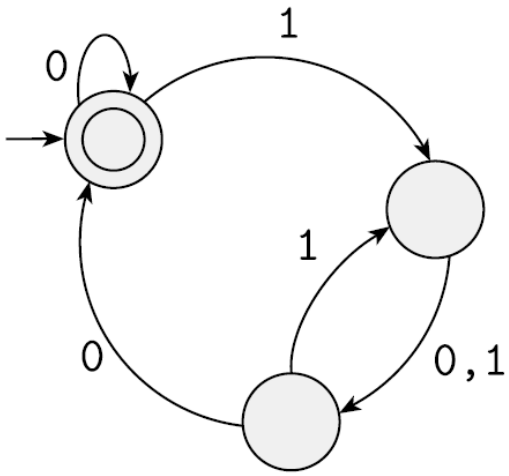
Problem

Answer all parts for the following DFA M and give reasons for your answers.



a. Is $\langle M, 0100 \rangle \in A_{\text{DFA}}$?

b. Is $\langle M, 011 \rangle \in A_{\text{DFA}}$?

c. Is $\langle M \rangle \in A_{\text{DFA}}$?

d. Is $\langle M, 0100 \rangle \in A_{\text{REX}}$?

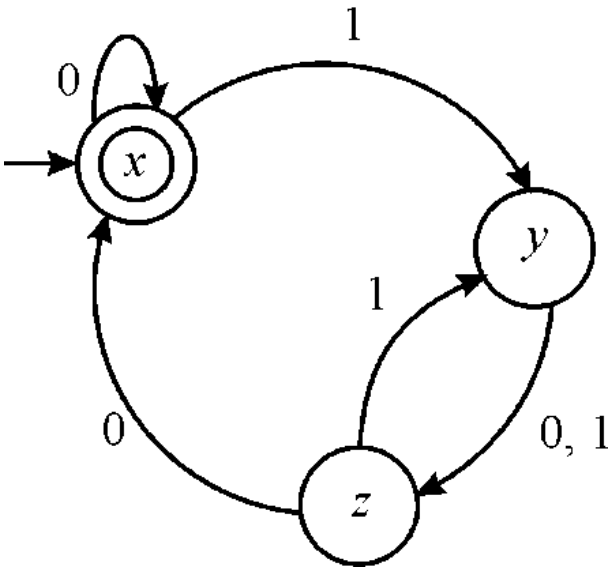
e. Is $\langle M \rangle \in E_{\text{DFA}}$?

f. Is $\langle M, M \rangle \in EQ_{\text{DFA}}$?

Step-by-step solution

Step 1 of 7

Consider the following figure:



As it is already given *M* is a DFA.

Theorem: A_{DFA} is a language which test whether given DFA *M* accepts provided string *w* or not. Use the following steps to check DFA accepted given string or not.

Consider a Turing machine *W* which decides A_{DFA}.

$W = \text{On input } \langle M, w \rangle, \text{ where } M \text{ is a DFA and } w \text{ is string}$

1. On *w* input simulate *M*.

2. When simulation ends at accept states then it is **accepted.** When simulation does not end at accept states then it is **rejected**.

Comment

Step 2 of 7

Use above theorem to check whether *M* accepts string 0100 or not. Simulate, DFA *M* on 0100. Consider the figure shown above for the production of *M*.

• As shown in above figure the starting and final (accepting) state of *M* is *x*. After taking the first input 0 from string 0100 the next state is still *x*.

• After taking the second input 1 from string 0100 the next state is *y*. similarly after taking fourth '0' input from 0100 the next state becomes *x* which is an acceptable state.

Hence, *M* accepts input string 0100.

Comment

Step 3 of 7

Use above theorem to check whether *M* accepts string 011 or not. Simulate, DFA *M* on 011. Consider the figure shown above for the production of *M*.

• As shown in above figure the starting and final (accepting) state of *M* is *x*. After taking the first input 0 from string 011 the next state is still *x*.

• After taking the second input 1 from string 011 the next state is *y*. similarly after taking third '0' input from 011 the next state becomes *z* which is not an acceptable state.

Hence, *M* does not accept input string 011.

Comments (1)

Step 4 of 7

Use above theorem to check whether *M* accepts given string or not. Here input string *w* is not given in A_{DFA}. Here A_{DFA} is not in proper format.

Hence, *M* does not accept input string.

Comment

Step 5 of 7

Consider a language A_{REX} which is used to determine whether particular regular expression can generate provided string or not. Consider a Turing machine *W* which decides whether regular expression can generate provided string or not.

$W = \text{On input } \langle R, w \rangle, \text{ where R is a Regular expression and } w \text{ is string}$

Here, in this part *M* is a DFA instead of regular expression but for A_{REX} it should be regular expression to check whether it can generate string 0100 or not.

Hence, *M* cannot generate string 0100.

Comment

Step 6 of 7

Use marking algorithm E_{DFA} to check, whether DFA accept any string or not. Use the following steps to check DFA accepted any string or not.

Consider a Turing machine *W* which decides E_{DFA}.

W = On input $\langle M \rangle$, where M is a DFA

1. Marks the initial state of *M* hat is *x*.
2. Repeat the state till new state is not marked.
3. Mark the next state of transition which comes from any other marked states.
4. If accept state is not marked then **accept**, else **reject**.

Consider the transition diagram shown above. In that diagram the initial and final state are same that is *x*. so when applying the first step of E_{DFA} theorem then initial as well as final states both get marked. According to step 4 of above theorem it is not acceptable when final states is marked.

Hence, *M* cannot accept any string. M is not an empty language.

Comments (5)

Step 7 of 7

Consider theorem EQ_{DFA} to determine whether two DFA identify the same language.

Consider a Turing machine *W* which decides EQ_{DFA}.

Here, in this part $L(M) = L(M)$ therefore derived language $L(C)$ becomes empty.

Therefore Turing machine W accepts the derived language $L(C)$.

Hence, yes the language of *M* accepts itself.

Comment

Problem

Consider the problem of determining whether a DFA and a regular expression are equivalent. Express this problem as a language and show that it is decidable.

Step-by-step solution

Step 1 of 2

Proof of the decidability of the language:

• Express the language as $L = \langle R, S \rangle |$ R is a Deterministic Finite Automata(DFA) and S is a regular expression with $L(R) = L(S)$)}.

• Recollect the Theorem 4.5 states a Turing machine *T* that decides the language EQ_{DFA} = { $\langle P, Q \rangle$ | P and Q are Deterministic Finite Automata's(DFA) $L(P) = L(Q)$ }.

Comment

Step 2 of 2

• Assume that *T* is the Turing Machine which decides language *L*.

• It can be defined as follows:

• T = "On input $L = \langle R, S \rangle$, where *R* is a Deterministic Finite Automata(DFA) and *S* is a regular expression:

• Convert R into a Deterministic Finite Automata(DFA) D_R using the algorithm in the proof of Kleene's Theorem.

• Operate a Turing machine *TM* as a decider *F* using Theorem 4.5 on input $\langle R, D_S \rangle$.

• If *F* accepts, accept the language *L*.

• If *F* rejects, reject the language *L*.

Comments (2)

Problem

Let $ALL_{\mathrm{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \Sigma^*\}.$ Show that ALL_{DFA} is decidable.

Step-by-step solution

Step 1 of 4

A DFA (Deterministic Finite Automaton) starts travelling, via arrows of the DFA, from the start state to the accept state and when it reaches an accept state, it accepts some string.

Comment

Step 2 of 4

Consider the following details:

$ALL_{\mathrm{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \Sigma^*\}$

Prove that ALL_{DFA} is decidable.

Comment

Step 3 of 4

A is a DFA that accepts every possible permutation and combination of its input string. Thus, its DFA has only a single state $q0$, which is both initial and final state.

Comments (3)

Step 4 of 4

So, on executing the Turing machine 'T' on INPUT(*A*):

• Mark the initial state of *A*.

• Repeat until no new states gets marked:

• The state that has any transition coming into it from any other already marked state will be marked.

• **ACCEPT**: when all the accept states are marked, otherwise **REJECT.**

Comments (4)

Problem

Let

$$A\varepsilon_{\mathrm{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG that generates } \varepsilon\}.$$ Show that $A\varepsilon_{\mathrm{CFG}}$

is decidable.

Step-by-step solution

Step 1 of 1

Decidability of the language

Given: In this a language $A_{\varepsilon CFG}$ is given.

Proof: For showing that the language $A_{\varepsilon CFG}$ is decidable, build a Turing machine T for deciding the language $A_{\varepsilon CFG}$. For all Context free grammars G

• If the grammar G derives ε then $T(\langle G\rangle)$ accepts

• If the grammar G does not derive $\in$ then $T(\langle G\rangle)$ rejects.

Constructions:

For proofing the decidability of $A_{\varepsilon CFG}$ firstly convert the context free grammar G into an equivalent G' in CNF. If $S \to \varepsilon$ is the rule in the CFG G' then it means that G' derives ε.

If the CFG G' derives ε then G also derives it as $L(G) = L(G')$. As G' is in CNF so only possible $\in$-rule in G' is $S \to \varepsilon$. If G' contains $S \to \varepsilon$ in production rules then $\varepsilon \in L(G')$. If G' does not contains the rule $S \to \varepsilon$ then $\varepsilon \notin L(G')$.

Turing machine T = on input $\langle G\rangle$ where G is a context free grammar

• Convert the grammar G in CFG G'.

• If G' contains the production rule $S \to \varepsilon$ then accept it.

• Otherwise reject it.

Conclusion:

From the above construction it is clear that $\langle G\rangle \in A_{\varepsilon CFG}$ iff $\langle G, \varepsilon\rangle$ is also belongs to the A_{CFG}. So the above construction is correct. Hence the language $A_{\varepsilon CFG}$ is decidable.

Comment

Problem

Let $E_{\mathrm{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$. Show that $\overline{E_{\mathrm{TM}}}$, the complement of E_{TM}, is Turing-recognizable.

Step-by-step solution

Step 1 of 1

Re-cognition of the language by Turing machine

Given: In this a language E_{TM} is given. Show that the complement of E_{TM} which is written as $\overline{E_{TM}}$ is recognized by Turing machine M.

Proof: Assume that $t_1, t_2, t_3, \ldots$ is a list of strings which are present in Σ^*. For proving $\overline{E_{TM}}$ is Turing recognizable, user has to determine whether any of the string from $t_1, t_2, t_3, \ldots$ is accepted by the Turing machine M or not.

If the Turing machine M accepts at least one string t_i from the list $L(M) \neq \phi$ so the Turing machine M belongs to $\overline{E_{TM}}$.

If the Turing machine M does not accept any string then $L(M) = \phi$ so the Turing machine M does not belong to $\overline{E_{TM}}$.

List of all strings cannot sequentially execute on the Turing machine M as if the Turing machine M can accept the string $\langle M\rangle \in \overline{E_{TM}}$ but it is looping in string t_1. Since the Turing machine M accept the string t_2 from the list so $\langle M\rangle \in \overline{E_{TM}}$.

If list of all the strings is executed on M in a sequential manner then user never extract the past of first string.

So for avoiding this problem related to sequential execution and recognizing $\overline{E_{TM}}$ construct a Turing machine.

Construction of Turing machine M:

Turing machine S= on input $\langle M\rangle$ here M is a Turing machine.

1. Repeat the following methods for $i = 1, 2, 3, \ldots$

2. Execute each and every string from the list $t_1, t_2, t_3, \ldots$ on the Turing machine M.

3. If any string is accepted then accept it

4. Otherwise reject it.

Conclusion:

Hence the string is accepted by the Turing machine so the complement of E_{TM} which is written as $\overline{E_{TM}}$ is Turing recognizable.

Comment

Problem

Let X be the set {1, 2, 3, 4, 5} and Y be the set {6, 7, 8, 9, 10}. We describe the functions f : X → Y and g : X → Y in the following tables. Answer each part and give a reason for each negative answer.

n	$f(n)$
1	6
2	7
3	6
4	7
5	6

n	$g(n)$
1	10
2	9
3	8
4	7
5	6

A**a.** Is f one-to-one?

b. Is f onto?

c. Is f a correspondence?

A**d.** Is g one-to-one?

e. Is g onto?

f. Is g a correspondence?

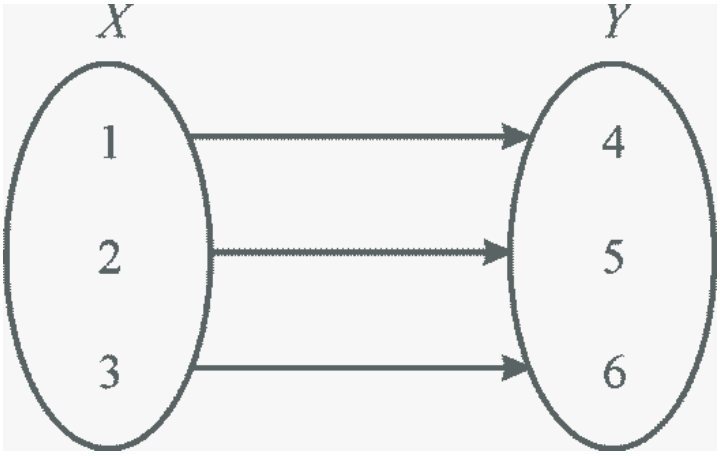
Step-by-step solution

Step 1 of 7

Given that X={1,2,3,4,5} and Y={6,7,8,9,10} .

a. **One-to-one Function:** A function $f : X \to Y$ is said to be one to one if and only if when for each and every member of X there exist a unique matching member Y. It is also known as injective function.

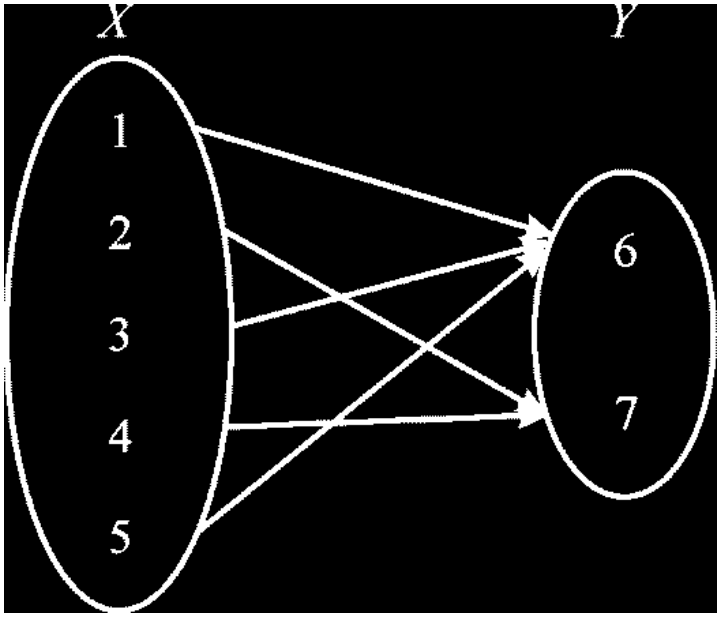
For example:



Here in the above diagram, for every member of X there exists a unique matching member Y. Thus, the above function is one-to-one.

Now, consider the first table given in the textbook draw the diagram for showing the matching between the members.

Consider the following diagram:



Here, in the above diagram for every member of X there exists a matching member $Y = f(X)$.

But in this case:

$f(1) = f(3) = f(5) = 6$

$f(2) = f(4) = 7$

So the function f is not a one-to-one function.

Comment

Step 2 of 7

b. **Onto function:**

A function $f : X \to Y$ is said to be onto if and only if when for each and every member of $y \in Y$ there exist a matching member $x \in X$ with the property that $y = f(x)$ where $x \in X, y \in Y$.

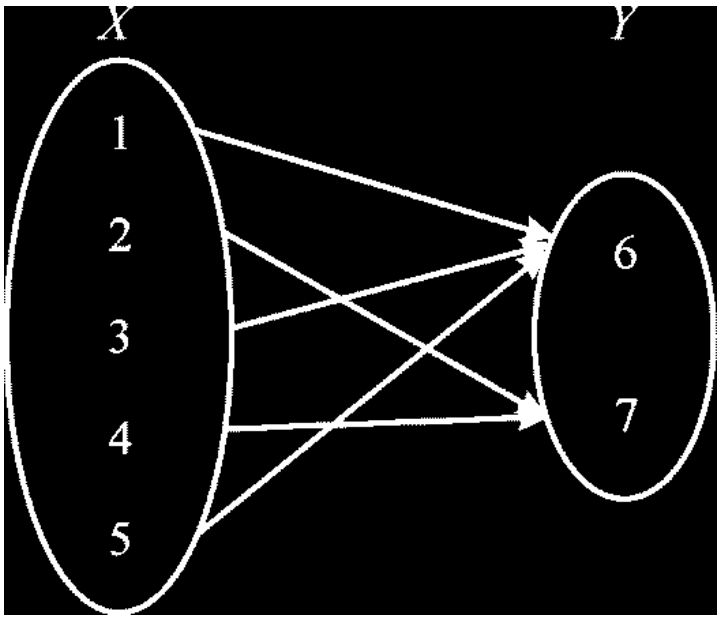
In other words it can be described as follows:

The size of f(x) should be equal to size of the set Y.

It is also known as surjective function.

Now, consider the first table given in the textbook draw the diagram for showing the matching between the members.

Consider the following diagram:



Here, in the above for each and every member of $y \in Y$ there does not exist a matching member $x \in X$ with the property that $y = f(x)$ where $x \in X, y \in Y$.

$f(1) = f(3) = f(5) = 6$
$f(2) = f(4) = 7$

and the elements 8, 9 and 10 in set Y are left out without any elements in $x \in X$ such that $y = f(x)$

So the function f is not an onto function.

Comments (6)

Step 3 of 7

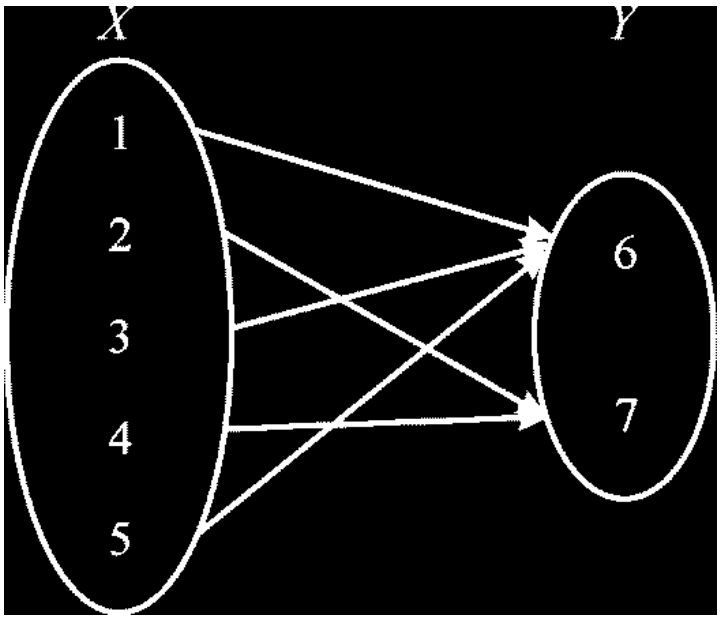
c. **Correspondence:** A function $f : X \to Y$ is said to be one to one correspondences when this function is one to one and onto at the same time. It is also known as bijective function.

Now, consider the first table given in the textbook draw the diagram for showing the correspondence between them.

Consider the following diagram:

Comment

Step 4 of 7



It is clear from the above diagram that the function $f : X \to Y$ is not a one-to-one and not onto. So the function $f : X \to Y$ is not one-to-one and not onto function.

So, the function f is not a correspondence.

Comment

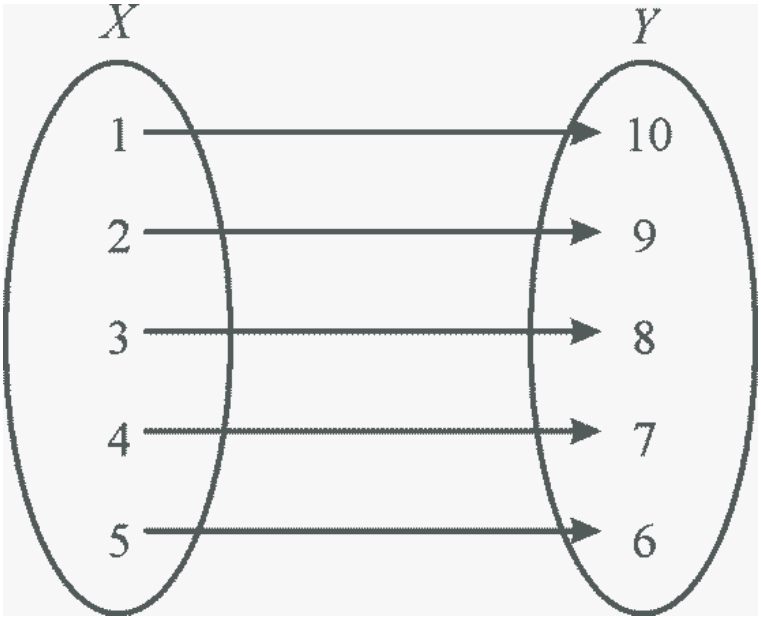
Step 5 of 7

d. **One-to-one Function:** A function $g : X \to Y$ is said to be one-to-one if and only if when for each and every member of X there exist a unique matching member Y.

It is also known as injective function.

Now, consider the first table given in the textbook draw the diagram for showing the matching between the members.

Consider the following diagram:



Here in the above diagram, for every member of X there exists a unique matching member $Y = f(X)$.

So, the function g is a one-to-one function.

Comment

Step 6 of 7

e. **Onto function:**

A function $f : X \to Y$ is said to be onto if and only if when for each and every member of $y \in Y$ there exist a matching member $x \in X$ with the property that $y = f(x)$ where $x \in X, y \in Y$.

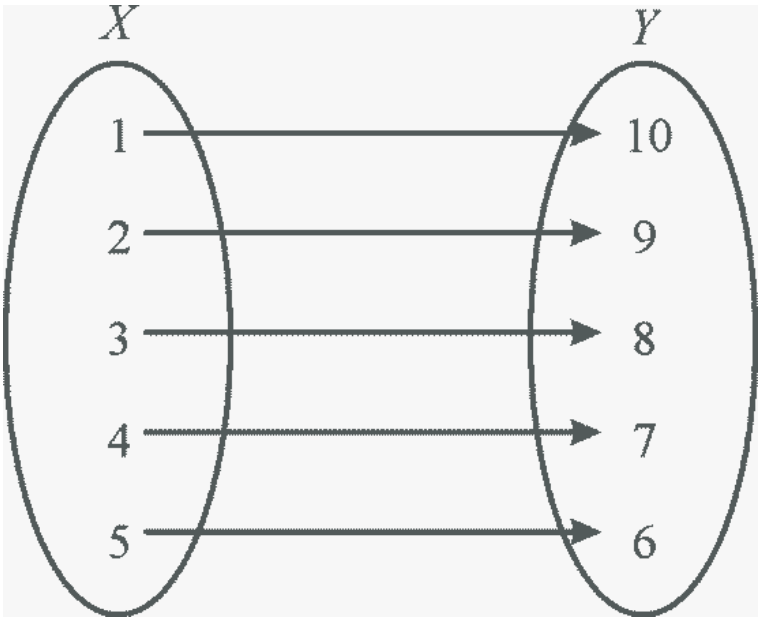
In other words it can be described as follows:

The size of f(x) should be equal to size of the set Y.

It is also known as surjective function.

Now, consider the first table given in the textbook draw the diagram for showing the matching between the members.

Consider the following diagram:



Here in the above diagram, for every member of X, there exists a matching member Y with the property $y = f(x)$.

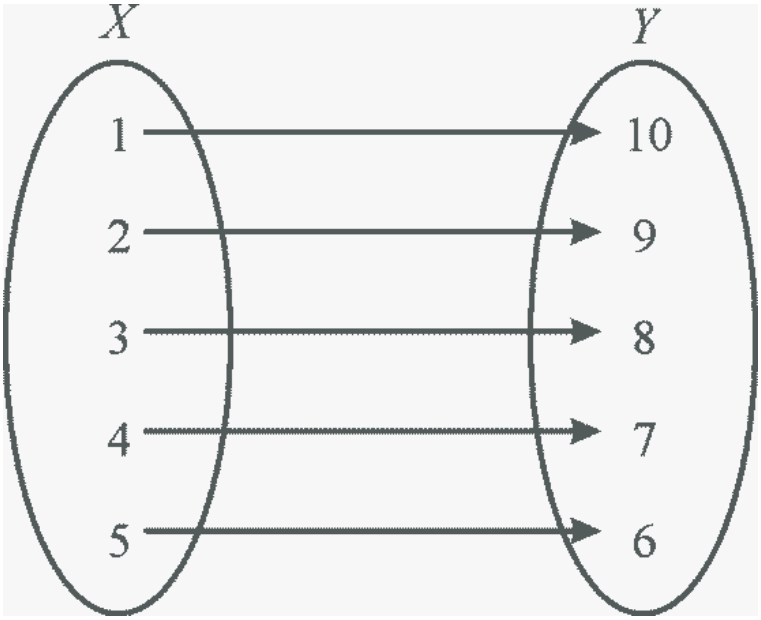
So the function g is a onto function

Comment

Correspondence: A function $g : X \to Y$ is said to be one to one correspondences when this function is one to one and onto at the same time. It is also known as bijective function.

Now, consider the first table given in the textbook draw the diagram for showing the correspondence between them.

Consider the following diagram:



It is clear from the above diagram that the function $g : X \to Y$ is one to one and onto at the same time.

So the function g is a correspondence.

Comment

Problem

Next

Let β be the set of all infinite sequences over {0,1}. Show that β is uncountable using a proof by diagonalization.

Step-by-step solution

Step 1 of 3

Let $\mathcal{B}$ be the set of all infinite sequences over {0,1}. Every element in $\mathcal{B}$ is an infinite sequence $(a_1, a_2, a_3, \ldots)$ where $a_i \in \{0,1\}$. Assume $\mathcal{B}$ is countable. Define a correspondence *f* between $\mathcal{B}$ and $N = \{1,2,3,\ldots\}$.

Comment

Step 2 of 3

Suppose, for $z \in N$, $f(z) = (a_{z1}, a_{z2}, a_{z3}, \ldots)$ where a_{zi} is defined as the i^{th} bit in the z^{th} sequence.

In other words,

z	$f(z)$
1	$(a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, \ldots)$
2	$(a_{21}, a_{22}, a_{23}, a_{24}, a_{25}, \ldots)$
3	$(a_{31}, a_{32}, a_{33}, a_{34}, a_{35}, \ldots)$
4	$(a_{41}, a_{42}, a_{43}, a_{41}, a_{45}, \ldots)$
$\vdots$	$\vdots$

Comment

Step 3 of 3

Now, a sequence b is defined in such a way that $b = (b_1, b_2, b_3, \ldots)$ belongs to $\mathcal{B}$ over $\{0,1\}$ where $b_i = 1 - a_{ii}$ for $i \in N$.

Consider the following example,

z	$f(z)$
1	$(0,1,1,0,0,\ldots)$
2	$(1,0,1,0,1,\ldots)$
3	$(1,1,1,1,1,\ldots)$
4	$(1,0,0,1,0,\ldots)$
$\vdots$	$\vdots$

The sequence *b* is computed as $b = (1-a_{11}, 1-a_{22}, 1-a_{33}, 1-a_{44}, \ldots) = (1,1,0,0,\ldots)$. Therefore, $b \in \mathcal{B}$ is different from every sequence by minimum one bit. Thus, b is not equal to $f(z)$ for any z. It is a contradiction that $\mathcal{B}$ is uncountable.

Therefore, $\mathcal{B}$ is uncountable.

Comment

Problem

Let $T = \{(i, j, k) | \ i, j, k \in \mathcal{N}\}.$ Show that T is countable.

Step-by-step solution

Step 1 of 5

Countability: A set is countable if a set is either finite or has the same cardinality as the set of positive integers.

Comment

Step 2 of 5

Given $T = \{\langle i, j, k\rangle | i, j, k \in N\}$

• The goal is to prove that T is a countable set.

• First, let's define a set P = $\{\langle i, j, k\rangle \in T \mid i+j+k = s\}$ for each triple $\langle i, j, k\rangle$ where $i, j, k \in N$, let $i+j+k$ be the sum s of the triplet.

• Now, for each number $s \in N$,

• There are finitely many triples that has sum equal to s.

• Enumerating the triples with sum zero, then triples with sum equal to 1, then sum equal to 2 and so on.

• The previous step will follow all the triples in T.

• Hence, set P is finite for every $s \in N$.

Comment

Step 3 of 5

Now, since P is finite and according to the given definition, it is countable too, therefore the set $P' = \{\langle i', j', k'\rangle \in T \mid i'+j'+k' = s'\}$ is also countable.

Comment

Step 4 of 5

Therefore, any set P_i where $i \in N$, the union $T = \bigcup_{i \in N} P_i$ is also countable since, a countable union of a number of finite sets is countable.

Comment

Step 5 of 5

Hence, it is proved that T is countable.

Comment

Problem

Review the way that we define sets to be the same size in Definition 4.12 (page 203). Show that "is the same size" is an equivalence relation.

DEFINITION 4.12

Assume that we have sets A and B and a function f from A to B. Say that f is ***one-to-one*** if it never maps two different elements to the same place—that is, if $f(a) \neq f(b)$ whenever $a \neq b$. Say that f is ***onto*** if it hits every element of B—that is, if for every $b \in B$ there is an $a \in A$ such that $f(a) = b$. Say that A and B are the ***same size*** if there is a one-to-one, onto function $f: A \longrightarrow B$. A function that is both one-to-one and onto is called a ***correspondence***. In a correspondence, every element of A maps to a unique element of B and each element of B has a unique element of A mapping to it. A correspondence is simply a way of pairing the elements of A with the elements of B.

Step-by-step solution

Step 1 of 3

Given:

A function $f: A \to B$ in which A and B are two sets. The function $f: A \to B$ is **one to one function** as it never maps two different elements of same set with one element of another set.

The function $f: A \to B$ is **onto function** as each and every element of the set B is hit by the element of set A.

As the function $f: A \to B$ is one to one and onto at the same time then it means the set A and B has the same cardinality. If the cardinality of these two sets is same so these sets are of **same size or equinumerous**.

If the function $f: A \to B$ is one to one and onto at the same time it means the function $f: A \to B$ is **correspondence function** also. **Correspondence function** is also known as **bijective function**.

If the function $f: A \to B$ is one to one and onto or bijective function, then sets A and B are of **same size**.

Comment

Step 2 of 3

Proof:

Equivalence relation: A relation is known as equivalence in nature if it is reflexive, transitive and symmetric.

Same size relation is equivalence relation if and only if it is symmetric, reflexive and transitive.

• **For reflexivity:** If the user checks the identity function on the set A then this identity function is a bijection from A to A.

Hence the **same size** relation is reflexive relation.

• **For symmetry: if the function** $f: A \to B$ is a bijective function then it means the inverse function f^{-1} is also bijective function from the set B to set A.

Hence if $A \sim B$ then $B \sim A$, so **the same size** relation is symmetric relation also.

• **For transitivty: Assume that** $A \sim B$ and $B \sim C$. Then the function $f: A \to B$ is bijective function from A to B and the function $g: B \to C$ from B to C.

Therefore the composition of two bijective functions f and g is also a bijective function from A to C.

Hence the **same size** relation is transitive relation as $A \sim C$.

Comment

Step 3 of 3

Conclusion:

Hence the **same size** relation is reflexive, symmetric and transitive in nature so the **same size** relation is equivalence relation.

Comment

Problem

Let $INFINITE_{DFA} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A)$ is an infinite language$\}$. Show that $INFINITE_{DFA}$ is decidable.

Step-by-step solution

Step 1 of 1

Decidability of Infinite $_{DFA}$ can be proved as follows:

For deciding $INFINITE_{DFA}$ construct a Turing machine $T1$.

T1= "On input $\langle A\rangle$ *, where A is* Deterministic Finite Automata.

1. Assume the value of *n* as the number of states of A.

2. Build Deterministic Finite Automata *M1* that accepts all the strings of length *n* or more.

3. Build a Deterministic Finite Automata *M2* such that *L (M2) = L (A)* $\cap$ *L (M1)*

4. Test *L (M2) =* ϕ using the E_{DFA} decider *T* from the theorem 4.4.

5. If *T* accepts reject; if *T* rejects, accept."

The above algorithm functions because a Deterministic Finite Automata that accepts infinitely many strings must accept randomly long strings.

Hence, the algorithm recognizes such Deterministic Finite Automata.

Contrariwise, if the algorithm recognizes a Deterministic Finite Automata, the Deterministic Finite Automata recognizes certain strings of length *n* or more where *n* is the number of states of the DFA.

This string may be pumped in the method of the pumping lemma in favor of the regular languages to find several recognized strings.

Since Turing machine $T1$ decides $INFINITE_{DFA}$ so $INFINITE_{DFA}$ is decidable.

Comment

Problem

Let $INFINITE_{PDA}$ $= \{\langle M\rangle \mid M \text{ is a PDA and } L(M) \text{ is an infinite language}\}.$ Show that $INFINITE_{PDA}$ is decidable.

Step-by-step solution

Step 1 of 1

Given that

$Infinite_{PDA} = \{ \langle M \rangle \mid \text{M is a PDA and L(M) is an infinite language } \}$

The decidability of *Infinite* $_{PDA}$ is as follows:

• To decide the $Infinite_{PDA}$ convert the given PDA *M* into a CFG *G*.

• Now convert into an equivalent grammar G' in Chomsky Normal Form (CNF).

• Generate a graph A1: Each Variable is a vertex, and for a rule or production

• $T \to UV$ add edges (T, U) from T to U. and (T, V) from T to V.

• Apply DFS or BFS from the start state to see if A1 has a directed cycle. If it does, accept. Otherwise reject.

• If $\langle M \rangle \in \text{INFINITE}_{\text{PDA}}$, then it has a string of length greater than pumping length of L (M).

• Following the proof of pumping lemma, it means there is a Variable V which can derive sVt for some strings s, t. (Since it is in CNF it must be non-empty).

• This implies that the graph A1 has a cycle involving the variable V.

• Assume there is a cycle in A1. Then it is clear that for some variable V, a derivation $V \to {}^*sVt$ must exist and s or t must be a non-empty, since G' is in CNF.

• Since there is a scope for finding a cycle from S. there is rule $S \to {}^*aVb.$

• Thus, $S \to {}^*au^iVv^ib.$ for all i>=0 and so L(M) is infinite.

Hence, basing on the above discussion it is proved that $Infinite_{PDA}$ is decidable.

Comment

Problem

Let A = { $\langle M \rangle \mid M$ is a DFA that doesn't accept any string containing an odd number of 1s}. Show that A is decidable.

Step-by-step solution

Step 1 of 2

Consider a language,

$A = \{< M > \mid M \text{ is a DFA which does not accept any string containing an odd number of 1s}\}$

The language is said to be decidable if there exists a Turing machine for it. Construct a Turing machine for A to check the decidability.

Comment

Step 2 of 2

The Turing machine for A is as follows:

$I =$ "On input $<M>$ where M is a DFA:

1. Construct a new DFA D_X that accepts any string containing an odd number of 1s.
2. Construct another DFA D_Y such that $L(D_Y) = L(M) \cap L(D_X)$.
3. Check whether $L(D_Y) = \phi$, using the E_{DFA} decider T.
4. If T accepts, accept; otherwise reject."

There exists a Turing machine for A. Therefore, A is decidable.

Comment

Problem

Let A = { $\langle R, S \rangle |$ } Show that A is decidable.

Step-by-step solution

Step 1 of 3

The idea of the proof is as follows:

• L(R) is a subset of L(S) iff L(R) intersected with the complement of L(S), $L(S)^C$ is the empty set.

• Use the Theorem 4.4 (or its equivalent for regular expressions) to prove that

the condition "L(R) intersection $L(S)^C$ = empty set" is decidable.

Comment

Step 2 of 3

Proof:

The following Turing machine T decides the language A.

"On input string *w*,

1. Check that *w* encodes a pair $< R, S >$ where R and S are regular expressions

If it does not, then reject *w*.

2. Translate R into an equivalent DFA DR and S into an equivalent DFA DS.

3. Build a DFA DS^C that accepts the language $L(DS)^C$.

4. Build a DFA D that is the intersection of DR with DS^C that is, L(D) = L(DR) intersection $L(DS^C)$.

5. Now, run the Turing machine T with input < D > to determine if L(D) is empty.

If T accepts < D > (which means that L (D) is empty), then accept *w*.

If T rejects < D > (which means that L (D) is NOT empty), then reject *w*."

Comment

Step 3 of 3

Now prove that the Turing machine T decides the language A.

• First of all, prove that T halts on all inputs.

• Let *w* be any word. There are two possibilities.

• Either *w* codifies a pair of regular expressions or it doesn't. Checking this will take a finite amount of time.

• If x doesn't codify a pair of regular expressions, then T stops rejecting *w*.

• Otherwise, assume *w* codifies a pair < R, S >.

• Constructing the DFAs described in steps 2, 3, and 4 above is possible (regular languages are closed under intersection and complementation) and takes finite time.

Hence, in all cases T stops on any input *w* after a finite amount of time.

Hence, it is proved that language A is decidable.

Comments (2)

Problem

Let ∑ = {0,1}. Show that the problem of determining whether a CFG generates some string in 1^* is decidable. In other words, show that

$$\{\langle G\rangle \mid G \text{ is a CFG over } \{0,1\} \text{ and } 1^* \cap L(G) \neq \emptyset\}$$

is a decidable language.

Step-by-step solution

Step 1 of 3

• Given a CFG over $(0,1)$ that generates **some** string in 1^*. The language generated by this CFG, denoted by Language A can be rephrased in the following way: $A=\{<G>|G \text{ is a CFG over } (0,1)^* \text{ and } 1^* \cap L(G) \neq \emptyset\}$.

• **To test decidability, the fact that intersection of a Regular Language (RL) & Context Free Language (CFL) is a CFL shall be harnessed.**

Comment

Step 2 of 3

• Note that 1^* is a Regular Language & $L(G)$ is a CFL (since G is a CFG), therefore, $1^* \cap L(G)$ is clearly a CFL.

• A Turing Machine (TM) has the task to test whether the problem at hand is decidable or undecidable.

• If for an input, TM culminates in either ACCEPT or REJECT state, the problem is referred to as **decidable**.

Comment

Step 3 of 3

Let H be the TM that decides A. Using the Theorem 4.8, the decidability is employed by the following algorithm:

H="on input <G>, where G is a CFG":

Construct a CFG B, such that $L(B) = 1^* \cap L(G)$ (remember that $1^* \cap L(G)$ is CFL so the statement is valid).

1. Run the TM R that decides E_{CFG} on $< B >$. It may be elaborated here that E_{CFG} denotes the problem of determining whether a CFG (here B) generates any strings at all is decidable. Formally,

$$E_{CFG} = \{< B > \mid B \text{ is a CFG and } L(B) = \emptyset\}$$

2. **If R accepts, reject**. It means that the language generated by the CFG B is empty. Therefore, TM H culminates in REJECT state.

3. **If R rejects, accept**. In other words, the problem at hand is Decidable since language generated by the CFG *B* is NOT empty.

Elaboration: If TM R accepts, it would imply that,

$$\begin{aligned} L(B) &= 1^* \cap L(G) \\ &= \emptyset \end{aligned}$$

That, is for some string $w \in 1^*$, $w \notin L(G)$, hence R should reject.

Comment

Problem

Show that the problem of determining whether a CFG generates all strings in 1^* is decidable. In other words, show that

$$\{\langle G\rangle \mid G \text{ is a CFG over } \{0,1\} \text{ and } 1^* \subseteq L(G)\}$$

is a decidable language.

Step-by-step solution

Step 1 of 1

Given:

$S = \{\langle G\rangle \mid G\}$ is a context free grammar over the string $\{0,1\}$ and $1^* \subseteq L(G)$

User need to proof that string 1^* is generated by CFG and it is decidable.

Proof:

First the assumption is made that grammar *G* is in the CNF. If it is not present in the CNF then Turing machine should be use to make the grammar in CNF.

Suppose there is total *n* variable in the grammar *G* and the grammar there is one pumping lemma constant in grammar. Assume pumping lemma constant is *s* and the value of this pumping lemma constant is assumed to be 2^{n-1}.

It is prove that if the value of *k* is greater than or equal to *s*. If the grammar *G* can generate 1^k then it would be capable of generating 1^{k+s}.

Grammar *G* is capable of generating string which is in the form 1^m for $0 \le m < s! + s$ then G is use for generating Σ^*.

Turing machine is use for verifying that whether the string is accepted or not. If the string is recognized then Turing machine accepts that particular string otherwise Turing machine reject that particular string.

Construction:

If the Grammar *G* is use for generating 1^k then the grammar *G* should also generates $1^{k+p!}$

With the help of the lemma 1^k is divided into the multiple string named *a*, *b*, *c*, *d* and *e* in such a manner that *abcde*=1^k. $|bcd| \le s, |bd| > 0$, $ab^i cd^i e \in L(G)$ for all the value of *i* which is greater than or equal to zero.

Suppose, $h = |bd|$ in such a way that $1 \le h \le s$.

With the help of lemma $1^{k+ih} \in L(G)$ for the integer *i* greater than or equal to zero.

Hence, $1^{k+ih} \in L(G)$ it implies grammar G generates 1^{k+ih}.

Conclusion:

Turing Machine is working as a decider for CFG and deciding whether the string is accepted or not by using the predefined rules mentioned above.

Hence, it can be said that the Turing machine *S* is decidable.

Comment

Problem

Let A = { $\langle R \rangle \mid R$ is a regular expression describing a language containing at least one string w that has 111 as a substring (i.e., w = x111y for some x and y)}. Show that A is decidable.

Step-by-step solution

Step 1 of 3

Consider the language,

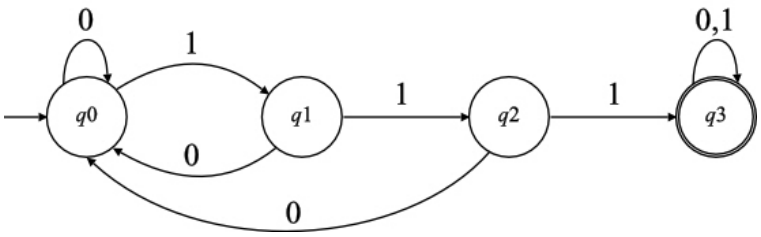
$$A = \left\{ \langle R \rangle \mid R \text{ is a regular expression describing a language which contain at least one string } w \text{ containing } 111 \text{ as its substring} \right\}$$

The decidability of the language *A* is proved as follows:

• Define a language *S* such that $S = \{w \in \Sigma^* \mid w \text{ consists } 111 \text{ as a substring}\}$.

• The regular expression (RE) for the language *S* is $(0 \cup 1)^* 111 (0 \cup 1)^*$.

• The DFA D_S for the language *S* is shown below:



Comment

Step 2 of 3

• Now think about some RE *R* on input alphabet Σ.

• If $S \cap L(R) \neq \phi$, then *R* produces a string containing 111 as a substring. Thus, $\langle R \rangle \in A$.

• Similarly, if $S \cap L(R) \neq \phi$ then R produces a string that does not contain 111. Thus, $\langle R \rangle$ does not belongs to *A*.

• Since $L(R)$ is described by regular language, $L(R)$ is a regular language. Both *S* and $L(R)$ are regular languages.

• $S \cap L(R)$ is regular because, regular languages are closed under intersection. Thus, $S \cap L(R)$ has some DFA $D_{S \cap L(R)}$.

• Theorem 4.4 shows that $E_{DFA} = \{\langle K \rangle \mid K \text{ is a DFA with } L(K) \neq \phi\}$ is decidable. Thus, there exists a Turing Machine *TM* which determines E_{DFA}.

• Relate TM *T* to $D_{S \cap L(R)}$ to determine if $L(R) \cap S \neq \phi$.

Comments (2)

Step 3 of 3

Summarization of the above discussion contributes the subsequent Turing machine *M* to decide *A*:

M = "On input $\langle R \rangle$, where R is a regular expression:

• Transform *R* into a DFA D_R by means of the algorithm in the proof of Kleene's Theorem.

• Build a DFA $D_{S \cap L(R)}$ for the language $S \cap L(R)$ from the DFAs D_S and D_R.

• Run TM *T* that decides E_{DFA} on input $\langle D_{S \cap L(R)} \rangle$.

• If T accepts, reject. If T rejects, accept.

The Turing machine T decides A . Therefore, the language A is decidable.

Comment

Problem

Prove that EQ_{DFA} is decidable by testing the two DFAs on all strings up to a certain size. Calculate a size that works.

Step-by-step solution

Step 1 of 1

Given:

Assume that $EQ_{DFA} = \{\langle A, B\rangle \mid A, B \text{ are } DFAs \text{ and } L(A) = L(B)\}$ and suppose M is a Turing machine for determining the decidability of EQ_{DFA}.

Construction of Turing machine M :

• M = "On input $\langle A, B\rangle$, where A and B are DFAs with the same terminal symbols. If symbols are not same then reject.

• Calculate the number of states in the DFA, A and B and stored them in n and m respectively.

• Iterate all strings which comes under Σ till $n.m$

• Now for each and every string w

o Simulate DFA A on the string w

o Simulate DFA B on the string w

o Here Turing Machine M is working as a decider by running on the output provided both DFA A and B. If result of both the simulation comes different the reject the string, otherwise accept it.

• After that all $n.m$ strings then accept it otherwise reject it.

• If M accepts, accept. If M rejects, reject."

Size for working:

The reason behind checking first $n.m$ strings is that if both DFAs do not accept the same language so there is a string w of size $|w|$ and it is less than equals to size of $n.m$ for $A(w) \neq B(w)$.

By using contradiction, suppose that the first string provides a different output of DFAs A and B is w'. The length l of $|w'|$ is greater than $n.m$.

Now the sequence of states for the DFA A is $a_0, a_1, a_2 \ldots\ldots a_l$ and the sequence of states for the DFA B is $b_0, b_1, b_2 \ldots\ldots b_l$ for describing the transitions for w' in DFA A and B.

As the value of l is greater than $n.m$, if the user places above sequences side by side so there is some repetition in pairs like a_i, b_i and a_j, b_j such that $a_i = a_j, b_i = b_j, i < j$ also present.

So user can remove all the sequences remaining a_i, b_i and get a smaller string w''. As the DFA A and B is same over as w' thus our contradiction becomes false.

As the string w'' has length less than equal to $n.m$ such that $A(w'') \neq B(w'')$.

So, checking all the strings up to size $n.m$ is enough.

Conclusion:

Here two Turing Machines are used M is used as decider to find decidability of A and B Turing Machine M is used to find Running condition of Turing machines M. This way A and B are being decided by M, A and B are components of EQ_{DFA} so, EQ_{DFA} is also decidable.

Comments (1)

Problem

Let C be a language. Prove that C is Turing-recognizable iff a decidable language D exists such that

$$C = \{x | \exists y \ (\langle x, y \rangle \in D)\}.$$

Step-by-step solution

Step 1 of 1

Given: A language C, and this language is Turing recognizable if and only if there exist y such that $\langle x, y \rangle \in D$.

Proof:

Assume that a language C. This language is recognized by a Turing machine M. When the input x is passed to the Turing machine this machine simulates machine M_d. The decider for the language D on the input string $\langle x, y_i \rangle$, here $y_1, y_2, y_3 \ldots\ldots$ string in lexicographic order.

Assume that the language C is recognizable. Suppose $L(M) = C$ which means that Turing machine M recognize the language C. If $L(M) = C$ then for every string x which belongs to the language C, an accepting computation of M on the input string x is present.

Suppose the language D on input $\langle x, y \rangle$ verifies whether y encodes an accepting computation of the input string x on machine M.

Example of such encoding is $c_0 \# c_1 \# c_2 \ldots\ldots \# c_n$

Here in the above encoding c_i's configurations of the Turing machine M on the input string x. c_0 Is the initial configuration of the input string x on machine M. The configuration c_n is the accepting configuration for input x. After each c_i, c_{i+1} come.

So it is clearly seen that D is decider.

Construction:

Here user supposed to construct Turing machine that will decide the decidability of C. Now follow these terms:

• For proving decidability of C one needs a Turing Machine so consider a Turing Machine T.

• Construct Turing Machine in a way so that each possible string of Y can be searched or found.

• Test the string whether it is according to the predefined rules $C = \{x | \exists y \langle x, y \rangle \in D\}$ in the question.

• If $\langle x, y \rangle \in D$ then T accepts otherwise rejects.

Conclusion:

Here D is Recognizable by C and C is recognizable by Turing machine T and C is also decided by Turing Machine T so $\langle x, y \rangle \in C$ as well this way C is Turing recognizable.

Comment

Problem

Prove that the class of decidable languages is not closed under homomorphism.

Step-by-step solution

Step 1 of 1

Given:

Consider a decidable language *K* and a homomorphism *m* to show that $m(K)$ is un-decidable.

Proof:

• Assume that $K = \left\{ \begin{array}{l} uv \mid u \in \{0,1\}^*, v \in \{a,b\}^*, u = \langle W, w \rangle, \\ \text{and } v \text{ encodes an integer } n \\ \text{using turing machine } W \text{ having input } w \text{ get halts in } n \text{ steps} \end{array} \right\}$

• As K is decidable so simulate Turing machine *W* on input *w* for *n* steps.

• Consider homomorphism $m(0) = 0$, $m(1) = 1$ and $m(a) = m(b) = \varepsilon$

• $m(K) = \text{HALT}$ which is un-decidable.

Hence, Decidable language *K* is not closed under homomorphism *m*.

Comment

Problem

Let A and B be two disjoint languages. Say that language C ***separates*** A and B if $A \subseteq C$ and $B \subseteq \overline{C}$. Show that any two disjoint co-Turing-recognizable languages are separable by some decidable language.

Step-by-step solution

Step 1 of 1

Given:

Two disjoint languages are supposed to be *A* and *B*. Decidable Language *C* is chosen so that it is use for separating *A* and *B* if $A \subseteq C and B \subseteq \bar{C}$. Now, according to this concept two disjoint Co-Turing recognizable languages can be separated with the help of decidable language.

Proof:

Consider two DFA *M* and *N* for languages *A* and *B*. *O* is a DFA for separator *C*. Here language *A* is assumed to the subset of *C* and language *B* is assumed to be the subset of $\bar{C}$. In this way *C* is separating *A* and *B* because *C* or $\bar{C}$ is not deciding both *A* and *B* instead *A* is decided by *C* and *B* is decided by $\bar{C}$. It can be understood that *C* and $\bar{C}$ both are dissimilar and deciding A and B individually.

Construction:

Concept can be understood in better by using following approach:

Turing Machine *F* = runs on input *> where A and B are two DFA*

• Construction of DFA O is done for C. DFA O is working as a decider for language A and B.

• Run Turing Machine T on input of C. It will find the decidability capacity of O.

• If Turing machine accept the string then it is recognized and if Turing machine reject the string then it is not recognized.

Conclusion:

Now, if T accepts then DFA O for language C is working as a decider for language A otherwise it is working as decider for language B.

Now, it is quite easy to understand two disjoint Co-Turing recognizable languages which are separable by the use of some decidable language.

Comments (8)

Problem

Let S = { $\langle M \rangle |$ M is a DFA that accepts w^R whenever it accepts w}. Show that S is decidable.

Step-by-step solution

Step 1 of 4

Given: $S = \{\langle M \rangle | M \text{ is a DFA that accepts } w^R \text{ whenever it accepts } w\}$.

Here, M is a DFA that accepts w^R whenever it accepts w and M is recognizable and decidable on input w.

Note: - If a DFA accepts w^R whenever it accepts w, then L(M) = L(M^R), where M^R is the DFA that accepts the reverse of the strings accepted by M.

Comment

Step 2 of 4

Proof that S is decidable is as follows:

Consider the following Turing Machine T = "On Input M, Where M is a DFA".

1) Construct DFA N which accepts the reverse of a string accepted by M.

2) Submit to the Decider for EQ_{DFA}.

3) If it accepts, accept.

4) If it rejects, reject.

T is a Decider since, steps 1, 3 and 4 will not create an infinite loop and step-2 calls a decider. Also, T accept M which is a DFA if L(M) = L(M^R).

Therefore, T decides S. Thus, S is decidable.

Comment

Step 3 of 4

Construction:

DFA M^R can be constructed by first constructing NFA from M by reversing all transition in the following way:

• Change the initial state with new accepting state.

• After that, create a new initial start state with $\in$ transition to all earlier accepting state.

• Then construct a DFA from this NFA.

As, a DFA can accept only those particular languages that they are designed for, T is deciding the decidability of M. Decidability or Undecidability of a string depends upon recognition of its components.

It is eventually necessary that output of M is decidable by S. So, from the above proof it is clear that the language S is decidable.

Comment

Step 4 of 4

Conclusion:

It is eventually necessary that output of M is decidable by S. So, from the above proof it is clear that the language S is decidable.

Comment

Problem

Let *PREFIX-FREE*$_{REX}$ = { $\langle R \rangle$ | R is a regular expression and L(R) is prefix-free}. Show that *PREFIX-FREE*$_{REX}$ is decidable. Why does a similar approach fail to show that *PREFIX-FREE*$_{CFG}$ is decidable?

Step-by-step solution

Step 1 of 3

The prefix free of regular expression is defined as,

$$PREFIX-FREE_{REX} = \{\langle R \rangle \mid R \text{ is a regular expression and } L(R) \text{ is prefix free}\}$$

The string *x* is said to be a prefix of a string *y* if there exists a string *z* such that *xz*=*y*. If $x \neq y$ then *x* is said to be the proper prefix of *y*. A language is said to be prefix free if none of the strings have proper prefix of its members (i.e., strings).

Comment

Step 2 of 3

If there exists a TM for any language, then it said to be decidable. Construct the Turing machine *M* to check the decidability of the language.

M = "On input R where R is a regular expression

1. If R is not a valid regular expression then reject.
2. Construct a DFA (Deterministic Finite Automata) for the language accepted by R i.e., L(R).
3. Run the DFS (Depth First Search) from the start state q_0 and remove all the unreachable states from it.
4. For each final state q_f, run the DFS to check if there exists path to any other final state from it or any self loop to itself.
5. If there exists a path to another final state or self loop from q_f then reject. Otherwise, accept."

Thus, there exists a TM for $PREFIX-FREE_{REX}$. Therefore, it is decidable.

Comment

Step 3 of 3

The DFA accepts the prefix free language if and only if there is no accepting string goes through more than one final state. It can be checked by simple graph traversal technique like DFS. In this case, the DFA cannot have two final states for an accepting run. It is decidable.

In the case of context free grammar (CFG), it fails because every context free language cannot have deterministic PDA. In case of regular expressions, every regular expression has a DFA. If the PDA cannot be drawn deterministically for a language, then it cannot be restricted to have a single final state. Therefore, the similar approach fails to prove $PREFIX-FREE_{CFG}$ is decidable.

Comment

Problem

Say that an NFA is ***ambiguous*** if it accepts some string along two different computation branches. Let $AMBIG_{NFA}$ = { $\langle N \rangle$ | N is an ambiguous NFA}. Show that $AMBIG_{NFA}$ is decidable. (Suggestion: One elegant way to solve this problem is to construct a suitable DFA and then run E_{DFA} on it.)

Step-by-step solution

Step 1 of 2

$AMBIG_{NFA}$ **= { is ambiguous NFA} is decidable**

An NFA is ambiguous if it accept string and two or more different computational branches. Decidability of NFA's Ambiguity can be proved by using following approach.

Now, for a given NFA create a DFA D by using strategy NFA-to-DFA conversion which simulates N and accept a string only if it is accepted by NFA and computed using 2 or more computational branches. Now create decider for DFA E for determining whether D accepts any string accepted by NFA N.

Comment

Step 2 of 2

Simulate N by keeping the pebble on active state. Initially, put red pebble on initial state along $\in$ transaction. As per N's transaction, operations like moving, adding and removing are performed, preserving color of pebbles. Whenever 2 or more pebbles moved to same state replace its pebble with blue pebble. After that input is examined and accepted when either the state of blue pebble is accept state or different accept states have red pebbles.

For state of N, the pebbles have corresponding positions of DFA D. There will be three stages in which position contain blue pebble, red pebble or no pebble. On the basis of these stages, ambiguity is generated then it is decidable by recognizing multiple pebbles for same string. If there are multiple pebbles then it is ambiguous and decidable by multiple pebbles and more than 1 red pebbles.

Comment

Problem

A ***useless state*** in a pushdown automaton is never entered on any input string. Consider the problem of determining whether a pushdown automaton has any useless states. Formulate this problem as a language and show that it is decidable.

Step-by-step solution

Step 1 of 4

Useless states in automata are the **states whose removal from the automata does not make any difference or put any impact on the language accepted by the automaton**.

Given: A pushdown automaton.

Comment

Step 2 of 4

Here, it is required to test whether the machine has useless states or not and whether the problem of testing is decidable or not.

Comment

Step 3 of 4

• Let *P* be the set of all the strings accepted by a pushdown automaton.

• Let the language $L = \{x \in P \mid x \text{ contains a useless state}\}$.

• To show that language *L* is decidable, construct a Turing machine which accepts strings in language *L*.

• In reference to this book, consider that the question of whether a PDA has an empty language is decidable.

• It can reduce the question of whether a given state *n* is useless to this question by making *n* the only accept state and then determine whether the resulting push down automata has an empty language.

• If it does then, *n* is a useless state.

Comment

Step 4 of 4

Hence, our Turing machine successfully decides whether there is any useless state, by performing this test for each and every state in order.

Comments (2)

Problem

Let BAL_{DFA} = { ‹M›| M is a DFA that accepts some string containing an equal number of 0s and 1s}. Show that BAL_{DFA} is decidable. (Hint: Theorems about CFLs are helpful here.)

Step-by-step solution

Step 1 of 1

Given:

M Is a DFA, accepts a string which contains equal number of 0s and 1s.

Proof:

For proving BAL_{DFA} is decidable use the concept of context free language which is as follows:

Now, consider a string w that contains equal no of 0s and 1s that is accepted by DFA M. If a regular language is Context Free Language then it is decidable and it is decidable then it would be recognizable as well.

Now use the approach in the same way as uses for CFL.

String containing equal no of 0s and 1s can be generated by a particular grammar pattern or can be said by using a Context Free Grammar then it'd be a Context Free Language.

Now, it is already known that each and every context free language is decidable and it can be proved as follows:

Here M is a DFA that accepts string w that contains equal no of 0s and 1s consider A be a CFG that generates w.

Construction:

Consider a Turing machine that decides w, and built a copy of w in Turing machine A. This process works as follows:

T = Runs on input w

• Run Turing Machine S on input $\langle M, w \rangle$

Here, consider another Turing machine that generates output by considering input from M by running it on input w

• If machine accepts, accept; if rejects, reject

Conclusion:

The above Turing machine decides w running on M, this way BAL_{DFA} is decidable.

Comment

Problem

Let

$$PAL_{\text{DFA}} = \{\langle M\rangle \mid M \text{ is a DFA that accepts some palindrome}\}.$$

Show that PAL_{DFA} is decidable. (Hint: Theorems about CFLs are helpful here.)

Step-by-step solution

Step 1 of 4

Assume that $PAL_{DFA} = \{\langle M\rangle \mid M \text{ is a } DFA \text{ that accepts some palindrome}\}$. If a Turing machine can be presented for the given DFA that runs finitely and halts, then the PAL_{DFA} is decidable.

Comment

Step 2 of 4

Construct a decider *D* for PAL_{DFA} and a Turing machine K that can decides E_{CFG} :

D = "On input $\langle M\rangle$,

1. A PDA *P* is constructed as: $L(P) = \{w \mid w \text{ is a palindrome}\}$

2. A PDA *P'* is constructed so that $L(P') = L(P) \cap L(M)$

3. Now P' is converted into an equivalent CFG *G*.

4. Check if $L(G)$ is empty using Theorem 4.8 over Turing Machine K.

5. If $L(G)$ is empty then reject, otherwise, accept.

Comment

Step 3 of 4

For Turing machine K:

• Both steps 1 and 2 can be done in finite steps.

• Step 3 also takes finite steps to convert P' into its equivalent CFG.

• In step 4, the decider *K* checks whether the language $L(G)$ is empty or not. It can also be done in a finite step.

Comment

Step 4 of 4

Since D takes finite steps for any input, it means that it is a decider. Hence, PAL_{DFA} is decidable.

Comment

Problem

Let

$$E = \{\langle M \rangle | \ M \text{ is a DFA that accepts some string with more 1s than 0s}\}.$$

Show that E is decidable. (Hint: Theorems about CFLs are helpful here.)

Step-by-step solution

Step 1 of 1

Proving the decidability of the language

Given:

$E = \{\langle M \rangle | \ M\}$, is a DFA which is use for accepting a string *w* which contain more number of 1s and 0s.

Proof:

In order to prove that *E* is decidable, context free language should be use. As it is well known that if any string contain more number of 1's and 0's then it is context free language.

Now, consider a string *w* that contains more number of 1s than 0s that is accepted by DFA *M*. If a regular language is Context Free Language then it is decidable and it is decidable then it would be recognizable as well.

Now use the approach in the same way as uses for CFL.

String containing more number of 1s than 0s can be generated by a particular grammar pattern or can be said by using a Context Free Grammar then it would be a Context Free Language.

Now, it is already known that each and every context free language is decidable. Now prove that the context free language is decidable.

Construction:

M is a DFA that accepts string *w* that contains more number of 1s than 0s. Consider *A* be a CFG that generates *w*. Consider a Turing Machine that decides *w*. and built a copy of *w* in Turing Machine *A*.

Working of the Turing machine process is as shown:

T = Runs on input *w*

• Run Turing Machine *S* on input $\langle M,w \rangle$. Here, consider another Turing machine that generates output by considering input from *M* by running it on input *w*.

• If the string is accepted, then Turing machine accept that particular string.

• If the string is not accepted, the Turing machine rejects the particular string.

Conclusion:

So, by the above construction it is proved that string is decided by the Turing machine. Hence, it is said that the language *E* is decidable.

Comment

Problem

Let

$$C = \{\langle G, x\rangle \mid G \text{ is a CFG } x \text{ is a substring of some } y \in L(G)\}.$$

Show that C is decidable. (Hint: An elegant solution to this problem uses the decider for E_{CFG}.)

Step-by-step solution

Step 1 of 3

Suppose $C = \{\langle G, x\rangle \mid G \text{ is a CFG } x \text{ is a substring of some } y \in L(G)\}$. Now consider the following proof which shows that C is decidable.

A Turing machine M is constructed in such a way that decides C as follows:

• A DFA *A* is constructed in such a way that it is used to recognize that language of the regular expression $\sum^* \circ \{x\} \circ \sum^*$ (every string with *x* as their substring).

• A DFA *F* is constructed which is used for the context free language $L(G) \cap L(A)$.

• Perform simulation on the Turing machine that decides E_{CFG} on $L(F)$. If it accept, reject, otherwise accept.

Comments (1)

Step 2 of 3

It is already know that a language is also a context free language if it is an intersection of a regular language and a context free language. Therefore, *F* will be CFG.

• Furthermore, $L(A)$ is a language of every strings with *x* as their substring and it is described above that it is also a regular language.

• So, **if *G* produces some string *y* with *x* as its substring, the intersection, $L(F)$, should be non-empty.**

Comment

Step 3 of 3

Now it can be concluded from the above that ***C* is decidable.**

Comment

Problem

Let C_{CFG} = $\{\langle G, k\rangle |$ G is a CFG and L(G) contains exactly k strings where k ≥ 0 or k = 1}. Show that C_{CFG} is decidable.

Step-by-step solution

Step 1 of 1

Decidability

Consider a decider *M* which is used to check whether language of CFG is finite or infinite. Use another decider that is Turing machine *W* which shows that is C_{CFG} decidable.

1. *W* = "on input $\langle G,k\rangle$" where *G* is CFG and *k* is string

2. Check $L(G)$ is infinite using decider *M.*

• If $L(G)$ is infinite and $k=\infty$, it is accepted

• If $L(G)$ is infinite and $k\neq\infty$, it is rejected

• If $L(G)$ is finite and $k=\infty$, it is rejected

• If $L(G)$ is finite and $k\neq\infty$, continue

3. Calculate the pumping length *l* for grammar *G.*

4. Set $count=0$

5. Use *for* loop $i=0$ to l

• Use *for* loop to get all strings *S* whose length equal to *i*

• If *S* can be generated by *G* then make an increment in *count.*

6. Check value of *count* is equal to *k* then it is accept, otherwise reject.

Explanation:

• The Step 2 checks whether $L(G)$ is infinite or not. After step 2 there is grammar whose language which has finite set. In order to prove C_{CFG} is decidable there is only need to prove that the size of language is *k.*

• To do so use loop to find the all the possible string can be generated by grammar G. The grammar is finite therefore the length of string cannot be more than pumping length *l.*

• Make an increment in variable count if the string can be generated by grammar *G.*

• In the last step check value of *count* is equal to *k.*

• Now, it has finite number of steps therefore it can easily check.

Thus *W* is decider, therefore C_{CFG} is also decidable language.

Comments (2)

Problem

Let A be a Turing-recognizable language consisting of descriptions of Turing machines, $\{\langle M_1\rangle, \langle M_2\rangle, \ldots\}$, where every M_i is a decider. Prove that some decidable language D is not decided by any decider M_i whose description appears in A.

(Hint: You may find it helpful to consider an enumerator for A.)

Step-by-step solution

Step 1 of 2

Consider the Turing-recognizable language *A* which contains the descriptions of all the Turing machines, therefore there must exist an enumerator *E* to enumerate it.

Consider $\langle M_i\rangle$ is the i^{th} output of *E*. Assume $s_1, s_2, s_3 \ldots, s_i$ are the all possible strings of $\{0,1\}^*$. It means $s_1, s_2, s_3 \ldots, s_i$ are made up of combinations of 0's and 1's.

Comment

Step 2 of 2

Consider a decidable language *D* is defined as follows:

For a string S_i,

• If $\langle M_i\rangle$ accepts then S_i does not belongs to the language *D.*

• If $\langle M_i\rangle$ rejects then S_i belongs to the language *D.*

Here, the language *D* is a decidable language and its decider is not present in the list. Therefore, it is proved that there is a decidable language *D* whose decider is not present in *A*.

Comment

Problem

Next

Say that a variable A in CFL G is ***usable*** if it appears in some derivation of some string w ? G. Given a CFG G and a variable A, consider the problem of testing whether A is usable. Formulate this problem as a language and show that it is decidable.

Step-by-step solution

Step 1 of 3

Consider the variable *A* in CFG *G* is usable if it appears in some derivation of some string $w \in L(G)$.

Proof:

Let's consider the problem to find whether *A* is usable and formulate this problem as a language and show that it is decidable. The language is:

$USABLE_{CFG} = \{\langle G, A\rangle : G \text{ is a CFG}, A \text{ usable for } G\}$.

Comment

Step 2 of 3

For CFG *G*, *A* variable is usable if the derivation of: $s \Rightarrow *xAy \text{ of } G$ such that $L(x), L(A) \text{ and } L(y)$ are all nonempty, is exist.

Decidability of language can be proved with the help of some steps. There are two further steps required to find decidability of the language. Now, according to the question:

• Consider, a usable CFG is A_{CFG} that generates decidable language.

• If usable A_{CFG} is decidable then language will be decidable as well.

Comment

Step 3 of 3

Construction:

Now construct and use the approach to find the decidability of A_{CFG} and to find decidability of the language.

Consider a decider *M* for A_{CFG} and $L(x), L(A) \text{ and } L(y)$ to decide *M* and also assume a Turing Machine *R* that will decide, whether output provided by M_1 can be accepted or not.

$B = "on\ input < M >, where\ M\ is\ a\ CFG"$, *M* is another CFG that is working as a decider for A_{CFG}, Now

• Run Turing Machine *R* on input $<M,M_1>$, where M_1 is a CFG that generates $\in^*$, and working as decider for *M1*.

• If R acceptable then accepts, else reject.

Conclusion:

Here, it is quite obvious M_1 is working as a decider for *M* and *M* is working as a decider for A_{CFG} so A_{CFG} is decidable and then language produced by A_{CFG} will also be accepted by Turing machine so it will also be decidable.

Comment

Problem

The proof of Lemma 2.41 says that (q, x) is a *looping situation* for a DPDA P if when P is started in state q with x ? Γ

Step-by-step solution

Step 1 of 4

A language Q is said to be **decidable** if a Turing machine (which is also called as decider) M exists in which Q **is accepted** and **halts on every string input.** A language, that is decidable, is also known as a recursive language**.**

• Now, consider a string l, then for any string l , there exist the following condition:

- $l \in Q \Rightarrow M$, **Halts in a state of accepting**.
- $l \notin Q \Rightarrow M$, **halts in a state of non-accepting.**

It is also known that “each language which is decidable is also **Turing-acceptable**”.

Comment

Step 2 of 4

It is already known that “A looping condition for a DPDA (Deterministic Push Down Automata) P is (q,x) if state q is the starting point of P with $x \in \mathrm{T}$ which exists on the stack’s top and everything under x is never popped. Also, an input symbol is never be read”.

• The above given problem of looping can be solved by identifying the looping situation. In other words,” only in which no further inserted symbol is ever read and re-programming DPDA. Therefore, instead of looping, it rejects and reads the input.

Comment

Step 3 of 4

The situation arises above, produces a halt situation. The following algorithm explained it more carefully:

T =”On input $\langle B \rangle$, where B is DFA:

1. Initial state of B is marked.

2. Repeat this till a marked on new state:

3. Mark a position on which a **transition exists** in to it from anyother state which is marked already.

4. If none of the state is marked as accepted state, it will be **accepted**; otherwise, **rejected**”

The above algorithm produces a halt situation for accept and non-accept state.

Comments (3)

Step 4 of 4

Now consider **a language** F **, which is defined as** $F = \{\langle P,q,x \rangle | (q,x)$ **is a situation of looping for** $P\ \}$**.** Therefore, from the above discussion it can be said that **“the above given language** F **is a decidable language”.**

Comment