \mid xy \in A\}$. Now for any language A a string $\omega \in A$ also $\omega \in \Sigma^*$, and therefore $\varepsilon \in A$.

It is known that $\omega\varepsilon = \varepsilon\omega = \omega$. Here for any language $A = \{\omega\varepsilon\}$, $RC(\omega\varepsilon) = \varepsilon\omega = \omega$ and $RC(RC(\omega\varepsilon)) = RC(\varepsilon\omega) = \omega\varepsilon = \omega$.

Therefore, $\boxed{RC(A) = RC(RC(A))}$

Comments (1)

Step 2 of 4

b) Proof: Assume the language A is being defined with the help of regular expression E. Now structural induction on the size of regular expression E is to be proved.

Now, it is to be shown that $A(E) = A(RC(A))$, which implies the language $RC(A)$ is the reverse language of the specified language A.

Basis: If E is $\varepsilon, \varnothing$, or a for some symbol a, then $RC(E) = E$. That is, $RC(\varepsilon) = \varepsilon$, $RC(\varnothing) = \varnothing$ and $RC(a) = a$.

Induction: Three cases arise depending upon the specified expressions E which are as follows:

- $E = E_1 + E_2$. Then $RC(E) = RC(E_1) + RC(E_2)$.

The relational closure of any two languages is obtained with the computation and after that taking the union of the specified two languages.

Comment

Step 3 of 4

- $E = E_1E_2$. Then $RC(E) = RC(E_2)RC(E_1)$.

Here the expression is rotated with respect to two languages and language itself

For an example let if, $L(E_1) = \{01, 111\}$ and $A(E_2) = \{00, 10\}$, then,

$$A(E_1E_2) = \{0100, 0110, 11100, 11110\}, \text{ then we can say}$$
$$RC(A(E_1E_2)) = \{0010, 0110, 00111, 01111\}, \text{ where } \varepsilon \in A.$$
$$\text{Again, } RC(A(E_1)) = \{10, 111\}$$
$$RC(A(E_2)) = \{00, 01\}$$
$$A(RC(E_2)RC(E_1)) = \{0010, 00111, 0110, 01111\}$$
$$\text{and therefore, } RC(A(E)) = A(RC(E)).$$

Comment

Step 4 of 4

- $E = E_1^*$. Then $RC(E) = (RC(E_1))^*$. Here the justification is any string ω in $A(E)$ can be written as $\omega_1\omega_2\cdots\omega_n$ where $\omega_i \in L(E)$.

But $RC(\omega) = RC(\omega_n)RC(\omega_{n-1})\cdots RC(\omega_1)$ where each $RC(\omega_i) \in A(RC(E))$ and hence $RC(\omega)$ is in $L((RC(E_1))^*)$.

Therefore it is proved that the set of regular languages are closed under rotational closure.

Comment

Problem

In the traditional method for cutting a deck of playing cards, the deck is arbitrarily split two parts, which are exchanged before reassembling the deck. In a more complex cut, called Scarne's cut, the deck is broken into three parts and the middle part in placed first in the reassembly. We'll take Scarne's cut as the inspiration for an operation on languages. For a language A, let *CUT*(A) = {yxzl xyz ? A}.

a. Exhibit a language B for which *CUT*(B) ≠ *CUT*(*CUT*(B)).

b. Show that the class of regular languages is closed under *CUT*.

Step-by-step solution

Step 1 of 2

Consider the following language:

$B=\left\{0^n1^n \mid \{0,1\} \in \Sigma, n \geq 0\right\}$.

The above language B is a language for which $CUT(B) \neq CUT(CUT(B))$. In the above language, if the string w exists in B and user divides this string w in xyz as $0^n1^m1^{n-m}$ then $CUT(B)=1^m0^n1^{n-m}$. After again dividing it becomes $CUT(CUT(B))=0^n1^n$.

Hence $CUT(B) \neq CUT(CUT(B))$

Comment

Step 2 of 2

Assume that a regular language A and prove that $CUT(A)$ is also a regular language. Again, let the DFA that accepts A be M_A.

Now construct another DFA M_{CUT} that accepts the language $CUT(A)$ and hence it will be proved that $CUT(A)$ is a regular language.

DFA Construction: Let $M=(Q,\Sigma,\delta,q_0,F)$ recognizes A. Construct $M_{CUT}=(Q_{CUT},\Sigma,\delta_{CUT},q_{CUT},F_{CUT})$ to recognize $CUT(A)$.

i) $Q_{CUT}=Q$. The states of M_{CUT} are same as M.

ii) The new start state of the DFA is q_{CUT}.

iii) $F_{CUT}=F$. Let w be a string in A accepted by M. If user divides this string w in xyz then w' in $CUT(A)$ be yxz which will be accepted by M_{CUT}. Now in both cases the machine enters into a accept state on input of the substring z. Hence the accepting states of both of the machines are same.

iv) Now define transition function of M_{CUT}, δ_{CUT} for $q_{CUT} \in Q_{CUT}$ and $a \in \Sigma$ as follows:

$$\begin{aligned}\delta_{CUT}(q_{CUT},a) &= \delta(q,a) \text{ where } q \in Q \text{ and } q \notin F \\ &= \delta(q,a) \text{ where } q \in F\end{aligned}$$

Hence, from the above DFA it is clear that accepting states of both of the machines are same. So, the class of regular languages is closed under CUT operation.

Comment

Problem

Let $\Sigma = \{0,1\}$. Let $WW_k = \{ww \mid w \in \Sigma^*$ and w is of length k}.

a. Show that for each k, no DFA can recognize WW_k with fewer than 2^k states.

b. Describe a much smaller NFA for $\overline{WW_k}$, the complement of WW_k.

Step-by-step solution

Step 1 of 2

Consider the following language:

$WW_k = \{ww \mid w \in \Sigma^* \text{ and } w \text{ is of length } k\}$.

Therefore a string in WW_k language is at least of length k. Now consider two different k- bit strings $x = x_1 \ldots x_k$ and $y = y_1 \ldots y_k$, i be some location such that $x_i \neq y_i$.

Hence one of the strings contains a 1 at the i^{th} position where other contains a 0. Assume that $z = 0^{i-1}$.

Then z distinguishes x and y as exactly one of xz and yz has the k^{th} bit from the end as 1.

Since there exists, total 2^k binary strings of length k which can be mutually distinguish by the argument mentioned above, so any DFA for the given language has at least 2^k states.

Comment

Step 2 of 2

The language $\overline{WW_k}$ can be described as follows:

$\overline{WW_k} = \{ww \mid w \in \Sigma^* \text{ and } |w| < k\}$.

Therefore the user can build non deterministic finite automata with exactly $k+1$ states. This NFA can recognize the language $\overline{WW_k}$.

Assume that the non-deterministic finite automata consist of $Q = \{0,1,\ldots,k\}$ with the names of the state's corresponding to how many of the last k bits the NFA has seen.

Define $\delta(0,0) = 0, \delta(0,1) = 1$ and $\delta(i-1,0 \mid 1) = i$ for $2 \le i \le k$. We set $q_0 = 0$ and $F = \{k\}$

The machine starts from the initial state which is state 0, as it will traverse 1 initial state which is state 0, as it will traverse 1 it understand that it is the k^{th} bit from the finishing state or end state and proceed to state 1.

As it has reached on the state k, it accepts the string if and only if the string contains exactly $k-1$ bits. It checks for $k-1$ bits as it starts traversing from 0^{th} index.

Comments (5)

Problem

We define the ***avoids*** operation for languages A and B to be

A *avoids* B = {wl w ? A and w doesn't contain any string in B as a substring}.

Prove that the class of regular languages is closed under the *avoids* operation.

Step-by-step solution

Step 1 of 3

Regular Language:

• Regular language is the language which is generated or expressed with the help of regular expression and is recognizable by the finite automata.

• It is possible to get different types of regular languages from different types of regular grammar.

• By the definition of regular language it is known that if a finite automata recognizes a language then only it is said to be regular.

• Thus, in this case of **"avoids"** operation over two regular languages A and B

• If a finite automata accepting the language of **"avoids"** can be constructed

• Then it can be said that the set of regular languages is closed under the operation **"avoids".**

Comment

Step 2 of 3

As per given details there are two languages as *A* and *B* for which the avoid operations is defines as follows:

$$\text{A avoids B}=\{\text{w}\,|\,\text{w}\in\text{A and w does not contain any string in B as a substring}\}.$$

Here, it is required to prove that class of regular languages is closed under avoids operation.

Comment

Step 3 of 3

Proof:

• In the above:

(A avoids B) can be written as follows:

$$(\text{A}-(\text{A has B}))$$

Where (A has B) are string of A which contain strings of B as substrings.

• After that (A avoids B) will be strings of A that means strings of B will not contain B as substring.

• (A has B) can be written as follows:

$$(\text{A has B})\text{ as }\text{A}\cap(\Sigma^{*}\text{B}\Sigma^{*})$$

• After that (A has B) that means these are the strings of A that contains the strings of B as substring.

• Similarly, (A avoids B) can be written as follows:

$$(\text{A}-(\text{A}\cap(\Sigma^{*}\text{B}\Sigma^{*})))$$

Hence, the regular languages are closed under the concatenation, intersection, and subtraction and even closed under operation avoid.

Comments (1)

Problem

Let ∑ = {0,1}.

a. Let $A = \{0^k u 0^k \mid k \geq 1 \text{ and } u \in \Sigma^*\}$. Show that A is regular.

b. Let $B = \{0^k 1 u 0^k \mid k \geq 1 \text{ and } u \in \Sigma^*\}$. Show that B is not regular.

Step-by-step solution

Step 1 of 6

Pumping Lemma: There is an integer i for any language L which is regular in such a way that X belongs to L with $|X| \geq i$, There exits $p,q,r \in \sum^*$, such that $X = pqr$, and(a) $|pq| \leq i$ (b) $|q| > 0$ (c) $\forall n \geq 0 : pqr \varepsilon L$

Comment

Step 2 of 6

(a)

Consider the following details which is as follows:

Given language $A = \left\{0^k u\, 0^k \,\middle|\, k \geq 1 \text{and } u \in \Sigma^*\right\}$ and $\Sigma = \{0,1\}$

• In order to prove whether the given language is regular, the principle of pumping lemma is applied.

• Now, applying pumping lemma of regular language on the language A and let p be the minimum pumping length.

• Consider a string $\omega = 0^k u 0^k$ where ω is in language A of length *p* where p= $p = (2k + |u|)$

• Now, if ω is divided in xyz and it is known that $|xy| \leq p$ for $x = 0^k, y = u, z = 0^k$.

Comments (3)

Step 3 of 6

Case 1:

• If u=ξ

• Then language must be $0^k 0^k \in A$

$\omega = xy^i z$

$let\ i = 0$

$then\ \omega = 0^k \xi 0^k$

$= 0^k 0^k \in A$

Hence, for i=0 and ω is in specified language.

Comment

Step 4 of 6

Case 2:

$\omega = xyz$

$then\, x = z = 0^k$

$(|x| + |y| + |z|) <= p$

$k + |y| + k <= p$

$then |y| <= p - 2k$

• Let y=1 then the language will be:

$0^k 1 0^k \in A$

Hence, for y=1 and ω is in specified language.

Hence for any y belonging to $\sum^*$. So ω is in specified language.

Comment

Step 5 of 6

Case 3:

$\omega = xy^{p-2k}z$

$\omega = 0^k y^{p-2k} 0^k$

So, the y^{p-2k} always belongs to $\sum^*$

• So, from the pumping lemma it is proved that the language is regular as ω always belongs to specified language.

Comments (2)

Step 6 of 6

(b)

Consider the following details which is as follows:

Given language $B = \left\{0^k 1 u\, 0^k \middle| k \geq 1 \text{ and } u \in \Sigma^*\right\}$ and $\Sigma = \{0,1\}$

• In order to prove whether the given language is regular, the principle of pumping lemma is applied.

• Now, applying pumping lemma of regular language on the language B and let p be the minimum pumping length.

• Consider a string $\omega = 0^2 1 u 0^2$ where ω is in language B of length *p* for k=2.

• Now, if ω is divided in xyz and it is known that $|xy| \leq p$ for x=0, y= 0 and z=$1u0^2$

• Apply condition 1 in pumping lemma that is $xy^i z \in B$ for $i \geq 0$.

• Assume that i=2, so x=0 $y^2 = 00$ z=1 $u0^2$

• Now $xy^i z$ is 0001u0^2 and can be written as 0^{3}1u0^2

• It is clear that 0^{3}1u0^2 does not belong to language *B* because the value of k is not same for string $0^k 1 u\, 0^k$ I k≥1.

• So { $xy^i z = 0^3 1 u 0^2$ for i=2} $\notin$ *B*.

Hence, it is proved that *B* is a not a regular language.

Comments (2)

Problem

Let M_1 and M_2 be DFAs that have k_1 and k_2 states, respectively, and then let $U = L(M_1) \cup L(M_2)$.

a. Show that if $U \neq \emptyset$, then U contains some string s, where $|s| < \max(k_1, k_2)$.

b. Show that if $U \neq \Sigma^*$, then U excludes some string s, where $|s| < k_1 k_2$.

Step-by-step solution

Step 1 of 4

a)

Suppose M_1 and M_2 are taken as DFAs which consists k_1 and k_2 states respectively. Now, consider $U = L(M_1) \cup L(M_2)$. So, it can be shown that "if $U \neq \phi$, then some string s exists in U, where $|s| < \max(k_1, k_2)$".

Comment

Step 2 of 4

To prove the above statement first assume that the DFAs M_1 and M_2, which are taken above, consists strings of unequal length and also different from each other.

- Now consider a start state, for the given grammar U, q is taken. Then, the production of M_1 and M_2 with a start variable q is given by:
  $q \to sM_1 \mid sM_2$
- The string consists of unequal length and also different from each other and the start variable are initiated for the grammar.
- Then, from the property of union, it will take the longest string from the taken two strings and it will add the un-matched string or variable.

Therefore, the final string which is taken should be less than the any string taken. Thus it can be said that **"if $U \neq \phi$, then some string s exists in U, where $|s| < \max(k_1, k_2)$".**

Comment

Step 3 of 4

b)

Suppose M_1 and M_2 are taken as a DFAs which consists k_1 and k_2 states respectively. Now, consider $U = L(M_1) \cup L(M_2)$. So, it can be shown that" if $U \neq \Sigma^*$, then some string s can be excluded from U, where $|s| < k_1 k_2$".

Comment

To prove the above statement first assume that the DFAs M_1 and M_2, which are taken above, consists strings of unequal length and also different from each other.

- Now consider $U = L(M_1) \cup L(M_2)$, which says that U consists the string of M_1 and M_2. The string which is taken may be longer in length than M_1 and M_2.
- Here, concatenation operation is applied between the initial string k_1 and k_2, then the length it consists may be longer than the original length of the string.
- Some string also exists in this final concatenated string, which is already taken. Therefore, these strings will be excluded in the final string taken.

Hence from the above discussion it can be said that "**if** $U \neq \Sigma^*$**, then some string** s **can be excluded from** U **, where** $|s| < k_1 k_2$**".**

Comment

Problem

Let $\Sigma = \{0,1,\#\}$. Let $C = \{x\#x^{\mathcal{R}}\#x \mid x \in \{0,1\}^*\}$. Show that $\overline{C}$ is a CFL.

Step-by-step solution

Step 1 of 4

Suppose $\Sigma = \{0,1,\#\}$. Now consider a language C, where $C = \left\{x\#x^R\#x \mid x \in \{0,1\}^*\right\}$, then it has to show that $\overline{C}$ **is a context free language**. This can be achieved by showing the contradiction, that is, the language C is not a context free language.

Comment

Step 2 of 4

Each string $k = x\#x^R\#x \in C$, contains $|x| = |x^R|$, and then $|k|$ is multiple of three. For a contradiction suppose that C is context free. As it is considered as a context free, so it contains a pumping length l. Take $k = 0^{2l}0^l1^l0^{2l} \in C$ with $|k| > l$. Hence, there exists $abcde$ in such a way that:

1. $ab^mcd^me \in A$, for all $m \geq 0$.
2. $|bd| > 0$. It means that either b or d should not be empty string.
3. $|bcd| \leq l$. It means that length of the parts b, c, and d should be at most p.

Now, the following cases, which are given below, show that "the value taken for $abcde$ is independent and always **shows a contradiction**".

Comment

Step 3 of 4

Consider the different **cases** which are given below:

Case 1: $|bd|$ is not multiple of three. Then, $k' = ab^2cd^2e \notin C$ since $|k'|$ will no longer be multiple of three.

Case 2: Only 0s from the prefix set of 0s exists in bcd and $|bd| = 3q$ for some q. Then, $ab^2cd^2e = 0^{3l+3q}1^l0^{2l} = 0^{2l+q}0^{l+2q}1^{l-q}1^q0^{2l} \notin C$. Since $x = 0^{2l+q}$ and $x^R \neq 1^q0^{2l}$, the **resultant third of the string**.

Case 3: bcd has only 1s and $|bd| = 3q$ for some q. Then, the **resultant third string** is given by $ab^2cd^2e = 0^{3l}1^{l+3q}0^{2l} = 0^{2l+q}0^{l-q}1^{l+2q}1^q0^{2l} \notin C$.

Case 4: Only 0s from the suffix set of 0s exists in bcd and $|bd| = 3q$ for some q. Then, the **resultant third string** is given by, $ab^0cd^0e = 0^{3l}1^l0^{2l-3q} = 0^{2l-q}0^{l+q}1^{l-2q}1^{2q}0^{2l-3q} \notin C$, since $x = 0^{2l-q}$ and $x^R \neq 1^{2q}0^{2l-3q}$.

Comment

Step 4 of 4

Now, if $i < l+2$, $q = 1$ is taken then, $ab^icd^ie = 0^{3l}1^{l+3(i-1)}0^{2l+(i-1)} = 0^{2l+(i-1)}0^{l-(i-1)}1^{l+2(i-1)} \in C$, Since there are not enough 1s to force then into x and $l+2$ is first time is guaranteed. Hence, there is a **contradiction occurred in each case**. Thus C **is not a context free language**. In other words, it can be said that "$\overline{C}$ **is a context free language**".